



MobiLink Synchronization User's Guide

Last modified: October 2002
Part Number: 38132-01-0802-01

Copyright © 1989–2002 Sybase, Inc. Portions copyright © 2001–2002 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-FORMS, APT-Library, APT-Translator, ASEP, 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, Dynamo, e-ADK, E-Anywhere, e-Biz Integrator, E-Whatever, EC-GATEWAY, ECMAP, ECRT, eFulfillment Accelerator, Electronic Case Management, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, 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 Server, First Impression, Formula One, Gateway Manager, GeoPoint, iAnywhere, iAnywhere Solutions, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InstaHelp, Intellidex, InternetBuilder, iremote, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, Logical Memory Manager, MainframeConnect, Maintenance Express, MAP, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, MethodSet, ML Query, MobiCATS, 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, Rapport, Relational Beans, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Report Workbench, Report-Execute, Resource Manager, RW-DisplayLib, RW-Library, S Designer, S-Designer, 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, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Viewer, Visual Components, VisualSpeller, 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 © 1997–2000 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.

Last modified October 2002. Part number 38132-01-0802-01.

Contents

| | | |
|---------------------|--|-------------|
| | About This Manual..... | xiii |
| | SQL Anywhere Studio documentation..... | xiv |
| | Documentation conventions..... | xvii |
| | The sample database..... | xx |
| | Finding out more and providing feedback..... | xxi |
| PART ONE | | |
| | Using MobiLink Technology | 1 |
| 1 | Introducing MobiLink Synchronization | 3 |
| | The MobiLink synchronization process..... | 4 |
| | MobiLink terminology | 7 |
| 2 | Synchronization Basics | 9 |
| | Parts of the synchronization system | 10 |
| | The consolidated database..... | 12 |
| | The MobiLink synchronization server | 18 |
| | MobiLink clients..... | 21 |
| | The synchronization process | 24 |
| | Options for writing synchronization logic | 38 |
| | Character set considerations | 42 |
| | Security | 45 |
| 3 | Writing Synchronization Scripts..... | 47 |
| | Introduction to synchronization scripts..... | 48 |
| | Scripts and the synchronization process | 53 |
| | Script types | 55 |
| | Script parameters..... | 60 |
| | Script versions..... | 61 |
| | Adding and deleting scripts in your consolidated database | 63 |
| | Writing scripts to upload rows | 66 |
| | Writing scripts to download rows | 70 |
| | Writing scripts to handle errors | 75 |

| | | |
|----------|--|------------|
| | Example scripts for UltraLite | 77 |
| | Testing script syntax | 78 |
| | DBMS-dependent scripts | 80 |
| 4 | Synchronization Techniques | 83 |
| | Introduction | 84 |
| | Development tips | 85 |
| | Timestamp-based synchronization | 86 |
| | Snapshot synchronization | 88 |
| | Partitioning rows among remote databases | 91 |
| | Maintaining unique primary keys | 95 |
| | Handling conflicts | 104 |
| | Data entry | 110 |
| | Handling deletes | 111 |
| | Handling failed downloads | 112 |
| | Downloading a result set from a stored procedure call | 113 |
| | Schema changes in remote databases | 116 |
| 5 | Adaptive Server Anywhere Clients | 117 |
| | Creating a remote database | 118 |
| | Publishing data | 119 |
| | Creating MobiLink users | 125 |
| | Subscribing MobiLink synchronization users | 128 |
| | Differences from version 7 | 132 |
| | Initiating synchronization | 138 |
| | Using ActiveSync synchronization | 143 |
| | Deploying remote databases | 148 |
| | Partitioning data between remote databases | 155 |
| | Temporarily stopping synchronization of deletes | 156 |
| | Customizing the client synchronization process | 157 |
| 6 | Writing Synchronization Scripts in Java | 165 |
| | Introduction | 166 |
| | Setting up Java synchronization logic | 167 |
| | Running Java synchronization logic | 169 |
| | Writing Java synchronization logic | 170 |
| | Sample: Java synchronization logic | 177 |
| | MobiLink Java API Reference | 183 |
| 7 | Writing Synchronization Scripts in .NET | 187 |
| | Introduction | 188 |
| | Setting up .NET synchronization logic | 189 |

| | | |
|-----------|---|------------|
| | Running .NET synchronization logic | 191 |
| | Writing .NET synchronization logic | 194 |
| | .NET synchronization example | 200 |
| | MobiLink .NET API Reference | 203 |
| 8 | MobiLink Performance | 219 |
| | Performance tips | 220 |
| | Key factors influencing MobiLink performance | 224 |
| | Monitoring MobiLink performance..... | 229 |
| 9 | MobiLink Monitor | 231 |
| | Introduction | 232 |
| | Starting the MobiLink Monitor | 233 |
| | Using the MobiLink Monitor | 235 |
| | Saving Monitor data | 243 |
| | Customizing your statistics | 244 |
| | Statistical Properties | 247 |
| 10 | Authenticating MobiLink Users | 251 |
| | About MobiLink users | 252 |
| | Choosing a user authentication mechanism..... | 254 |
| | User authentication architecture | 255 |
| | Providing initial passwords for users..... | 257 |
| | Synchronizations from new users | 258 |
| | Prompting end users to enter passwords | 259 |
| | Changing passwords | 260 |
| | Custom user authentication mechanisms | 261 |
| 11 | Synchronizing Through a Web Server | 263 |
| | Introduction | 264 |
| | Configuring MobiLink clients and servers for the Redirector..... | 266 |
| | Configuring the Redirector (all versions) | 268 |
| | Configuring NSAPI Redirector for Netscape Web servers | 270 |
| | Configuring ISAPI Redirector for Microsoft Web servers | 272 |
| | Configuring the servlet Redirector | 273 |

| | | |
|-----------|--|------------|
| 12 | Running MobiLink Outside the Current Session..... | 275 |
| | Running the UNIX MobiLink server as a daemon | 276 |
| | Running the Windows MobiLink server as a service..... | 277 |
| | Troubleshooting MobiLink server startup..... | 282 |

| | | |
|-----------|---|------------|
| 13 | Transport-Layer Security | 283 |
| | About transport-layer security | 284 |
| | Invoking transport-layer security | 293 |
| | Certificate authorities | 298 |
| | Certificate chains | 299 |
| | Enterprise root certificates | 300 |
| | Globally signed certificates | 305 |
| | Obtaining server-authentication certificates..... | 307 |
| | Verifying certificate fields | 310 |

PART TWO

| | | |
|--|--------------------------------|------------|
| | MobiLink Tutorials..... | 313 |
|--|--------------------------------|------------|

| | | |
|-----------|---|------------|
| 14 | Tutorial: Synchronizing Adaptive Server Anywhere Databases | 315 |
| | Introduction | 316 |
| | Lesson 1: Creating and populating your databases..... | 318 |
| | Lesson 2: Running the MobiLink synchronization server | 322 |
| | Lesson 3: Running the MobiLink synchronization client..... | 324 |
| | Tutorial cleanup | 326 |
| | Summary..... | 327 |
| | Further reading | 328 |

| | | |
|-----------|--|------------|
| 15 | Tutorial: Writing SQL Scripts Using Sybase Central .. | 329 |
| | Introduction | 330 |
| | Lesson 1: Creating your databases | 331 |
| | Lesson 2: Creating scripts for your synchronization | 335 |
| | Lesson 3: Running the MobiLink synchronization server | 338 |
| | Lesson 4: Running the MobiLink synchronization client..... | 340 |

| | | |
|-----------------------|--|------------|
| | Lesson 5: Monitoring your MobiLink synchronization using log files | 342 |
| | Tutorial cleanup..... | 344 |
| | Further reading..... | 345 |
| 16 | Tutorial: Using MobiLink with an Oracle 8i Consolidated Database..... | 347 |
| | Introduction | 348 |
| | Lesson 1: Create your databases | 349 |
| | Lesson 2: Starting the MobiLink synchronization server | 355 |
| | Lesson 3: Running the MobiLink synchronization client..... | 356 |
| | Summary..... | 357 |
| | Further reading..... | 358 |
| 17 | Using MobiLink Sample Applications | 359 |
| | Introduction | 360 |
| | The CustDB sample..... | 361 |
| | The Contact sample..... | 365 |
| PART THREE | | |
| | MobiLink Reference..... | 377 |
| 18 | MobiLink Synchronization Server Options | 379 |
| | MobiLink synchronization server..... | 380 |
| 19 | MobiLink Synchronization Client..... | 409 |
| | MobiLink synchronization client | 410 |
| | dbmlsync options | 413 |
| 20 | Synchronization Events | 433 |
| | Overview of MobiLink events | 436 |
| | authenticate_user connection event | 446 |
| | authenticate_user_hashed connection event | 450 |
| | begin_connection connection event..... | 452 |
| | begin_download connection event..... | 454 |
| | begin_download table event | 456 |
| | begin_download_deletes table event..... | 458 |
| | begin_download_rows table event..... | 460 |
| | begin_synchronization connection event..... | 462 |


| | |
|---|-----|
| begin_synchronization table event | 464 |
| begin_upload connection event | 466 |
| begin_upload table event | 468 |
| begin_upload_deletes table event | 470 |
| begin_upload_rows table event | 472 |
| download_cursor cursor event | 474 |
| download_delete_cursor cursor event | 477 |
| download_statistics connection event | 479 |
| download_statistics table event | 482 |
| end_connection connection event | 485 |
| end_download connection event | 487 |
| end_download table event | 489 |
| end_download_deletes table event | 491 |
| end_download_rows table event | 493 |
| end_synchronization connection event | 495 |
| end_synchronization table event | 497 |
| end_upload connection event | 499 |
| end_upload table event | 502 |
| end_upload_deletes table event | 504 |
| end_upload_rows table event | 506 |
| example_upload_cursor table event | 508 |
| example_upload_delete table event | 509 |
| example_upload_insert table event | 510 |
| example_upload_update table event | 511 |
| handle_error connection event | 512 |
| handle_odbc_error connection event | 515 |
| modify_last_download_timestamp connection event | 517 |
| modify_next_last_download_timestamp connection event | 519 |
| modify_user connection event | 521 |
| new_row_cursor cursor event | 523 |
| old_row_cursor cursor event | 525 |
| prepare_for_download connection event | 527 |
| report_error connection event | 529 |
| report_odbc_error connection event | 531 |
| resolve_conflict table event | 533 |
| synchronization_statistics connection event | 535 |
| synchronization_statistics table event | 537 |
| time_statistics connection event | 539 |
| time_statistics table event | 541 |
| upload_cursor cursor event | 543 |
| upload_delete table event | 545 |
| upload_fetch table event | 547 |
| upload_insert table event | 549 |
| upload_new_row_insert table event | 551 |

| | | |
|-----------|---|------------|
| | upload_old_row_insert table event | 553 |
| | upload_statistics connection event | 554 |
| | upload_statistics table event..... | 557 |
| | upload_update table event..... | 560 |
| 21 | MobiLink SQL Statements | 563 |
| | ALTER PUBLICATION statement..... | 565 |
| | ALTER SYNCHRONIZATION DEFINITION statement (deprecated) | 566 |
| | ALTER SYNCHRONIZATION SITE statement (deprecated) | 567 |
| | ALTER SYNCHRONIZATION SUBSCRIPTION statement | 568 |
| | ALTER SYNCHRONIZATION TEMPLATE statement (deprecated) | 569 |
| | ALTER SYNCHRONIZATION USER statement..... | 570 |
| | CREATE PUBLICATION statement..... | 571 |
| | CREATE SYNCHRONIZATION DEFINITION statement (deprecated) | 572 |
| | CREATE SYNCHRONIZATION SITE statement (deprecated) | 573 |
| | CREATE SYNCHRONIZATION SUBSCRIPTION statement | 574 |
| | CREATE SYNCHRONIZATION TEMPLATE statement (deprecated) | 575 |
| | CREATE SYNCHRONIZATION USER statement..... | 576 |
| | DROP PUBLICATION statement..... | 577 |
| | DROP SYNCHRONIZATION DEFINITION statement (deprecated) | 578 |
| | DROP SYNCHRONIZATION SITE statement (deprecated) | 579 |
| | DROP SYNCHRONIZATION SUBSCRIPTION statement | 580 |
| | DROP SYNCHRONIZATION TEMPLATE statement (deprecated) | 581 |
| | DROP SYNCHRONIZATION USER statement [MobiLink]..... | 582 |
| | START SYNCHRONIZATION DELETE statement | 583 |
| | STOP SYNCHRONIZATION DELETE statement | 584 |
| 22 | Stored Procedures..... | 585 |
| | Stored procedures to add or delete scripts | 586 |
| | Client event-hook procedures | 592 |

| | | |
|-----------|---|------------|
| 23 | Utilities | 609 |
| | ActiveSync provider installation utility | 610 |
| | MobiLink stop utility | 613 |
| | MobiLink client database extraction utility | 614 |
| | MobiLink user authentication utility | 618 |
| | Certificate reader utility | 620 |
| | Certificate generation utility | 621 |
| 24 | Data Type Conversions | 625 |
| | Sybase Adaptive Server Enterprise | 626 |
| | IBM DB2 | 627 |
| | Oracle | 629 |
| | Microsoft SQL Server | 630 |
| 25 | MobiLink Communication Error Messages | 631 |
| | Communication error messages sorted by code | 632 |
| | Communication error messages sorted by message | 636 |
| | Communication error messages sorted by constant | 640 |
| | Communication error descriptions | 644 |
| 26 | MobiLink synchronization server Warning Messages | 683 |
| | MobiLink synchronization server warning messages sorted by code | 684 |
| | MobiLink synchronization server warning messages sorted by message | 688 |
| | MobiLink synchronization server warning descriptions | 692 |
| A | ODBC Drivers | 707 |
| | ODBC drivers supported by MobiLink | 708 |
| 27 | Deploying MobiLink Applications | 711 |
| | Deployment overview | 712 |
| | Deploying the MobiLink server | 713 |
| | Deploying Adaptive Server Anywhere MobiLink clients | 714 |
| | Deploying UltraLite MobiLink clients | 715 |

| | |
|-------------------|------------|
| Index..... | 717 |
|-------------------|------------|

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" on page 19 of the book <i>Introducing SQL Anywhere Studio</i> . |

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 set

The SQL Anywhere Studio documentation set 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.
- ◆ **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.

-
- ◆ **Adaptive Server Anywhere Error Messages** This book provides a complete listing of Adaptive Server Anywhere error messages together with diagnostic information.
 - ◆ **Adaptive Server Anywhere C2 Security Supplement** 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. The book does *not* include the security features added to the product since certification.
 - ◆ **MobiLink Synchronization User's Guide** This book describes all aspects of 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.
 - ◆ **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.
 - ◆ **UltraLite User's Guide** This book describes how to build database applications for small devices such as handheld organizers using the UltraLite deployment technology for Adaptive Server Anywhere databases.
 - ◆ **UltraLite User's Guide for PenRight! MobileBuilder** This book is for users of the PenRight! MobileBuilder development tool. It describes how to use UltraLite technology in the MobileBuilder programming environment.
 - ◆ **SQL Anywhere Studio Help** This book is provided online only. It includes the context-sensitive help for Sybase Central, Interactive SQL, and other graphical tools.

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

Documentation formats

SQL Anywhere Studio provides documentation in the following formats:

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

To access the online books on Windows operating systems, choose Start►Programs►Sybase SQL Anywhere 8►Online Books. You can navigate the online books using the HTML Help table of contents, index, and search facility in the left pane, and using the links and menus in the right pane.

To access the online books on UNIX operating systems, run the following command at a command prompt:

```
dbbooks
```

- ◆ **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 following books are included in the SQL Anywhere Studio box:
 - ◆ *Introducing SQL Anywhere Studio.*
 - ◆ *Adaptive Server Anywhere Getting Started.*
 - ◆ *SQL Anywhere Studio Quick Reference.* This book is available only in printed form.

The complete set of books is available as the SQL Anywhere Documentation set from Sybase sales or from e-Shop, the Sybase online store, at <http://e-shop.sybase.com/cgi-bin/eshop.storefront/>.

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 are shown 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. 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.

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

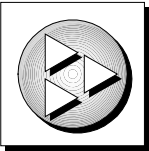
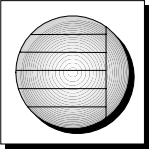
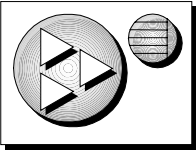
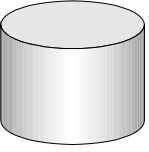
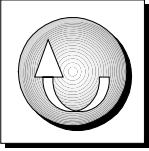
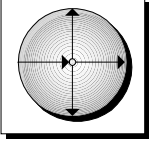

If the QUOTES option is chosen, one of ON or OFF must be provided.
The brackets and braces should not be typed.

- ◆ **One or more options** If you choose more than one, separate your choices with commas.

{ **CONNECT, DBA, RESOURCE** }

Graphic icons

The following icons are used in this documentation:

| Icon | Meaning |
|---|--|
|  | A client application. |
|  | A database server, such as Sybase Adaptive Server Anywhere or Adaptive Server Enterprise. |
|  | An UltraLite application and database server. In UltraLite, the database server and the application are part of the same process. |
|  | 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, SQL Remote Message Agent, and the Replication Agent (Log Transfer Manager) for use with Replication Server. |
|  | A Sybase Replication Server. |
|  | A programming interface. |

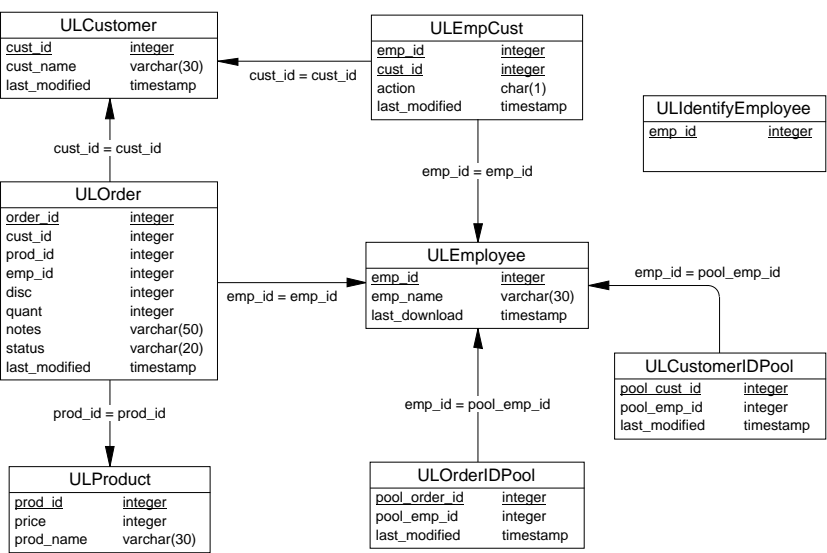
The sample database

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

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 ONE

Using MobiLink Technology


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


C H A P T E R 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 315
 - ◆ "Tutorial: Writing SQL Scripts Using Sybase Central" on page 329
 - ◆ "Tutorial: Using MobiLink with an Oracle 8i Consolidated Database" on page 347
 - ◆ "Using MobiLink Sample Applications" on page 359

 For a more detailed introduction to MobiLink technology, see "Synchronization Basics" on page 9.

Contents

| Topic | Page |
|--------------------------------------|------|
| The MobiLink synchronization process | 4 |
| MobiLink terminology | 7 |

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 any ODBC-compliant database, holds the master copy of all the data. Remote databases can be either Adaptive Server Anywhere or UltraLite databases.

Synchronization begins when a MobiLink remote site opens a connection to a MobiLink synchronization server. During synchronization a 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 sends back all relevant changes to the remote site.

MobiLink features

The MobiLink synchronization server is adaptable and flexible. MobiLink behavior can be adjusted using a comprehensive range of command line options for the MobiLink synchronization server, **dbmlsrv8**, and the Adaptive Server Anywhere synchronization client, **dbmlsync**. You can set a number of options on the typical MobiLink server or client command line to manage the following:

- ◆ **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**.
- ◆ **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.
- ◆ **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.

☞ For more information about monitoring your synchronizations, see "Monitoring MobiLink performance" on page 229.

- ◆ **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.

🔗 For more information about performance tuning, see "MobiLink Performance" on page 219.

MobiLink synchronization characteristics

Following are some of the features of MobiLink synchronization.

- ◆ **Two-way synchronization** Changes to a database can be made at any location.
- ◆ **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 is initiated at the remote database.
- ◆ **Session-based** All changes are 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 at 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, IBM DB2, or Microsoft SQL Server. Remote databases can be Adaptive Server Anywhere or UltraLite databases. 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, VxWorks, 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** The MobiLink synchronization server is multi-threaded, and multiple MobiLink servers can be run simultaneously using load balancing. 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.

MobiLink terminology

client In MobiLink contexts, client can refer to any application, database engine or executable that receives data resources from a server or requests a service.

client communication stream Clients can communicate with the synchronization server via a number of supported communications protocols.

consolidated database A database that contains all of the data, typically an enterprise level database. Supported products include Oracle, IBM's DB2, Microsoft SQL Server, Adaptive Server Anywhere, and Adaptive Server Enterprise.

download The stage in synchronization where data is transferred from a consolidated database to a remote database.

MobiLink synchronization server A Sybase session-based synchronization technology designed to synchronize UltraLite and Adaptive Server Anywhere databases with industry-standard SQL database-management systems.

MobiLink client There are two kinds of MobiLink clients. For Adaptive Server Anywhere remote databases the MobiLink client is the dbmlsync command line utility. For UltraLite remote databases, the MobiLink client is built in to the UltraLite runtime library.

publication A database object on the remote database that identifies data to be synchronized. A publication consists of articles that identify tables and columns to be synchronized.

Redirector A Web server plug-in that routes requests and responses between a client and the MobiLink synchronization server. This plug-in also implements load-balancing and fail-over mechanisms.

reference database An Adaptive Server Anywhere database used in the development of UltraLite clients, or as a convenience in the creation of remote Adaptive Server Anywhere clients. You can use a single Adaptive Server Anywhere database as both reference and consolidated database during development. Databases made with other products cannot be used as reference databases.

remote database An Adaptive Server Anywhere or UltraLite database that exchanges synchronization messages with a consolidated database. Remote databases may share all or some of the data in the consolidated database.

scripts Code written to handle MobiLink events. Scripts programmatically control data exchange to meet business needs.

session-based synchronization A type of synchronization where a synchronization results in consistent data representation across both the consolidated and remote databases.

subscription A database object that serves as a link in a remote database between a publication and a MobiLink user allowing the data described by the publication to be synchronized.

synchronization The coordination of data between multiple databases towards the end of consistent data representation.

transactional integrity The guaranteed maintenance of transactions across the synchronization system. Either a complete transaction is synchronized, or no part of the transaction is synchronized.

upload The stage in synchronization where data is transferred from a remote database to a consolidated database.

CHAPTER 2

Synchronization Basics

About this chapter

This chapter is an introduction to using MobiLink technology.

☞ For an introduction to MobiLink features, see

- ◆ "Introducing MobiLink Synchronization" on page 3

☞ For hands-on tutorials introducing MobiLink, see

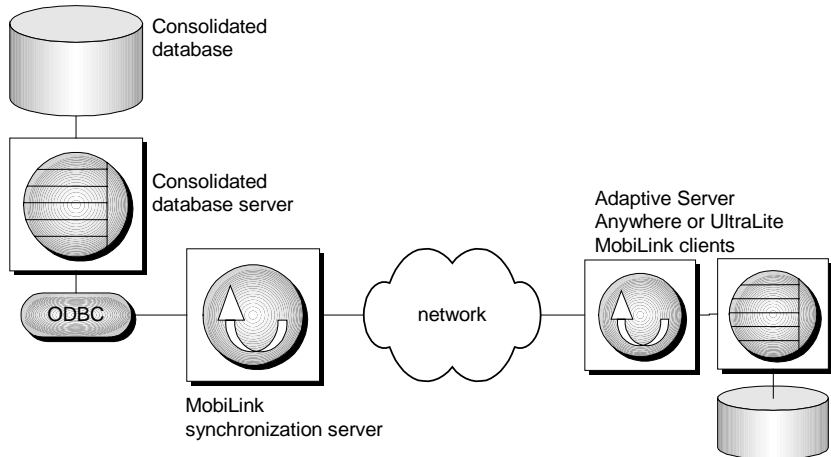
- ◆ "Tutorial: Synchronizing Adaptive Server Anywhere Databases" on page 315
- ◆ "Tutorial: Writing SQL Scripts Using Sybase Central" on page 329
- ◆ "Tutorial: Using MobiLink with an Oracle 8i Consolidated Database" on page 347
- ◆ "Using MobiLink Sample Applications" on page 359

Contents

| Topic | Page |
|---|------|
| Parts of the synchronization system | 10 |
| The consolidated database | 12 |
| The MobiLink synchronization server | 18 |
| MobiLink clients | 21 |
| The synchronization process | 24 |
| Options for writing synchronization logic | 38 |
| Character set considerations | 42 |
| Security | 45 |

Parts of the synchronization system

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



- ◆ **consolidated database** This database contains the master copy of all information in the synchronization system.
For more information, see "The consolidated database" on page 12.
- ◆ **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 "Consolidated database server requirements" on page 12.
- ◆ **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" on page 707.
- ◆ **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 18.

- ◆ **Network** The connection between the MobiLink synchronization server, dbmlsrv8, and the MobiLink client utility, dbmlsync, can use a number of protocols.

🔗 For more information, see "-x option" on page 396.

- ◆ **MobiLink client** 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 21.

The consolidated database

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

The consolidated database can be any supported ODBC-compliant product. You can use a Sybase product, such as Adaptive Server Anywhere or Adaptive Server Enterprise, or you can use a product sold by other companies, such as Oracle, IBM DB2, or Microsoft SQL Server.

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 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.

Consolidated database server requirements

MobiLink synchronization was designed to work with Sybase database products as well as other ODBC-supporting database-management systems, such as Oracle, IBM DB2, and Microsoft SQL Server. You can use synchronization scripts to exploit the features of your particular consolidated server.

☞ For a list of the supported platforms, see "MobiLink synchronization consolidated databases" on page 141 of the book *Introducing SQL Anywhere Studio*.

☞ For information about writing synchronization scripts for specific consolidated databases, see "DBMS-dependent scripts" on page 80.

How remote tables relate to consolidated tables

Synchronization designs can specify mappings between tables and rows in the remote database with tables and rows in the consolidated database. The only restriction is that columns match across both databases.

Arbitrary
relationships
permitted

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.

Note

Synchronization scripts are stored only in the consolidated database.

Direct relationships
are simple

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.

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.

Creating a consolidated database

To create a database that can be used as a MobiLink consolidated database, you must install the MobiLink system tables. Setup scripts are provided for Sybase Adaptive Server Anywhere, Sybase Adaptive Server Enterprise, Oracle 8, Microsoft SQL Server, and IBM DB2. A setup script is not required for Adaptive Server Anywhere databases.

Install the system tables

The way you install the MobiLink system tables depends on the DBMS you wish to use as a consolidated database.

- ◆ **Sybase Adaptive Server Anywhere** Adaptive Server Anywhere databases are automatically configured so that they can be used as a MobiLink consolidated database without running a setup script. However, there is a setup script provided for Adaptive Server Anywhere databases. It is called *syncasa.sql* and it is located in the scripts subdirectory of your SQL Anywhere installation. This file is provided so that you can examine its source code. For example, it includes source code for the *ml_add_connection_script* stored procedure that can be used if you want to directly insert scripts into system tables.
- ◆ **Sybase Adaptive Server Enterprise** For Adaptive Server Enterprise version 12.5 or later, run the *syncase125.SQL* 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.
- ◆ **Microsoft SQL Server** Run the *syncmss.SQL* script, located in the *MobiLink\setup* subdirectory of your SQL Anywhere installation.
- ◆ **Oracle** Run the *syncora.SQL* script, located in the *MobiLink\setup* subdirectory of your SQL Anywhere installation.
- ◆ **IBM DB2** For IBM DB2 version 6 or later, run the *syncdb2long.SQL* script, located in the *MobiLink\setup* subdirectory of your SQL Anywhere installation. For IBM DB2 prior to version 6, run the *syncdb2.SQL* 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.

Also, 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 "Creating a DB2 consolidated database for CustDB" on page 362.

☞ For instructions on running scripts, see the documentation for your DBMS. For tips on using specific consolidated databases with MobiLink, see "DBMS-dependent scripts" on page 80.

Create an ODBC data source

To carry out synchronization, the MobiLink synchronization server needs an ODBC connection to your consolidated database. You must have an ODBC driver for your server and create an ODBC data source for the database on the machine on which your MobiLink synchronization server is running.

☞ For more information on ODBC drivers, see "ODBC Drivers" on page 707.

MobiLink system tables

The MobiLink system tables store information for MobiLink users, tables, scripts, and script versions. The MobiLink system tables are stored in the consolidated database. You will probably not directly access these tables, but you will alter them when you perform actions such as adding synchronization scripts.

Notes

- ◆ The following section shows details of the MobiLink system tables. In some DBMSs, the data types are slightly different.
- ◆ IBM DB2 only supports column names and other identifiers of 18 characters or less. In a DB2 consolidated database, MobiLink system tables are truncated where necessary.

ml_connection_script

Stores the mapping for connection scripts.

| Row | Description |
|------------|----------------------------|
| version_id | INTEGER. Primary key. |
| event | VARCHAR(128). Primary key. |
| script_id | INTEGER. |

ml_script

Stores the text of all scripts, and an ID for mapping.

| Row | Description |
|-----------------|-----------------------|
| script_id | INTEGER. Primary key. |
| script | LONG VARCHAR. |
| script_language | VARCHAR(128). |

ml_script_version

Stores the name, ID and comment of script versions.

| Row | Description |
|-------------|-----------------------|
| version_id | INTEGER. Primary key. |
| name | VARCHAR(128). |
| description | LONG VARCHAR |

ml_scripts_modified

Keeps track of the last time script tables were changed.

| Row | Description |
|---------------|-------------------------|
| last_modified | TIMESTAMP. Primary key. |

ml_subscription

Keeps track of the log offsets per subscription for Adaptive Server Anywhere remote databases.

| Row | Description |
|------------------|----------------------------|
| user_id | INTEGER. Primary key. |
| publication_name | VARCHAR(128). Primary key. |
| progress | NUMERIC(20,0). |

ml_table

Stores the ID and name of tables that MobiLink can synchronize.

| Row | Description |
|----------|-----------------------|
| table_id | INTEGER. Primary key. |
| name | VARCHAR(128). |

ml_table_script

Stores the mapping for table scripts.

| Row | Description |
|------------|----------------------------|
| version_id | INTEGER. Primary key. |
| table_id | INTEGER. Primary key. |
| event | VARCHAR(128). Primary key. |
| script_id | INTEGER. |

ml_user

Stores all registered MobiLink users, including their password and their synchronization state. The state is used only for UltraLite remotes.

| Row | Description |
|-----------------|-----------------------|
| user_id | INTEGER. Primary key. |
| name | VARCHAR(128). |
| commit_state | INTEGER. |
| progress | NUMERIC(20,0) |
| hashed_password | BINARY(20) |

The MobiLink synchronization server

The MobiLink synchronization server manages (but does not initiate) the synchronization process. Thus, much of the built-in logic for the system resides in this application.

The MobiLink synchronization server also provides the interface between the consolidated database and the remote clients. Given sufficient resources, you can have as many remote applications synchronize simultaneously as you can specify on the command line using the `-w` option, making MobiLink synchronization well suited to large-scale deployment.

The MobiLink synchronization server controls the synchronization process. Each time a client synchronizes, a pre-defined sequence of events occurs.

You customize the synchronization process by writing synchronization scripts. You store these scripts in the consolidated database. Scripts are typically SQL statements or stored procedures and are stored in the consolidated database. You associate each script with a particular event in the pre-defined sequence. Whenever that event occurs, the MobiLink synchronization server automatically executes your script.

Running the MobiLink synchronization server

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

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

- ◆ Run `dbmlsrv8`. Use the `-c` option to specify the ODBC connection parameters for your consolidated database.

You must specify the database connection parameters. Other options are available, but are not required.

🔗 For information about connection parameters, see "`-c` option" on page 384.

🔗 For more information about `dbmlsrv8` options, see "MobiLink Synchronization Server Options" on page 379.

Example

The following command starts the MobiLink synchronization server, identifying the ODBC data source *UltraLite Sample 8.0* as the consolidated database. Enter the entire command on one line.

```
dbmlsrv8
-c "dsn=UltraLite Sample 8.0;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 275.

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 by:

- ◆ Clicking Shutdown on the MobiLink server window.
- ◆ Letting it shut down automatically by default when the application disconnects.
- ◆ Using the `dbmlstop` utility.

☞ For more information, see "MobiLink stop utility" on page 613.

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 recommended for normal operation of a production environment.

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*.

```
dbmlsrv8 -o mlsrv.log -c ...
```

☞ For more information, see "`-o` option" on page 387, "`-os` option" on page 388, and "`-ot` option" on page 389.

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" on page 393.

Controlling which
errors are reported

You can also control which error messages are reported.

☞ For more information, see "-zw option" on page 405, "-zwd option" on page 406, and "-zwe option" on page 407.


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 117.

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.

Initiating
synchronization
from an UltraLite
application

Synchronization is initiated from your application by a single call to a synchronization function when using TCP/IP, HTTP, or HTTPS. 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 on initiating synchronization, see "Adding synchronization to your application" on page 94 of the book *UltraLite User's Guide*.

Specifying the communications protocol for clients

The MobiLink synchronization server has command line options to specify the communications protocol or protocols for the synchronization client to connect to the MobiLink server.

The `-x` option allows you to specify a communications protocol through which MobiLink communicates with synchronization clients. The kind of communication protocol you choose must match the synchronization protocol used by the client. The syntax for this command line option is:

dbmlsrv8 -c "connection-string" -x protocol

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

```
dbmlsrv8 -c "dsn=ASA 8.0 Sample" -x tcpip
```

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

(keyword=value;...)

For example:

```
dbmlsrv8 -c "dsn=ASA 8.0 Sample" -x  
tcpip(host=localhost;port=2439)
```

☞ For more information about communication protocols, see "`-x` option" on page 396.

The MobiLink user

There is one MobiLink user name for each remote database in the MobiLink system. This name uniquely identifies each MobiLink client. 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, see

- ◆ "About MobiLink users" on page 252
- ◆ "Creating MobiLink users" on page 125

- ◆ "ml_user" on page 17

The synchronization process

A **synchronization session** is the process of bi-directional 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 always 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 applies all the changes in a single transaction. When it has finished, the MobiLink synchronization server commits the transaction.

Note

MobiLink operates using a default isolation level of 2 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. 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 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 "-e extended options" on page 414.

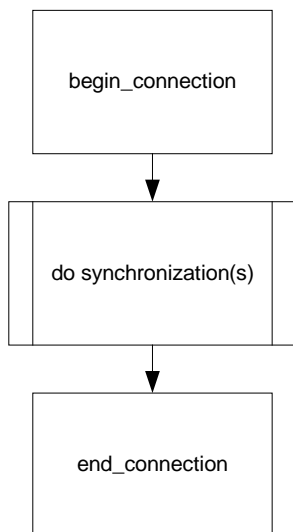
During MobiLink synchronization, there are few distinct exchanges of information. The client builds and uploads the entire upload stream. In response, the synchronization server builds and downloads the entire download stream. Limiting the chattiness of the protocol is especially important when communication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

MobiLink events

MobiLink synchronization is an event-driven process. When the MobiLink client initiates a synchronization, a number of synchronization events occur inside the MobiLink server. On 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:

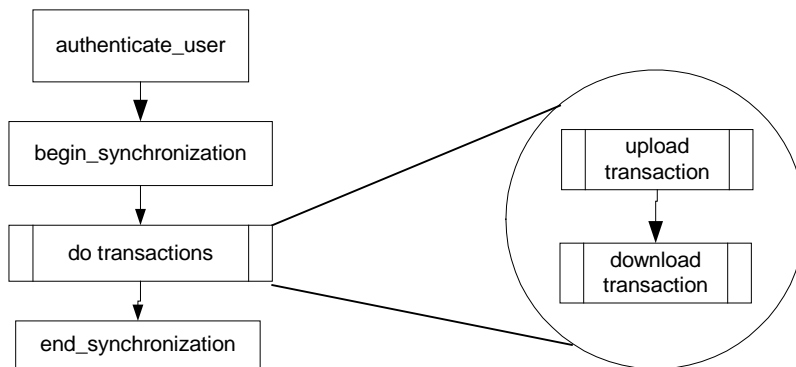
Event occurs ➤ *Script is invoked (if it exists)*

Prior to a begin_connection event, MobiLink waits for a synchronization request to occur. When the synchronization request happens, the begin_connection event is fired and synchronization starts.



Following the synchronization, MobiLink again waits for a synchronization request for the current script version. If no requests are initiated, the end_connection event is fired. But if another synchronization request for the same version is received, then MobiLink handles the next synchronization request on the same connection. Between the begin_connection and end_connection events are a number of other important events that all affect the current synchronization.

do synchronization(s)



For more information on these initial events, see "Scripts and the synchronization process" on page 53.

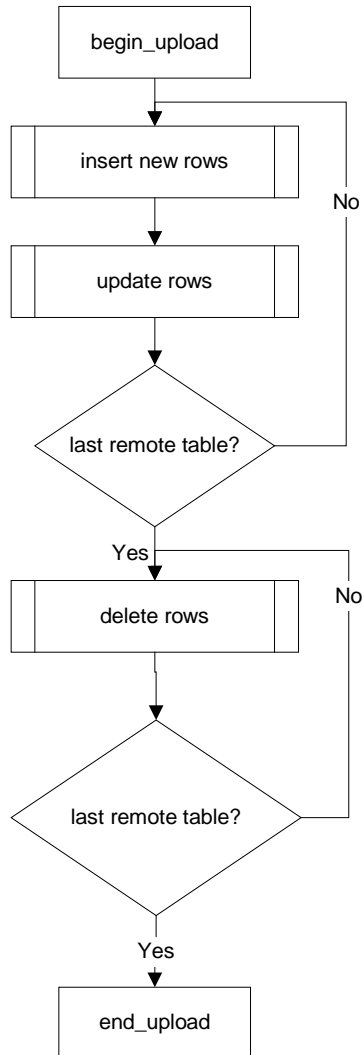
The events contained in the upload and download transactions are outlined below.

The upload transaction

The upload transaction precedes the download transaction.

The begin_upload event marks the beginning of the upload transaction. The upload transaction is a two-part process. First, inserts and updates are performed for all remote tables, and second, the rows are deleted for all remote tables.

upload transaction



The end_upload event marks the end of the upload transaction.

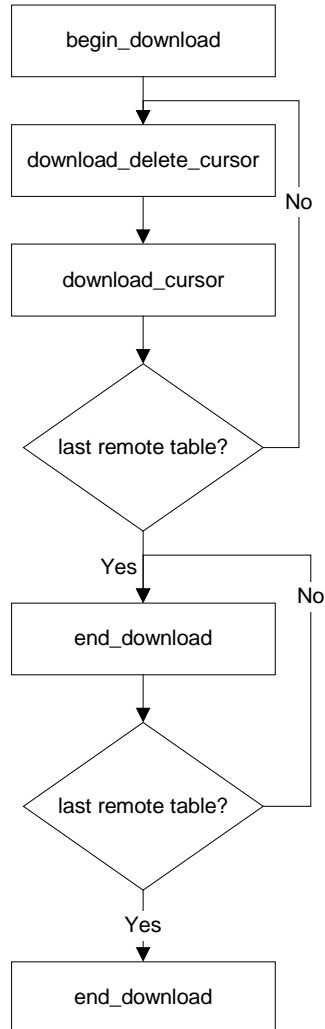
☞ For more information on the events that happen during upload, see "Writing scripts to upload rows" on page 66.

The download transaction

The download transaction consists of a `begin_download` event.

The download transaction is also a two-part process. First, deletes are performed for all remote tables, and then the rows are added for all remote tables in the `download_cursor`. The `end_download` event ends the download for all remote tables, and then the `end_download` event ends the download for the transaction.

download transaction



For more information on the events that happen during download see "Writing scripts to upload rows" on page 66.

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 | INSERT INTO emp (emp_id,emp_name) VALUES (?) |
| upload_delete | DELETE FROM emp WHERE emp_id=? |
| upload_update | UPDATE emp SET emp_name=? WHERE emp_id=? |
| upload_old_row_insert | INSERT INTO old_emp (emp_id,emp_name) VALUES (?,?) |
| upload_new_row_insert | INSERT INTO new_emp (emp_id,emp_name) VALUES (?,?) |

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.

Other synchronization scripts for tables can be generated using cursors. The following table shows examples of two cursor-based scripts.


| Event | Script |
|-----------------|---|
| upload_cursor | SELECT cust_id, emp_id, cust_name FROM cust WHERE cust_id = ? |
| download_cursor | SELECT emp_id, emp_name FROM emp WHERE emp_name = ? |


The first event, `upload_cursor`, triggers the running of the `upload_cursor` script. The `upload_cursor` script acquires data from `cust` for a specified `cust_id`. Similarly, the second event, `download_cursor`, acquires data from `emp` for a specified `emp_name`.

Statement-based events recommended


For most purposes, it is recommended that you use the statement-based events `upload_delete`, `upload_insert`, and `upload_update` instead of the `upload_cursor` event to process the upload stream.

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 38.

 For more information about your options for writing scripts, see


- ◆ "Introduction to synchronization scripts" on page 48
- ◆ "Writing Synchronization Scripts in Java" on page 165
- ◆ "Writing Synchronization Scripts in .NET" on page 187

 For information about DBMS-dependent scripting, such as scripting for Oracle, MS SQL Server, IBM's DB2 or Adaptive Server Enterprise databases, see "DBMS-dependent scripts" on page 80.

Storing scripts


Scripts are stored in system tables in the consolidated database. You can add SQL 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 63.

Procedural language

MobiLink 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" on page 585.

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 165 and "Writing Synchronization Scripts in .NET" on page 187.

Still more control over the synchronization process exists with stored procedures 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" on page 592.

Built-in automation and communications fault recovery

Built-in automation frees you from routine implementation tasks. For example, the MobiLink client automatically keeps track of the rows in each table that have been inserted, updated, or deleted since the last synchronization. When you synchronize, the MobiLink client automatically identifies the new, modified, or deleted rows and uploads them to the MobiLink synchronization server.

During the synchronization process, state information is maintained in both the application and the MobiLink synchronization server. Should the synchronization be interrupted, both the MobiLink synchronization server and application retain information regarding their own states. Upon the next attempt to synchronize, they automatically continue synchronization where they left off, without explicit direction.

Transactions in the synchronization process

A successful synchronization commits two transactions: an upload transaction and a download transaction.

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

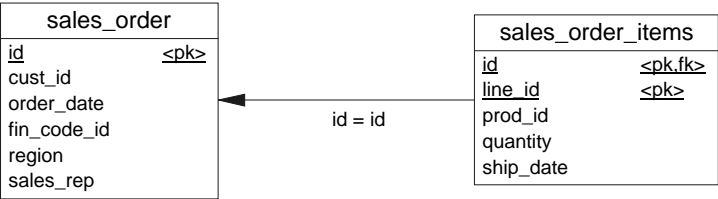
The MobiLink client resolves referential integrity violations automatically. This feature minimizes administration requirements. It also prevents an error in a synchronization script from disabling an MobiLink client.

An efficient means 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.

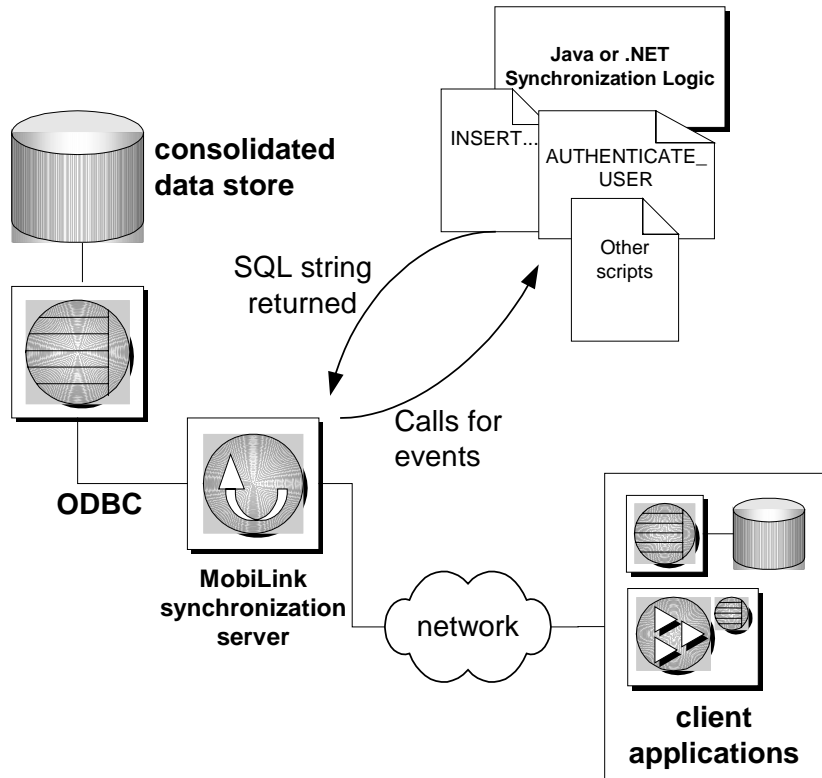
This arrangement has the following advantages.

- ◆ 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.

Options for writing synchronization logic

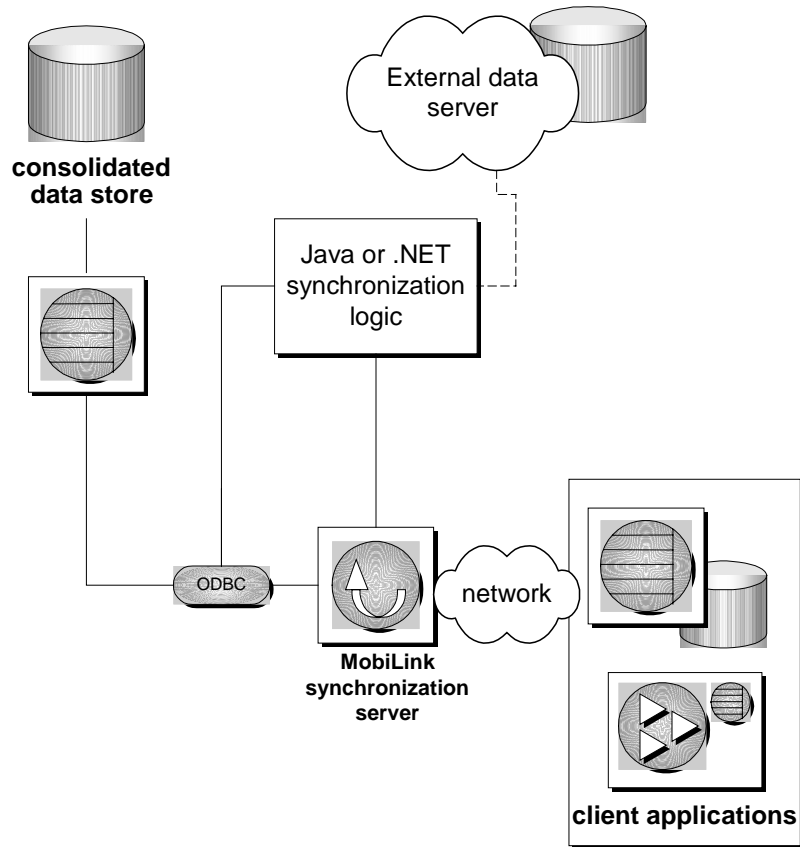
MobiLink synchronization scripts can be written in SQL, in Java, or in .NET programming languages. Java or .NET are a good choice whenever your design is restricted by the limitations of the SQL language or by the capabilities of your database-management system.

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. A SQL string may be returned to MobiLink.



SQL synchronization logic is restricted to the procedural language capabilities of your consolidated database. Few SQL languages 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

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 183.

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 203.

Further reading

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

- ◆ "Writing Synchronization Scripts" on page 47
- ◆ "Synchronization Techniques" on page 83
- ◆ "Writing Synchronization Scripts in Java" on page 165
- ◆ "Writing Synchronization Scripts in .NET" on page 187

Character set considerations

Each character of text is represented in one or more bytes. The mapping from characters to binary codes is called the **character set encoding**. Some character sets used for languages with small alphabets, such as European languages, use a single-byte representation. Others, such as Unicode, use a double-byte representation. Because they use twice the storage space for each character, double-byte character sets can represent a much larger number of characters.

When the character set of your MobiLink remote database is the same as your consolidated database, character translation issues are avoided. The characters you type can be transferred reliably.

Errors can occur or data can be lost when text using one character set must be translated to another character set. Not all characters can be represented in all character sets. In particular, single-byte character sets can represent a much smaller number of characters than multi-byte systems because of the limited number of codes available.

Most text needs to be sorted to build indexes and to prepare ordered result sets, such as directory listings. The **sort order** identifies the order of the characters. For example, a sort order typically states that the letter *a* comes before the letter *b*, which comes before the letter *c*.

Each database has a **collation sequence**. You set the collation sequence when you create the database, although how you do so can differ between database systems. The collation sequence defines both the character set and the sort order for that database.

Tip

Whenever possible, define the collation sequence of your reference database to be the same as that of your consolidated database. Doing so ensures that the collation sequence of your MobiLink remote database will match that of your consolidated database as closely as possible. This arrangement reduces the chance of erroneous translations.

🔗 For more information, see "Character sets in UltraLite" on page 64 of the book *UltraLite User's Guide*

Character-set translation during synchronization: Windows

During synchronization, characters may need to be translated from one character set to another. The following translations occur as characters are passed between the remote application and the consolidated database.

Character-set translation during upload

The MobiLink client sends data to the MobiLink synchronization server using the character set of the remote database.

- 1 The MobiLink synchronization server communicates with the consolidated database using the Unicode ODBC API. To do so, the MobiLink synchronization server translates all characters received from the remote database into Unicode.
- 2 If necessary, the ODBC driver for the consolidated database server translates the characters from Unicode into the character set of your consolidated database. This translation is controlled solely by the ODBC driver for your consolidated database system. Hence, behavior can differ between two different database systems, particularly systems made by different manufacturers. MobiLink synchronization works with a number of database systems. Check the documentation of your particular consolidated server for details.

Character-set translation during download

- 1 The ODBC driver for the consolidated database system receives characters in the coding of the consolidated database. It translates these characters into Unicode to pass them through the Unicode API to the MobiLink synchronization server. This translation is controlled solely by the ODBC driver for your consolidated database system. Check the documentation of your particular consolidated server for details.
- 2 The MobiLink synchronization server receives characters through the Unicode ODBC API. If the remote database uses a different character set, the MobiLink synchronization server translates the characters before downloading them.

Examples

- ◆ UltraLite applications on Windows CE devices use the Unicode character set.

When you synchronize a Windows CE application, no character translation occurs within the MobiLink synchronization server. The server finds that data arriving from the application is already in Unicode and passes it directly to the ODBC driver. Similarly, no character-set translation is necessary when downloading data.

- ◆ All Adaptive Server Anywhere databases and all UltraLite applications on platforms other than Windows CE use the character set determined by the collating sequence of the database.

When you synchronize a remote database, the MobiLink synchronization server performs character set translations between the character set of the remote database and Unicode.

Character set translation during synchronization: non-Windows


ODBC drivers on non-Windows platforms do not have Unicode entry points. The MobiLink synchronization server exchanges data with the ODBC driver using the character set determined by the collating sequence of the remote database.

When the remote database is an UltraLite application running under Windows CE, the MobiLink synchronization server performs character-set translation between Unicode and the character set being used with ODBC.

Controlling ODBC driver character-set translation

Because most consolidated databases are unlikely to use Unicode, it is important to understand how the ODBC driver for your consolidated database system converts data to and from Unicode. Some ODBC drivers use the language settings of the machine running MobiLink to determine what character set to use. In these cases, it is best if the language and code-page settings of the machine running the MobiLink synchronization server match those of the consolidated database.

Other ODBC drivers, such as the driver for Sybase Adaptive Server Enterprise, allow each connection to use a specific character set. To avoid translation errors, the character set used by MobiLink should be set to match that of the consolidated database.

 For a detailed description of how character-set translations take place in your database server's ODBC driver, consult that product's ODBC driver documentation.

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 is protected by 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" on page 387 of the book *ASA Database Administration Guide*.

- ◆ **Protecting data in the remote databases** If you are using Adaptive Server Anywhere MobiLink clients, the data is protected using the Adaptive Server Anywhere security features. 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 consolidated database, see "Keeping Your Data Secure" on page 387 of the book *ASA Database Administration Guide*.

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

🔗 For more information, see "Transport-Layer Security" on page 283.

- ◆ **Protecting the synchronization system from unauthorized users** MobiLink synchronization is secured by a password-based user authentication system. This mechanism prevents unauthorized users from downloading data or uploading changes.

🔗 For more information, see "Authenticating MobiLink Users" on page 251.

CHAPTER 3

Writing Synchronization Scripts

About this chapter

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

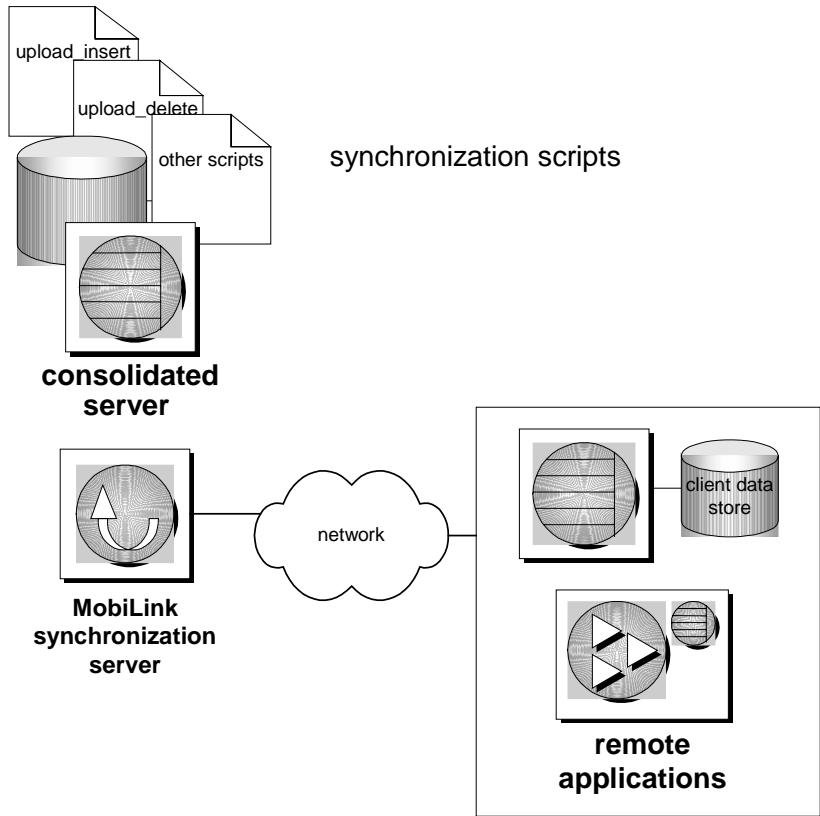
You can write scripts in SQL, Java, or .NET. This chapter applies to all kinds of scripts, but focuses on how to write synchronization scripts in SQL.

Contents

| Topic | Page |
|---|------|
| Introduction to synchronization scripts | 48 |
| Scripts and the synchronization process | 53 |
| Script types | 55 |
| Script parameters | 60 |
| Script versions | 61 |
| Adding and deleting scripts in your consolidated database | 63 |
| Writing scripts to upload rows | 66 |
| Writing scripts to download rows | 70 |
| Writing scripts to handle errors | 75 |
| Example scripts for UltraLite | 77 |
| Testing script syntax | 78 |
| DBMS-dependent scripts | 80 |

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 24.

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 38.

🔗 For information about writing scripts in .NET, see "Writing Synchronization Scripts in .NET" on page 187.

🔗 For information about writing scripts in Java, see "Writing Synchronization Scripts in Java" on page 165.

🔗 For information about how to implement synchronization scripts, see "Synchronization Techniques" on page 83.

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 `dbmlsrv8 -za` and `-zac` options 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" on page 402.

The `-zac` option creates the cursor-based scripts `upload_cursor` and `download_cursor`.

For more information, see "`-zac` option" on page 403.

Example

Start the MobiLink synchronization server using the `-za` switch. At a command prompt, type:

```
dbmlsrv8 -c "dsn=YourDBDSN" -za
```

Run `dbmlsync` and set the `SendColumnNames` extended option to ON. At a command prompt, 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 `dbmlrv8 -ze` and `-zec` options 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`, and `example_upload_delete`.

For more information, see "`-ze` option" on page 403.

The `-zec` option creates the cursor-based scripts `example_upload_cursor` and `example_download_cursor`.

For more information, see "`-zec` option" on page 404.

Example

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

At a command prompt, type:

```
dbmlsrv8 -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 |
|--------------------------------------|---|
| <code>example_upload_insert</code> | <code>INSERT INTO emp (emp_id,emp_name) VALUES (?,?)</code> |
| <code>example_upload_update</code> | <code>UPDATE emp SET emp_name=? WHERE emp_id=?</code> |
| <code>example_upload_delete</code> | <code>DELETE FROM emp WHERE emp_id=?</code> |
| <code>example_download_cursor</code> | <code>SELECT emp_id, emp_name FROM emp</code> |

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.

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" on page 436.

🔗 For the details of upload stream processing, see "Writing scripts to upload rows" on page 66.

🔗 For the details of download stream processing, see "Writing scripts to download rows" on page 70.

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" on page 433.

Script types

Synchronization scripts can be cursor-based or statement-based.

- ◆ **cursor scripts** These scripts perform the actions associated with the upload and download of data. Each script is associated with one particular remote table. It is recommended that you use statement-based scripts to handle the upload stream.
- ◆ **statement-based scripts** Statement-based scripts perform upload of data, including insert, update and delete functions.

Synchronization scripts can apply to connections or connections and 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.

Cursor scripts


Each cursor script must contain a `SELECT` statement. The MobiLink synchronization server defines a cursor based on this statement.

The following cursor scripts are used most frequently:

- ◆ **upload_cursor** This script defines a cursor through which the MobiLink synchronization server applies the upload stream to the consolidated database during the first step of synchronization.

Note

It is recommended that you use statement-based scripts to handle the upload stream.

 For more information, see "Writing scripts to upload rows" on page 66, and "upload_cursor cursor event" on page 543.

- ◆ **download_cursor** This script prepares inserts and updates for download to remote databases during the second step of synchronization. You must define a `download_cursor` cursor script for each remote table to which you want to download rows.

🔗 For more information, see "Writing scripts to download rows" on page 70, and "download_cursor cursor event" on page 474.

- ◆ **download_delete_cursor** This script prepares deletes for download to remote databases during the second step of synchronization. You use this script to select rows that are to be deleted from the remote database.

🔗 For more information, see "Writing download_delete_cursor scripts" on page 72, and "download_delete_cursor cursor event" on page 477.

The following cursor script events are also available. These scripts are used for conflict resolution.

- ◆ **old_row_cursor** Defines a cursor that the MobiLink synchronization server uses to insert the old values of uploaded rows.

🔗 For more information, see "old_row_cursor cursor event" on page 525.

- ◆ **new_row_cursor** Defines a cursor that the MobiLink synchronization server uses to insert the new values of uploaded rows.

🔗 For more information, see "new_row_cursor cursor event" on page 523.

🔗 For more information about conflict resolution, see "Handling conflicts" on page 104.

Statement-based scripts





Unlike cursor scripts, statement-based scripts do not require a `SELECT` statement. Following are some of the statement-based script events that are available.


- ◆ **upload_delete** MobiLink synchronization server uses this event during processing of the upload stream to handle rows deleted from the remote database.

🔗 For more information, see "Writing upload_delete scripts" on page 67, and "upload_delete table event" on page 545.

- ◆ **upload_insert** MobiLink synchronization server uses this event during processing of the upload stream to handle rows inserted into the remote database.

🔗 For more information, see "Writing upload_insert scripts" on page 67, and "upload_insert table event" on page 549.

- ◆ **upload_update** MobiLink synchronization server uses this event during processing of the upload stream to handle rows updated at the remote database.
 For more information, see "Writing upload_update scripts" on page 67, and "upload_update table event" on page 560.
- ◆ **upload_fetch** MobiLink synchronization server uses this event during processing of the upload stream to identify conflicts for rows updated at the remote database.
 For more information, see "Writing upload_fetch scripts" on page 67, and "upload_fetch table event" on page 547.
- ◆ **upload_old_row_insert** MobiLink synchronization server uses this event when conflicts are detected during processing of the upload stream to handle the old values of the updated row.
 For more information, see "upload_old_row_insert table event" on page 553.
- ◆ **upload_new_row_insert** MobiLink synchronization server uses this event when conflicts are detected during processing of the upload stream to handle the new values of the updated row.
 For more information, see "upload_new_row_insert table event" on page 551.

 For more information about uploading rows, see "Writing scripts to upload rows" on page 66.

The statement-based scripts can be written using the following syntax:

| Script name | Script syntax |
|------------------------------|---|
| upload_insert | INSERT INTO table_name (column_name1,column_name2, ...) VALUES (?, ?,...) |
| upload_delete | DELETE FROM table_name WHERE pk1=?, pk2=? |
| upload_update | UPDATE table_name SET column_name1=?,column_name2=?, ... WHERE pk1=?, pk2=? |
| upload_old_row_insert | INSERT INTO old_table_name (column_name1,column_name2, ...) VALUES (?, ?,...) |
| upload_new_row_insert | INSERT INTO new_table_name (column_name1,column_name2, ...) VALUES (?, ?,...) |

Following are examples of some of the scripts:

| Script name | Script example |
|-----------------------|--|
| upload_insert | INSERT INTO emp (emp_id,emp_name) VALUES (?) |
| upload_delete | DELETE FROM emp WHERE emp_id=? |
| upload_update | UPDATE emp SET emp_name=? WHERE emp_id=? |
| upload_old_row_insert | INSERT INTO old_emp (emp_id,emp_name) VALUES (?,?) |
| upload_new_row_insert | INSERT INTO new_emp (emp_id,emp_name) VALUES (?,?) |

Connection scripts

Connection scripts control actions centered on connecting and disconnecting. They also permit actions at synchronization-level events such as beginning and ending the upload or download process.

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.

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.

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.

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 parameters are typically available within 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.
- ◆ **MobiLink user name** 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 name** 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" on page 433.

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 62.

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 61.

❖ 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 OK.


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

- ◆ 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" on page 586.

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. There are stored procedures and Sybase Central wizards that make this easy.

 For information about the MobiLink system tables, see "MobiLink system tables" on page 15.

Adding or deleting scripts with Sybase Central

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 62.

❖ To add or delete a connection script:

- 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 OK.

❖ To add or delete a remote table in the list of synchronized tables:

- 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 OK.

❖ **To add or delete a table script in a database:**

- 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 OK.

Adding or deleting scripts with stored procedures

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" on page 586.

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" on page 15.

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 must define upload scripts.

There are two approaches to handling uploaded rows:

- ◆ **Statement-based scripts** In this approach, 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.

 For an overview, see "Statement-based scripts" on page 56.

- ◆ **Cursor-based scripts** In this approach, you write a single script that defines a cursor in the consolidated database. The script is commonly a SELECT statement or a stored procedure that returns a result set. The uploaded rows are inserted, deleted, or updated through this cursor.

Note: For most purposes, statement-based scripts are recommended. For more information about the performance benefits of statement-based scripts, see "Performance tips" on page 220.

The MobiLink synchronization server uploads data in a single transaction. For a description of the upload process, see "Events during upload" on page 438.

Notes

- ◆ The upload starts and ends with connection events. Other events are table-level events.
- ◆ 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 140.
- ◆ The upload_cursor script takes a different set of parameters from the other scripts. Its parameters are the primary key values identifying the row being uploaded.

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" on page 549.

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" on page 560.

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" on page 545.

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" on page 547.

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 upload_cursor scripts

The upload_cursor script is a SELECT statement that defines a cursor in the consolidated database table. This cursor is used to apply uploaded inserts, updates, and deletes. The upload_cursor script takes one parameter for each column in the primary key of the table. The script must be a SELECT statement. The WHERE clause of the SELECT statement must use the parameters to uniquely identify rows in the consolidated database.

Meaning of the
SELECT statement

Although both upload_cursor and download_cursor scripts contain SELECT statements, the meaning of the two statements is quite different. The download script selects all rows to be downloaded, but the MobiLink synchronization server requires the WHERE clause of the query in the upload_cursor script to identify exactly one row in the consolidated database to be updated or deleted.

Inserts are a
special case

If a table is to have only inserts at all remote databases, with no updates or deletes, no WHERE clause is required. No rows are being deleted or updated, and so the WHERE clause does not need to uniquely identify the row.

SELECT FOR
UPDATE

The old_row_cursor and new_row_cursor scripts are always used only for inserts. Therefore, they never require parameters or a WHERE clause.

SELECT statements in the upload_cursor script define cursors through which updates, deletes, and inserts are performed. SELECT statements in the new_row_cursor or old_row_cursor scripts are used in conflict resolution to define cursors through which inserts are performed.

Some database-management systems use an additional clause on the `SELECT` statement to indicate that the query defines a cursor that may be used for changes. If the DBMS you are using as a consolidated database uses such a clause, you may wish to include it in `SELECT` statements in your upload cursors.

Microsoft SQL Server, Sybase Adaptive Server Enterprise, IBM DB2, and Oracle use a `FOR UPDATE` clause to indicate queries that may be updated. No additional clause is required for Adaptive Server Anywhere.

Writing scripts to download rows

There are two cursor 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 113.

The MobiLink synchronization server downloads data in a single transaction. For a description of the download process, see "Events during download" on page 444.

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 `dbmlsync` option "SendDownloadACK" on page 420 or UltraLite "send_download_ack synchronization parameter" on page 389 of the book *UltraLite User's Guide*.
- ◆ 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 35.

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.

The MobiLink synchronization server deletes rows by selecting 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. If the primary key values match no remote row, the values are ignored.

- ◆ 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.

Each `download_delete_cursor` script must select all the column values present in the primary key of the corresponding remote table. However, it may, optionally, select all the other columns, too. 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 rows 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, you can use the following syntax:

```
SELECT NULL FROM SYSIBM.SYSTABLES
```

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 35.

☞ For more information about `download_delete_cursor`, see "download_delete_cursor cursor event" on page 477.

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" on page 512.

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" on page 529.

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.

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.

Testing script syntax

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

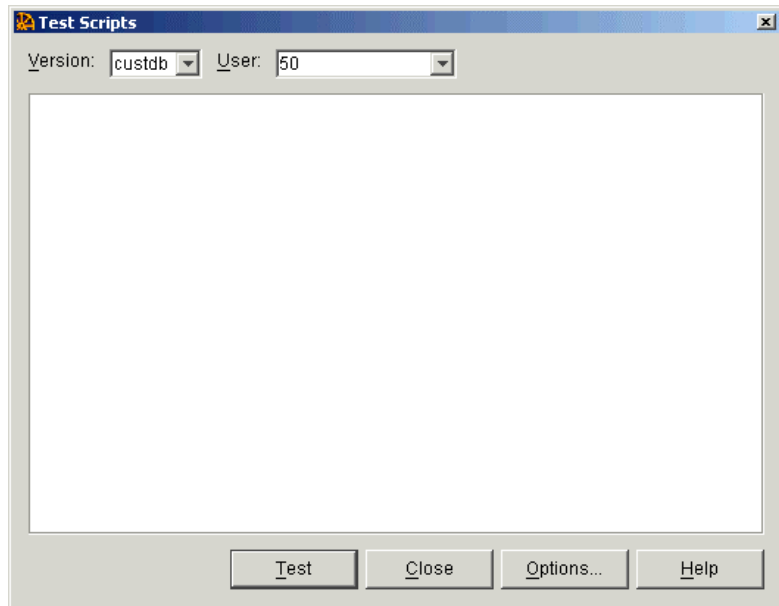
Testing the scripts is done without any remote site in place. No data is added to the database or downloaded from the database during testing. The validity of the synchronized data itself is not tested.

❖ To test your synchronization scripts

- 1 Start Sybase Central and connect to a database from MobiLink Synchronization. In this case, use the CustDB sample database.

🔗 For a description of how to carry out this step, see "Lesson 8: Browse the consolidated database" on page 35 of the book *UltraLite User's Guide*.

- 2 Under the UltraLite Sample database icon, right-click the Synchronized Tables folder or the Connection scripts folder, and select Test Scripts from the popup menu. The Test Script window appears.



- 3 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 170.

🔗 For information about .NET synchronization logic, see "Writing Synchronization Scripts in .NET" on page 187.

Supported DBMS scripting strategies

The MobiLink synchronization server can work with a variety of consolidated databases. The following considerations should help guide you while you use MobiLink for your database synchronization.

Oracle

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 Oracle sequence to maintain primary key uniqueness. The CustDB sample database provides coding examples, which can be found in *Samples\MobiLink\CustDB\custora.sql*.

☞ For an example of using an Oracle sequence, see "Tutorial: Using MobiLink with an Oracle 8i Consolidated Database" on page 347.

IBM DB2

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.

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.

☞ For an example of Java as a procedural language for DB2, see the CustDB scripts in the files *Samples\MobiLink\CustDB\custdbq.sql* and *Samples\MobiLink\CustDB\custdbq.java*.

☞ For more information about Java and .NET, see

- ◆ "Options for writing synchronization logic" on page 38
- ◆ "Writing Synchronization Scripts in Java" on page 165
- ◆ "Writing Synchronization Scripts in .NET" on page 187

Adaptive Server Enterprise

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 `StaticCursorLongColBuffLen`. For example,

```
dbmlsrv8 -c "...;StaticCursorLongColBuffLen=number"
```

where *number* is in bytes, and is larger than the largest expected BLOB.

Note that using this option consumes significantly more disk space on the computer that runs the MobiLink synchronization server.

Storing the MobiLink user name

Some database-management systems provide no convenient mechanism to store the identity of the current user.

☞ For more information, see "Storing the user name" on page 108.

Invoking procedures from scripts

Some databases, such as Microsoft SQL Server, require that procedure calls with parameters be written using the ODBC syntax.

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

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. You should take care when using CHAR.

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 "Data Type Conversions" on page 625.

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 "Using MobiLink Sample Applications" on page 359.

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 165
- ◆ "Writing Synchronization Scripts in .NET" on page 187

Contents

| Topic | Page |
|---|------|
| Introduction | 84 |
| Development tips | 85 |
| Timestamp-based synchronization | 86 |
| Snapshot synchronization | 88 |
| Partitioning rows among remote databases | 91 |
| Maintaining unique primary keys | 95 |
| Handling conflicts | 104 |
| Data entry | 110 |
| Handling deletes | 111 |
| Handling failed downloads | 112 |
| Downloading a result set from a stored procedure call | 113 |
| Schema changes in remote databases | 116 |

Introduction

The chapter "Writing Synchronization Scripts" on page 47 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_delete** script to handle modifications made at remote databases at the consolidated database.
- ◆ An **upload_insert** 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 deduce which functional 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.

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.

For compatibility with version 7 and earlier versions of the software, some users who wish to maintain existing scripts may wish to supply the dbmlsrv8 -zd option to alter the position of the last download timestamp parameter. The current section describes the default behavior.

❖ **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 "Synchronizing customers in the Contact sample" on page 370, and "Synchronizing contacts in the Contact sample" on page 372.

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 91.

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 72.

- 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 70.

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_cursor` 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 370.

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" on page 474, and "download_delete_cursor cursor event" on page 477.

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 95.

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 86.

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 370, and "Synchronizing contacts in the Contact sample" on page 372.

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 92) 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 372.

Maintaining unique primary keys

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" on page 73 of the book *ASA SQL User's Guide*
- ◆ "NEWID function " on page 159 of the book *ASA SQL Reference Manual*

- ◆ "UUIDTOSTR function " on page 193 of the book *ASA SQL Reference Manual*
- ◆ "STRTOUUID function " on page 185 of the book *ASA SQL Reference Manual*

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 97.


- 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 98.

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 "Global autoincrement default column values" on page 58 of the book *UltraLite User's Guide*.

 For information on how global autoincrement columns work in Adaptive Server Anywhere databases, see "How default values are chosen" on page 99. For information on how they work in UltraLite databases, see "Global autoincrement default column values" on page 58 of the book *UltraLite User's Guide*.

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" on page 350 of the book *ASA SQL Reference Manual*.

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:**
 - ◆ 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" on page 237 of the book *ASA Database Administration Guide*.

☞ For more information, see "Setting the `GLOBAL_DATABASE_ID` value" on page 98, and "`GLOBAL_DATABASE_ID` option" on page 569 of the book *ASA Database Administration Guide*.

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.

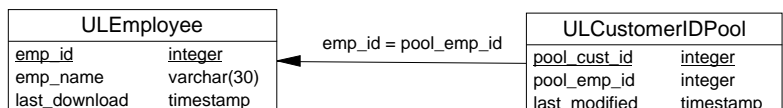
A primary key pool example

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 | |
|----------------|-------------|
| <u>cust_id</u> | 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 may 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

Never 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.

The process by which the MobiLink synchronization server detects conflicts depends on whether you are using statement-based uploads or cursor-based uploads. For most purposes, statement-based uploads are recommended.

Detecting conflicts with statement-based uploads

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" on page 547.

- ◆ 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" on page 560.

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" on page 553.
 - ◆ 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" on page 551.
 - ◆ 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" on page 533.

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 374.

Detecting conflicts with cursor-based uploads

When using `upload_cursor` scripts, the MobiLink synchronization server processes each uploaded update using the following procedure.

- 1 The MobiLink synchronization server compares the old uploaded values to the current values of the row with the same primary key values.
- 2 If the old uploaded values match the current contents in the consolidated database, the MobiLink synchronization server detects no conflict.
 - ◆ The MobiLink synchronization server updates the consolidated row using the new uploaded values. You define the cursor that the MobiLink synchronization server uses for this operation using the `upload_cursor` table script. The old uploaded values are discarded.
- 3 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 row using the cursor defined by the `old_row_cursor` script.
 - ◆ The MobiLink synchronization server inserts the new values row using the cursor defined by the `new_row_cursor` script.

- ◆ 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.

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.

☞ You can gain finer control over the conflict detection and resolution process. For details, see "Forced conflict resolution" on page 107.

Forced conflict resolution

Forced conflict resolution is a special technique that forces every uploaded row to be treated as if it were a conflict. Implementation of forced conflict resolution depends on whether you are using statement-based uploads or cursor-based uploads.

Forced conflict resolution using statement-based uploads

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.

Forced conflict resolution using cursor-based uploads

If no `upload_cursor` script is defined for a remote table, the MobiLink synchronization server attempts to insert all uploaded rows from that table using the cursors defined by the `old_row_cursor` and `new_row_cursor` 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 the `upload_cursor` script, the normal conflict-resolution procedure, described above, 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 `old_row_cursor` and `new_row_cursor` scripts, then analyze it as you see fit.

- ◆ **Performance** When the `upload_cursor` is 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 cursors defined by the `old_row_cursor` and `new_row_cursor` 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 `old_row_cursor` or `new_row_cursor` 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 `old_row_cursor` script is of the following form. The items in the select list correspond to the columns of the remote table.

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

However, the following syntax is also permitted.

```
SELECT 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.

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]" on page 563 of the book *ASA SQL Reference Manual*.


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" on page 474
- ◆ "Writing download_delete_cursor scripts" on page 72
- ◆ "download_delete_cursor cursor event" on page 477
- ◆ "Snapshot synchronization" on page 88
- ◆ "Temporarily stopping synchronization of deletes" on page 156
- ◆ "STOP SYNCHRONIZATION DELETE statement [MobiLink]" on page 563 of the book *ASA SQL Reference Manual*

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" on page 420 or "send_download_ack synchronization parameter" on page 389 of the book *UltraLite User's Guide*.

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` 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.

```
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

When the schema of a remote database changes, you need to re-generate the UltraLite database and build a new application. The application needs to be re-deployed and the new database populated by synchronizing with the MobiLink synchronization server. 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.

The most common schema changes are to add a new column to an existing table or to add a new table to the database.

Adding tables to remote databases

You can add tables to remote databases. The only change is that you will have a new row in *ml_table* and a new set of scripts that correspond to the new table. When a user synchronizes, the upload stream contains a list of the tables in the remote database. The MobiLink synchronization server only expects data for tables in this list. Only tables listed in the upload stream are synchronized.

Changing table definitions in remote databases

Changing the number or type of columns in an existing table must be done carefully. When a newer MobiLink client synchronizes, 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 accommodate the various versions of your application, you can create different versions of the synchronization scripts. Each time a client synchronizes, it specifies the correct script version name. Using this technique, an arbitrary number of versions of your application can co-exist while synchronizing with a single consolidated database.

☞ For more information, see "Script versions" on page 61.

CHAPTER 5

Adaptive Server Anywhere Clients

About this chapter

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

🔗 For a tutorial to walk you through some of the concepts in this chapter, see "Tutorial: Synchronizing Adaptive Server Anywhere Databases" on page 315.





Contents

| Topic | Page |
|---|------|
| Creating a remote database | 118 |
| Publishing data | 119 |
| Creating MobiLink users | 125 |
| Subscribing MobiLink synchronization users | 128 |
| Differences from version 7 | 132 |
| Initiating synchronization | 138 |
| Using ActiveSync synchronization | 143 |
| Deploying remote databases | 148 |
| Partitioning data between remote databases | 155 |
| Temporarily stopping synchronization of deletes | 156 |
| Customizing the client synchronization process | 157 |

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 119.
- 2 Create one or more publications in the new database.
 See "Creating MobiLink users" on page 125.
- 3 Create a MobiLink user.
 See "Subscribing MobiLink synchronization users" on page 128.
- 4 Subscribe a MobiLink user to one or more of the publications.
 See "Subscribing MobiLink synchronization users" on page 128.

Comparison to SQL Remote

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 data

A publication is a database object that identifies the data that is to be replicated. 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.

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.

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 MobiLink Synchronization Client folder.
- 3 Open the Publications folder and double-click Add Publication.
- 4 Type a name for the new publication. Click Next.
- 5 On the Tables tab, select a table from the list of Matching tables. Click Add. The table appears in the list of Selected tables on the right.
- 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" on page 314 of the book *ASA SQL Reference Manual*.

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.

❖ 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 and double-click Add Publication.
- 3 Type a name for the new publication. Click Next.
- 4 On the Tables tab, select a table from the list of Matching tables. Click Add. The table is added to the list of Selected tables on the right.
- 5 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.
- 6 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" on page 314 of the book *ASA SQL Reference Manual*.

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 and launch the Add Publication wizard.
- 3 Type a name for the new publication. Click Next.
- 4 On the Tables tab, select a table from the list of Matching tables. Click Add. The table is added to the list of Selected tables on the right.
- 5 On the Where 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.
- 6 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 rows 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 sending relevant order information to the sales rep Samuel Singer.

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

☞ For more information, see the "CREATE PUBLICATION statement" on page 314 of the book *ASA SQL Reference Manual*.

SUBSCRIBE BY

The CREATE PUBLICATION statement also allows a SUBSCRIBE BY clause. This clause can also 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 in Interactive SQL.

Notes

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

- ◆ Altering publications in a running MobiLink setup is likely to cause replication errors and can lead to loss of data unless carried out with care.

❖ **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 Right-click the publication or article and choose Properties from the popup menu.
- 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 (located within the MobiLink Synchronization Client folder).
- 3 Open the publication container.
- 4 Double-click Add Article.
- 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 OK 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 (located within the MobiLink Synchronization Client folder).
- 3 Open the publication container.
- 4 Right-click the article you want to delete and choose Delete from the popup menu.

❖ **To modify an existing publication (SQL):**

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

Example

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

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

☞ See also the "ALTER PUBLICATION statement" on page 216 of the book *ASA SQL Reference Manual*.

Dropping publications

You can drop a publication using either Sybase Central or the DROP PUBLICATION statement. If you drop a publication, all subscriptions to that publication are automatically deleted as well.

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" on page 402 of the book *ASA SQL Reference Manual*.

Creating MobiLink users

MobiLink users are not the same as database users. Each type resides in a different namespace. 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 252.

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 126.

❖ 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 Open the MobiLink Synchronization Client folder.
- 3 Open the MobiLink Users folder and double-click Add MobiLink User.
- 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]" on page 335 of the book *ASA SQL Reference Manual*.

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.
- ◆ **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 the performance and behavior.

You can override connection properties and extended options by setting properties for individual subscriptions, or setting extended options on the dbmlsync command line.

🔗 For more information on the meaning of the MobiLink properties and extended options, see "CREATE SYNCHRONIZATION USER statement [MobiLink]" on page 335 of the book *ASA SQL Reference Manual* and "MobiLink synchronization client" on page 410.

❖ 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 Locate the MobiLink user in the MobiLink Users folder, which is in the MobiLink Synchronization Client folder.
- 3 Right-click the MobiLink user and choose Properties from the popup 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 " on page 570.

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, which is in the MobiLink Synchronization Client 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 " on page 582.

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 119. For information about creating MobiLink users, see "Creating MobiLink users" on page 125.

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 subscriptions 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'
```

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.

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 -eu option.
- 2 options specified on the command line with the -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). You can specify options for a publication by creating a synchronization subscription without specifying a synchronization user.

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.

Altering MobiLink subscriptions

Synchronization subscriptions can be altered using 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 Right-click the desired user and choose Properties from the popup menu. The MobiLink user property sheet appears.
- 4 On the Subscriptions tab, select the subscription you wish to change and click Advanced.
- 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" on page 568.

Dropping MobiLink subscriptions

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

Note that if you drop all synchronization subscriptions for a particular MobiLink user, all records of that user are deleted from the database.

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]" on page 410 of the book *ASA SQL Reference Manual*.

Differences from version 7

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 still supported, and 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

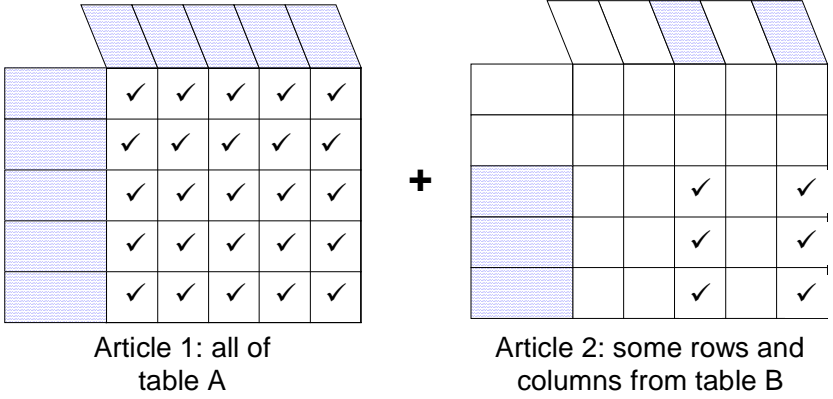
You can choose to synchronize all or any portion of the data in a client Adaptive Server Anywhere database. You can choose to synchronize entire tables, or you can choose to synchronize only particular columns and rows.

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 8.0 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 8.0 or later, publications and synchronization subscriptions should be used instead.

☞ For details, see "Creating a remote database" on page 118.

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 "MobiLink SQL Statements" on page 563.

Synchronizing with multiple servers

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

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

Rewriting synchronization definitions for version 8

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

Example

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

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

Instead, you should now do the following.

- 1 First, publish the *Customer* and *Sales_Order* tables.

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

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

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

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

The SITE clause in the synchronization definition specifies that this particular MobiLink client will synchronizing using the MobiLink user 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, then a subscription is created for the demo_sync_site MobiLink user. The TYPE, ADDRESS, and OPTION clauses have the same syntax.

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.

Tuning synchronization

MobiLink provides a number of extended options to tune 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 "CREATE SYNCHRONIZATION USER statement [MobiLink]" on page 335 of the book *ASA SQL Reference Manual* and "-e extended options" on page 414.

❖ To override an extended option on the `dbmlsync` command line:

- ◆ Supply the option values in the `-e` command line option for *dbmlsync*, in the form *option-name=value*. For example:

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

❖ To set an extended option for a subscription or user:

- ◆ Add the option to the CREATE SYNCHRONIZATION SUBSCRIPTION statement or CREATE SYNCHRONIZATION USER statement in the Adaptive Server Anywhere remote database. The values for each option cannot contain the characters "=", ",", or ";".

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 dbmsync 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 dbmsync 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 dbmsync from the directory that contains the renamed log files.

☞ For more information, see "Backup and Data Recovery" on page 299 of the book *ASA Database Administration Guide*.

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.

```
dbmsync -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

By default, the MobiLink client synchronization utility requires exclusive write access to all tables in the named publications. No other application can modify these tables at the same time. If synchronization is initiated when there are other connections to the remote database, write locks held by these connections can prevent synchronization from proceeding. In this case, synchronization is delayed until the other connections release their locks.

Sometimes, as is often the case for embedded applications, it may be possible to schedule synchronization only when no other operations are occurring in the remote database. Even when the remote database is used by a number of users or applications, it may be possible to find times of no activity during which synchronization can safely occur.

Once synchronization commences, other users are denied write access to the synchronized data. All processes requiring write access to the synchronized data must wait for the synchronization process to complete. The duration of this delay depends upon the amount of data to be exchanged, the speed of the connection to the MobiLink synchronization server, and the load on the server itself.

Forcing other connections to close

In some applications, it is essential that synchronization proceed as planned. In such situations, you can use the `dbmlsync -d` option to force the remote database to drop all other connections that are presently using resources required for synchronization.

When you do so, synchronization will proceed almost immediately, but other connected users or applications may be abruptly disconnected. Uncommitted changes are rolled back, so these users or applications must reconnect later to repeat any incomplete transactions.

Permitting concurrency during synchronization

You can permit other applications to obtain access to rows during synchronization by setting the **LockTables** (short form **lt**) extended option to OFF in the synchronization definition or on the `dbmlsync` command line. Even with this setting, rows modified by the synchronization process are still locked, as they would be by any other database connection.

The `dbmlsync` utility detects rows that are modified by other connections between the upload phase and the download phase.

If a conflict is detected, the download phase is cancelled and the download operations rolled back to avoid overwriting the new change. The `dbmlsync` utility then retries the synchronization, including the upload step. This time, because the row is present at the beginning of the synchronization process, it is included in the upload stream and therefore not lost.

By default, `dbmlsync` retries the synchronization after a conflict until no conflict occurs. You can customize this behavior by setting the extended option **ConflictRetries** (short form **cr**). The default value for **ConflictRetries** is `-1`, which indicates that `dbmlsync` should continue retrying until successful. A value of any non-negative integer *N* indicates that `dbmlsync` should retry up to *N* times if not successful.

Initiating synchronization from an application

You may wish to include the features of `dbmlsync` in your application, rather than provide a separate executable to your customers. If you are developing in C, you can do so.

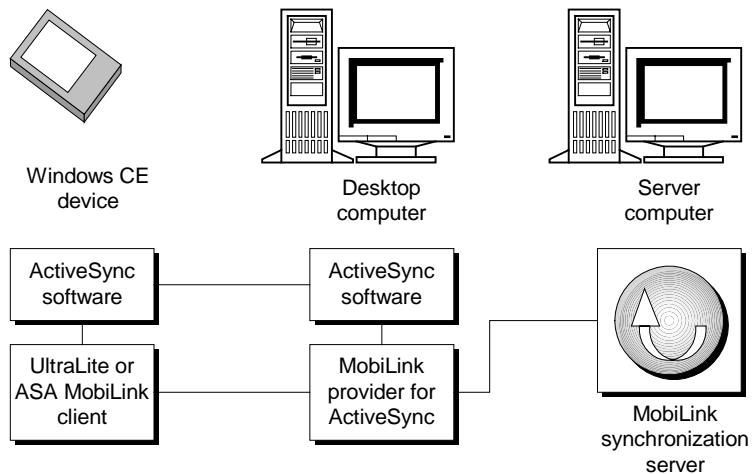
Include the *dbtools.h* header file located in the *h* subdirectory of your SQL Anywhere directory. This file contains a description of the `a_sync_db` structure and the `DBSynchronizeLog` function, which you use to add this functionality to your application.

☞ For more information, see "DBSynchronizeLog function" on page 299 of the book *ASA Programming Guide*, and "a_sync_db structure" on page 320 of the book *ASA Programming Guide*.

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 144.
- ◆ Install the MobiLink provider for ActiveSync.
 🔗 See "Installing the MobiLink provider for ActiveSync" on page 145.
- ◆ Register the Adaptive Server Anywhere client for use with ActiveSync.
 🔗 See "Registering Adaptive Server Anywhere clients for ActiveSync" on page 146.

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 162.

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]" on page 335 of the book *ASA SQL Reference Manual*.

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 146.

❖ 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 *-k* and *-v* command line options.


☞ For more information, see "ActiveSync provider installation utility" on page 610.

- 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 on registering applications, see "Registering Adaptive Server Anywhere clients for ActiveSync" on page 146.


Registering Adaptive Server Anywhere clients for ActiveSync


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

 For information on the alternative approach, see "ActiveSync provider installation utility" on page 610.

❖ 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 145.
- 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" on page 410.
 - ◆ **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.

Deploying remote databases

The basic method for creating a MobiLink remote database is to create the database and add the appropriate publications and subscriptions. However, alternative methods exist that can prove convenient when your design calls for a number of remote databases. These methods are as follows:

- ◆ Customize a prototype remote database.
- ◆ Extract remote databases from a reference database.

Customizing a prototype remote database

Customizing a prototype remote database is one method of deploying MobiLink remote databases. For an overview of deployment methods, see "Deploying remote databases" on page 148.

This method is illustrated in the Contact sample, which can be found in *Samples\MobiLink\Contact* in your SQL Anywhere directory.

❖ 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]" on page 493 of the book *ASA SQL Reference Manual*, and "Running command files" on page 99 of the book *ASA Getting Started*.

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]
```

Extracting remote databases

Extracting remote databases is one method of deploying MobiLink remote databases.

In this method, you first create a single Adaptive Server Anywhere database, called a **reference database**, which serves as a template for all the remote databases. To this database you add the publications and subscriptions needed by all the clients.

You are then ready to create the remote databases using information in the reference database as a template. This process is known as **extracting** the remote databases. The remote databases are extracted one at a time using the *mlxtract* command line utility.

The extracted remote databases need not all have the same structure. When extracting a remote database, you must name a particular MobiLink user. The remote database contains only the publications to which that user subscribed.

The reference database

A reference database is an Adaptive Server Anywhere database that serves as the template of the Adaptive Server Anywhere remote databases that you plan to create.

Each remote database must contain a subset of the tables, columns, and indexes found in the reference database.

The schema of the reference database may, but need not, be similar to the schema of the consolidated database.

In addition to the database schema, you must create two types of database objects within the reference database.

- ◆ **publications** A publication identifies data that is to be published in one more remote databases.
- ◆ **synchronization subscriptions** A synchronization subscription identifies which user or users will publish a particular set of data in their remote databases. A single user can subscribe to multiple publications.

A reference database should contain at least one synchronization subscription for each MobiLink remote database.

The reference versus the consolidated database

Do not confuse the reference database with the consolidated database. The consolidated database serves as the master repository of data in your replication system. The reference database is a template from which you can extract remote databases.

The following characteristics differentiate the reference database from the consolidated database.

- ◆ **database product** The reference database must be built with Adaptive Server Anywhere. In contrast, MobiLink supports a number of database products in the role of consolidated database, including Sybase Adaptive Server Anywhere, Sybase Adaptive Server Enterprise, Oracle, and IBM DB2.

- ◆ **presence of data** The consolidated database is the master repository of information in the synchronization system. In contrast, the reference database need not contain data. Optionally, you can populate the reference database with data that you want the remote databases to contain initially.
- ◆ **presence of publications and synchronization subscriptions** Publications and synchronization subscriptions are required in a reference database. They are permitted in a consolidated database, but have no effect.
- ◆ **schema** The schema of the reference data is a template for the remote databases. In contrast, MobiLink technology allows the consolidated database to have a different structure. For example, the data stored in a particular table and column at one client can be stored in a different table and column at another client, and in yet another location at the consolidated database.

Can a consolidated database function as a reference database?

When the following conditions are satisfied, a consolidated database can serve as a reference database. Using one database for both purposes is convenient, as you need maintain only one database instead of two.

- ◆ The consolidated database must be created with Adaptive Server Anywhere.
- ◆ The schema of the consolidated database must be a superset of the union of the schemas at the remote databases. In other words, the tables and columns in each remote databases must exist in the consolidated database.

Constructing a reference database

A reference database is a template for the remote databases. If a table or column must appear in one or more remote database, it must appear in the reference database.

A reference database can contain tables or columns not needed by any clients. Unnecessary tables and columns are ignored during the extraction process.

❖ To construct a new reference database

- 1 Create a new Adaptive Server Anywhere database.
- 2 Create all the tables that appear in any of the remote databases.
- 3 Create the publications needed for all the remote databases.
- 4 Subscribe all MobiLink synchronization users to the publications, as required.

Using an existing database

When you finish, you will have made a database that is the union of the tables, columns, publications, and subscriptions needed at all the client sites.

The extraction method is particularly convenient when you already have an Adaptive Server Anywhere database that contains the necessary tables and columns. In this case, you can use this database as the reference database, rather than creating a special-purpose reference database. In particular, the consolidated database may meet these requirements, depending on the design of your system.

To use this shortcut, the existing database must meet all of the following requirements.

- ◆ The database was made with Adaptive Server Anywhere.
- ◆ The database contains all the tables and columns required in the clients.
- ◆ The tables and columns have the same names as required at the clients.

If any of these conditions is not met, construct a new reference database.

❖ To use an existing database as a reference database

- 1 Create all the publications needed for the remote databases.
- 2 Subscribe all the MobiLink synchronization users to the publications, as required.

When you finish, you will have added the union of the publications and subscriptions needed at all the client sites.

Extracting remote databases

Once the reference database is complete, you can extract the remote databases. The extraction process is based on MobiLink synchronization users. It assumes that one remote database should be created for each MobiLink synchronization user.

Each time you extract a remote database you must specify a particular MobiLink synchronization user. By default, all publications to which the user has subscribed are extracted. Tables and columns are created in the remote database only if they appear in a publication to which the user has subscribed. The client does not inherit other tables or columns that exist in the reference database.

❖ To extract databases (one step)

- 1 Start the reference database.

**Multi-step
extraction**

- 2 Run the `mlxtract` command line utility. For example, to extract the remote database for a MobiLink synchronization called `mluser1` from a database named `refdb`, you can type the following statement at the system command prompt.

```
mlxtract -an -c "uid=DBA;pwd=SQL;dbn=refdb" mluser1
```

- 3 Repeat the previous step for each additional user's remote database.

The above method is convenient because you arrive at the finished remote database after issuing only one statement per remote database. However, it is sometimes more convenient to break the extraction process into multiple steps. This method is more flexible as it allows you to manipulate the files between steps, or take shortcuts such as making copies of the new database rather than re-creating one from scratch each time.

In other words, performing the extraction process is useful when you need more control over the process to efficiently create a custom solution.

❖ To extract a remote database (multi-step)

- 1 Run the `mlxtract` command line utility, specifying a MobiLink synchronization user ID on the command line. The utility writes a *reload.sql* command file for the named client.
- 2 Create a new Adaptive Server Anywhere database.
- 3 Start the new database.
- 4 Connect to the new database from Interactive SQL:
- 5 Run the reload command file by executing the following statement in Interactive SQL:

```
read path\reload.sql
```

Optionally, you can use the `-an` option for `mlxtract` to combine these steps into one operation.

Tip

The database extraction utility is intended to assist in preparing remote databases, but is not a black box solution in all circumstances. To provide custom solutions, you can edit the *reload.sql* command file. In particular, if you are extracting a very large number of clients, you may find it more efficient to run the `mlxtract` utility only once, then customize the *reload.sql* command file for each client.

The format of Adaptive Server Anywhere database files is independent of both operating system and file system. You can create the database under one operating system, then copy the database and log file to another machine. For example, you could create a database on a Windows machine, then copy it to a UNIX machine and use it with an Adaptive Server Anywhere server running there.

Initializing new remote databases with data

Newly extracted remote databases generally contain no data. It is customary to fill them with a starting set of data the first time synchronization occurs. During the first synchronization, the download scripts for each table select the appropriate rows from the consolidated database and send them to the client.

Alternatively, you can initialize an Adaptive Server Anywhere remote database with data during the extraction process. This method may be convenient if, for example, the client applications may be first synchronized over a slow network connection.

❖ To initialize a remote database with data during extraction:

- 1 Fill the tables in the reference database with the correct data.
- 2 Supply the `-id` option each time you run the remote database extraction utility, `mlxtract`.

🔗 For more information on available extraction options, see "MobiLink client database extraction utility" on page 614.

Troubleshooting deployment

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

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 template for the sales staff and another template for those employees who do technical support.


You can further fine-tune the data any given remote database will receive by using a conditional clause within the synchronization user definition. 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.

Example

Assume that you have three different categories of personnel at your company: manager, sales agent, and consultant. All employees in your company that you wish to deploy fall within one of the above categories. The personnel in each category require a different synchronization schema from the personnel in other categories. For example, the managers require a different set of tables from the sales agents and consultants. The sales agents will require a different set of tables from the managers and consultants. The consultants require a different set of tables from the other two groups.

Using publications, you can easily generate and re-generate databases for deployment. If you decide to modify the remote databases, for example add a new table, you only need to modify the reference database, then re-extract the remote databases.

Since each group requires a different schema, you should create three different synchronization templates in the reference database, one template for each group. Next, create a synchronization site in the reference database for each remote database. When creating each site, use the template that best suites the user's requirements. Finally, create the remote databases using the *mlxtract* command line utility.

 For more information on available extraction options, see "MobiLink client database extraction utility" on page 614.

Temporarily stopping synchronization of deletes

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

There can be circumstances, however, when you wish to delete rows from synchronized data and not have those changes uploaded. This feature can be used to make unusual corrections, for example, 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 or until a `COMMIT` or `ROLLBACK` statement terminates the transaction, whichever comes first. 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. Each event hook is identified by a stored procedure name. By writing a stored procedure with the supplied name, you can program a set of events that are executed as part of the synchronization process.

You can use the 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 the event hooks to synchronize subsets of data that cannot be included in a synchronization definition. 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.

The event-hook stored procedures are executed on the same connection as the synchronization itself.

✍ For more information about specific event-hook procedures, see "Client event-hook procedures" on page 592.

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 they each is reached 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 single 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.

The procedure `sp_hook_dbmlsync_upload_end` is called regardless of whether the upload succeeds or fails.

```
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
// no check for the success of the upload

// download events
sp_hook_dbmlsync_download_begin
for each table
    sp_hook_dbmlsync_table_begin
    sp_hook_dbmlsync_table_end
next table
sp_hook_dbmlsync_download_end

// end synchronization
sp_hook_dbmlsync_end
```

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
```


Once implemented, each procedure is automatically executed whenever an error of the named type occurs.

Example

The following code determines whether or not the upload succeeded:

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_end( )
BEGIN
    IF EXISTS (
        SELECT 1
        FROM #hook_dict
        WHERE name = 'upload_status'
            AND value = 'committed' )
    THEN
        UPDATE sync_params
        SET sync_required = 'N';
    END IF;
END;
```

The `sp_hook_dbmlsync_upload_end` event-hook provides two read-only parameter values available, namely synchronization definition name and upload status. The above code tests whether the latter parameter has the value completed, as it should following a successful upload.

 For a list of the parameter values supplied at each event, see "Client event-hook procedures" on page 592.

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 dbmlsync connection (typically a user with REMOTE DBA authority).
- ◆ A group ID of which the dbmlsync user is a member.

Connections for event-hook procedures

Each event-hook procedure is executed on the same connection as the synchronization itself. The following are exceptions:

- ◆ `sp_hook_dbmlsync_download_com_error`
- ◆ `sp_hook_dbmlsync_download_fatal_sql_error`

These procedures carry out operations 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 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 exchanged by reading and modifying rows in the *#hook_dict* table, which is defined as follows.

```
DECLARE GLOBAL TEMPORARY 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. For example, one row could contain the name *synchronization definition name* and the value of the synchronization definition name parameter.

Before calling any of the stored procedures, dbmlsync checks the *#hook_dict* table, and adds or updates the parameters for that event. Procedures can read the values by selecting from this table.

Some parameters are of type out. In these cases, the procedure can modify the values in the table. In these cases dbmlsync retrieves any new values from the *#hook_dict* table after the procedure has terminated.

☞ For a list of the parameter values supplied at each event, see "Client event-hook procedures" on page 592.

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.

```
CREATE PROCEDURE sp_hook_dbmlsync_delay()  
BEGIN  
    DECLARE down_time_start TIME;  
    DECLARE is_down_time varchar(128);  
  
    SET down_time_start = '19:00';
```

```

IF ABS( DATEDIFF(minute,
    down_time_start, CURRENT TIME) < 60 )
THEN
    set is_down_time='60';
ELSE
    SET is_down_time = '0';
END IF;

UPDATE #hook_dict
    SET value = is_down_time
    WHERE name = 'delay duration'
END

```

- ◆ The following procedure is executed in the remote database at the beginning of synchronization. It retrieves the current synchronization definition 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 = 'synchronization definition name';

    MESSAGE syncdef TYPE INFO TO CONSOLE;
END

```

- ◆ The following procedure not only reads parameter values, but also modifies them, returning the new values to the MobiLink client synchronization utility.

```

CREATE PROCEDURE sp_hook_dbmlsync_delay()
BEGIN
    DECLARE is_down_time VARCHAR(128);

    SELECT value INTO is_down_time
        FROM #hook_dict
        WHERE name = 'delay duration';

    MESSAGE 'is_down_time = ' || is_down_time
        TYPE INFO TO CONSOLE;

    IF ( is_down_time = '4' ) THEN
        SET is_down_time='0';
        MESSAGE 'setting to zero' TYPE INFO TO CONSOLE;
    ELSE
        SET is_down_time='4';
        MESSAGE 'setting to 4' TYPE INFO TO CONSOLE;
    END IF;

    UPDATE #hook_dict
        SET value = is_down_time
        WHERE name = 'delay duration';

```

```
UPDATE #hook_dict
SET value = '14'
WHERE name = 'maximum accumulated delay';
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.

❖ To add scheduling to the synchronization subscription:

- ◆ Set the **schedule** option in the synchronization subscription. For example,

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

🔗 For more information about scheduling syntax, see "Schedule option syntax" on page 347 of the book *ASA SQL Reference Manual*.

🔗 You can override scheduling instructions and synchronize immediately using the dbmlsync `-is` option. The `-is` option instructs dbmlsync to ignore scheduling information specified in synchronization subscriptions. For more information, see "`-is` option" on page 424.

❖ To add scheduling from the dbmlsync command line:

- ◆ Set the schedule extended option. Extended options are set with `-e` or `-eu`. For example,

```
dbmlsync -e sch=weekday@11:30am-12:30pm ...
```


If scheduled synchronization is specified in either place, dbmlsync does not shut down after synchronizing, but runs continuously. If scheduled synchronization is specified in both the synchronization definition and the command line, the command line receives priority.

☞ For information about extended options, see "-e extended options" on page 414. For more information about how to set scheduling, see "Schedule option syntax" on page 347 of the book *ASA SQL Reference Manual*.

C H A P T E R 6

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 38.

 For information about writing scripts, see "Writing Synchronization Scripts" on page 47.

 For information about writing scripts in .NET, see "Writing Synchronization Scripts in .NET" on page 187.

Contents

| Topic | Page |
|---------------------------------------|------|
| Introduction | 166 |
| Setting up Java synchronization logic | 167 |
| Running Java synchronization logic | 169 |
| Writing Java synchronization logic | 170 |
| Sample: Java synchronization logic | 177 |
| MobiLink Java API Reference | 183 |


Introduction

MobiLink synchronization scripts can be written in Java. This program 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 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. MobiLink uses the ASANYSH* environment variables to determine the shared component path.


 For more information, see "ASANYSH8 environment variable" on page 209 of the book *ASA Database Administration Guide*.

❖ 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.

 For more information about methods, see "Methods" on page 172.


Each class with non-static methods should have a 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 171.

- 2 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" on page 589, and "ml_add_java_table_script" on page 590.

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' )
```

- 3 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. To set the classpath for user-defined classes, use a statement such as the following:

```
SET  
classpath=%classpath%;c:\local\Java\myclasses.jar
```

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.

```
dbmlsrv8 -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 to your classpath.

```
dbmlsrv8 -c "dsn=MyDataSource" -sl java ( -cp c:\local\Java\myclass.jar )
```


 For more information about the available Java options, see "`-sl java option`" on page 391.

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 183.

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.

Constructors

The constructor of your class may have one of two possible signatures.

```
public MyScriptClass (
    ianywhere.ml.script.DBConnectionContext sc )
```

or

```
public MyScriptClass ( void )
```

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 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 8.
- ◆ 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.

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 `dbmlsrv8` command line.

☞ For more information, see "`-sl java` option" on page 391.

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.

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 `DMLStartClass` option of the `dbmlsrv8 -sl java` option. For example, the following is part of a `dbmlsrv8` command line. It causes `mycl1` and `mycl2` to be loaded as start classes.

```
-sl java (-DMLStartClass=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" on page 391.

✎ To see the start classes that are constructed at server start time, see "`getStartClassInstances`" on page 184.

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 );  
    }  
}  
}
```


Sample: Java synchronization logic

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 167

- ◆ 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" );  
}  
}
```

MobiLink Java API Reference

This section explains the MobiLink Java interfaces and classes, and their associated methods and constructors.

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.

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.

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 | <p>public void addShutdownListener (ShutdownListener <i>sl</i>)</p> <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>sl</i>, which specifies that the ShutdownListener is to be notified on shutdown.</p> |
| removeShutdownListener | <p>public void removeShutdownListener (ShutdownListener <i>sl</i>)</p> <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>sl</i>, which specifies the listener that will no longer be notified on shutdown.</p> |
| shutdown | <p>public void shutdown()</p> <p>Forces the server to shut down.</p> |
| getStartClassInstances | <p>public java.lang.Object[] getStartClassInstances()</p> <p>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.</p> <p> For more information about user-defined start classes, see "User-defined start classes" on page 174.</p> <p>Following is an example of getStartClassInstances():</p> <pre> Object objs[] = sc.getStartClassInstances(); int i; for(i=0; i<objs.length; i+=1) { if(objs[i] instanceof MyClass) { //use class } } </pre> |
| makeConnection method | <p>public java.sql.Connection makeConnection()</p> <p>throws java.sql.SQLException</p> <p>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.</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

public ServerException()

Constructs a ServerException with no detail message.

public ServerException(java.lang.String s)

Constructs a ServerException with a specified detail message. There is one parameter, *s*, which specifies the detailed message.

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

public void shutdownPerformed(ServerContext sc)

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.

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.

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

public **SynchronizationException()**

Constructs a `SynchronizationException` with no detail message.

public **SynchronizationException(java.lang.String s)**

Constructs a `SynchronizationException` with the specified detail message. There is one parameter, `s`, which specifies a detail message.

CHAPTER 7

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 47.

🔗 For a description and comparison of SQL, Java, and .NET, see "Options for writing synchronization logic" on page 38.

🔗 For information about writing scripts in Java, see "Writing Synchronization Scripts in Java" on page 165.

Contents

| Topic | Page |
|---------------------------------------|------|
| Introduction | 188 |
| Setting up .NET synchronization logic | 189 |
| Running .NET synchronization logic | 191 |
| Writing .NET synchronization logic | 194 |
| .NET synchronization example | 200 |
| MobiLink .NET API Reference | 203 |

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 just as it accesses Java scripts 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 196.

Each class with non-static methods should have a 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 195.

- 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*" on page 588 and "*ml_add_dnet_table_script*" on page 589.

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_dnet_connection_script( 'version1',  
    'authenticate_user', 'myPackage.myClass.myMethod' )
```

- 3 Create one or more assemblies. You tell MobiLink where to locate these assemblies using options on the *dbmlsrv8* command line. There are two options to choose from:

- ◆ **Use -sl dnet (-MLAutoLoadPath)** This sets the 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, or when for some other reason you need to use a configuration file.

 For more information about loading assemblies, see "Loading assemblies" on page 191. For more information about the dbmlsrv8 option -sl dnet, see "-sl dnet option" on page 390.

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</c
onfigFile>
    <assembly name="Assembly3" />
    <assembly name="Assembly4" />
  </domain>
</config>
```

```
</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:Package.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 `dbmlsrv8` option `-sl dnet`, see `"-sl dnet option"` on page 390.

Printing information from .NET

You may choose to add statements to your .NET methods that print information to a file of your choice. 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 .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.

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 203.

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.

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 JVM—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 `dbmlsrv8 -sl dnet` option. For example, the following is part of a `dbmlsrv8` 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" on page 390.

☞ To see the start classes that are constructed at server start time, see "GetStartClassInstances method" on page 203.

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 *asany8*.

```
csc /out:c:\mlexample\example.dll /target:library  
/reference:\asany8\win32\iAnywhere.MobiLink.Script.dll  
AuthUser.cs
```

Register .NET code for the *authenticate_user* event. The method we need to execute 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' )
```

Next, run the MobiLink synchronization server with the following option. This option causes MobiLink to load all assemblies in *c:\myexample*:

```
-sl 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. This section focuses on C#, but there are equivalents in Embedded Visual Basic and C++.

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 197.

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.....
        }
    }
}
```

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*
ShtutdownListener(*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 **ShtutdownCallback** : **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.

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 `ServerContext.makeConnection()`.

GetServerContext
 method

public **iAnywhere.MobiLink.Script.ServerContext** GetServerContext()
 Member of **iAnywhere.MobiLink.Script.DBConnectionContext**

Returns the `ServerContext` for this MobiLink server.

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.

SynchronizationEx
 ception
 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`.

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.

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_TIME STAMP 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_WLONGVAR CHAR field | public SQL_WLONGVARCHAR |
| | Null-terminated Unicode string of driver-dependent maximum length. Has .NET type String. |

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. |

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**

The value is the Input/Output direction of this parameter.

Default: ParameterDirection.Input.

IsNullable property

public bool **IsNullable**

The value Indicates whether this parameter can be NULL.

Default: false.

ParameterName
property

public string **ParameterName**

| | |
|--------------------|---|
| | <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 <code>SQLType.SQL_NUMERIC</code> and <code>SQLType.SQL_DECIMAL</code> 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 <code>SQLType.SQL_NUMERIC</code> and <code>SQLType.SQL_DECIMAL</code> 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 <code>DbType</code>.</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 <code>DBParameters</code>. When <code>DBCommand</code> creates a <code>DBParameterCollection</code> it is empty and must be filled with appropriate parameters before the <code>DBCommand</code> executes.</p> |
| DBParameterCollection() method | <p>public DBParameterCollection()</p> <p>Creates an empty list of <code>DBParameters</code>.</p> |
| Contains(string parameterName) method | <p>public bool Contains(string <i>parameterName</i>)</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(<i>string parameterName</i>)</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(<i>string parameterName</i>)</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(<i>object value</i>) 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(<i>object value</i>) 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(<i>object value</i>)</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(<i>int index</i>, <i>object value</i>)</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(<i>object value</i>)</p> <p>Removes the given iAnywhere.MobiLink.Script.DBParameter from the collection. Takes one parameter, <i>value</i>, which is the iAnywhere.MobiLink.Script.DBParameter.</p> |

RemoveAt(int
index) method

public int **RemoveAt(int index)**

Removes the iAnywhere.MobiLink.Script.DBParameter at the given index in the collection. Takes one parameter, *index*, which is the index of the iAnywhere.MobiLink.Script.DBParameter.

CopyTo(Array
array, int index)
method

public void **CopyTo(Array array, int index)**

Copies the contents of the collection into the given array starting at the specified index. Takes two parameters: *array*, which is the array to copy the contents of the collection into; and *index*, which is the index in the array to begin copying the contents of the collection into.

GetEnumerator()
method

public IEnumerator **GetEnumerator()**

Returns an enumerator for the collection.

IsFixedSize
property

public bool **IsFixedSize**

Returns false.

IsReadOnly
property

public bool **IsReadOnly**

Returns false.

Count property

public int **Count**

The number of parameters in the collection.

IsSynchronized
property

public bool **IsSynchronized**

Returns false.

SyncRoot property

public object **SyncRoot**

Object that can be used to synchronize the collection.

this[string
parameterName]
property

public object **this[string parameterName]**

Gets or sets the iAnywhere.MobiLink.Script.DBParameter with the given name in the collection. Takes one parameter, *parameterName*, which is the name of the iAnywhere.MobiLink.Script.DBParameter to get or set.

this[int index]
property

public object **this[int index]**

Gets or sets the `iAnywhere.MobiLink.Script.DBParameter` at the given index in the collection. Takes one parameter, *index*, which is the index of the `iAnywhere.MobiLink.Script.DBParameter` to get or set.

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 "SQL-.NET data types" on page 195.

`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.

CHAPTER 8

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 | 220 |
| Key factors influencing MobiLink performance | 224 |
| Monitoring MobiLink performance | 229 |


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 may 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 they are called just before a commit.

 For more information about contention, see "Contention" on page 225.

 For information on the transaction structure of synchronization, see "Transactions in the synchronization process" on page 32.

- ◆ **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 of fast clients it was discovered that approximately five worker threads gave optimum throughput. For slower clients, more worker threads are needed to maximize download throughput, and the best upload throughput is 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 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 226.

🔗 For more information, see "-w option" on page 394 and "-wu option" on page 395.

- ◆ **Use statement-based uploads** Use statement-based uploads instead of cursor-based uploads. The statement-based upload scripts are `upload_update`, `upload_insert`, `upload_delete`, and `upload_fetch`. Performance testing shows that statement-based uploads can be significantly faster than cursor-based uploads, and no cases have been found where they are slower.

Using statement-based upload scripts allows MobiLink to apply inserts, updates and deletes in batch mode. The `-s` option controls how many rows are batched at a time. The default is ten. You may want to experiment with different values to improve performance.

🔗 For more information on statement-based uploads, see "Writing scripts to upload rows" on page 66.

- ◆ **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 `dbmlsync` extended option `DownloadBufferSize` in "-e extended options" on page 414.

- ◆ **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.

🔗 For more information about download acknowledgements, see the extended option `SendDownloadACK` in "-e extended options" on page 414 and "`send_download_ack` synchronization parameter" on page 389 of the book *UltraLite User's Guide*.

- ◆ **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 `dbmlsrv8 -u` option.

🔗 For more information, see "-u option" on page 393.

- ◆ **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 dbmlsrv8 -d option.

🔗 For more information about setting the memory allocated to the download buffer, see "-d option" on page 386.

- ◆ **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 dbmlsrv8 -bc option.

🔗 For more information, see "-bc option" on page 384.

- ◆ **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. This reduces the need for MobiLink to close and create database connections. You set the maximum number of connections with the dbmlsrv8 -cn option.

🔗 For more information, see "MobiLink database connections" on page 227 and "-cn option" on page 385.

- ◆ **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.
- ◆ **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 even need to be on the same computer as the MobiLink synchronization server. For more information, see "MobiLink Monitor" on page 231.

- ◆ **Java vs. SQL synchronization logic** No significant throughput difference has been found between using Java or SQL synchronization logic, although Java synchronization logic has some extra overhead per synchronization and requires more memory. Note that using Java synchronization logic moves execution of your synchronization logic from the computer running the consolidated database to the computer running the MobiLink server. This 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 select the required rows. However, too many indexes may slow uploads.
- ◆ **For large uploads, 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" on page 430.

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

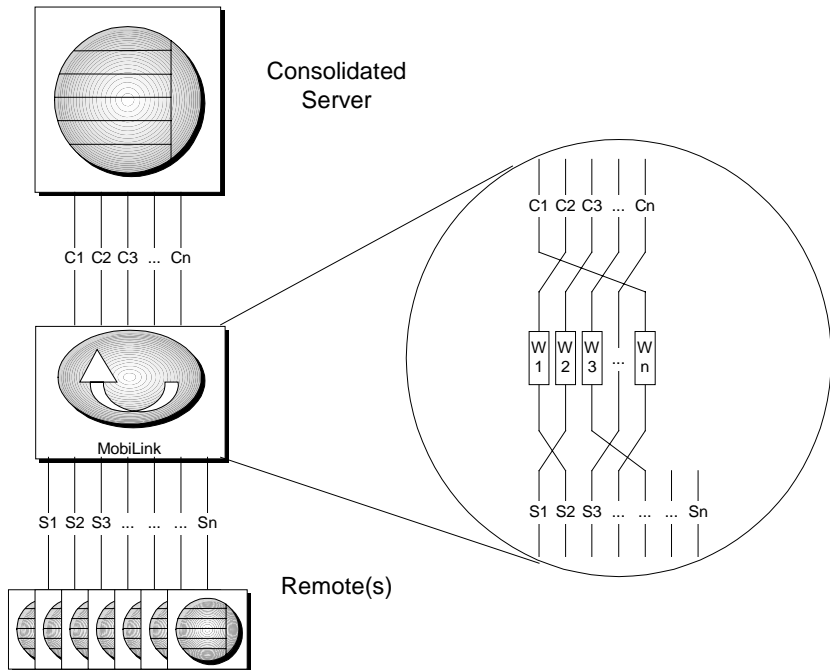
The key to achieving optimal MobiLink synchronization throughput is to have multiple synchronizations occurring simultaneously and executing efficiently. To enable multiple simultaneous synchronizations, MobiLink assigns a worker thread to each synchronization. A worker thread receives the changes uploaded 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 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.

Number of worker threads

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 dbmlsync extended option `SendDownloadAck` in the "-e extended options" on page 414.

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" on page 394 and "-wu option" on page 395.

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:

```
dbmlsrv8 -c "dsn=ASA 8.0 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, 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:


```
dbmlsrv8 -c "dsn=ASA 8.0 Sample" -w 7 -cn 21
```

 For more information on setting the number of connections for any number of script versions, see "-cn option" on page 385.

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 231.

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" on page 479
- ◆ "download_statistics table event" on page 482
- ◆ "synchronization_statistics connection event" on page 535
- ◆ "synchronization_statistics table event" on page 537
- ◆ "time_statistics connection event" on page 539
- ◆ "time_statistics table event" on page 541
- ◆ "upload_statistics connection event" on page 554
- ◆ "upload_statistics table event" on page 557

C H A P T E R 9

MobiLink Monitor

About this chapter The MobiLink Monitor is a tool for monitoring the performance of
MobiLink synchronizations.

| Contents | <table><tr><th>Topic</th><th>Page</th></tr><tr><td>Introduction</td><td>232</td></tr><tr><td>Starting the MobiLink Monitor</td><td>233</td></tr><tr><td>Using the MobiLink Monitor</td><td>235</td></tr><tr><td>Saving Monitor data</td><td>243</td></tr><tr><td>Customizing your statistics</td><td>244</td></tr><tr><td>Statistical Properties</td><td>247</td></tr></table> | Topic | Page | Introduction | 232 | Starting the MobiLink Monitor | 233 | Using the MobiLink Monitor | 235 | Saving Monitor data | 243 | Customizing your statistics | 244 | Statistical Properties | 247 |
|-------------------------------|--|-------|------|--------------|-----|-------------------------------|-----|----------------------------|-----|---------------------|-----|-----------------------------|-----|------------------------|-----|
| Topic | Page | | | | | | | | | | | | | | |
| Introduction | 232 | | | | | | | | | | | | | | |
| Starting the MobiLink Monitor | 233 | | | | | | | | | | | | | | |
| Using the MobiLink Monitor | 235 | | | | | | | | | | | | | | |
| Saving Monitor data | 243 | | | | | | | | | | | | | | |
| Customizing your statistics | 244 | | | | | | | | | | | | | | |
| Statistical Properties | 247 | | | | | | | | | | | | | | |

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 server 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.

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. Therefore, you may want to start the Monitor before starting synchronization. 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►Sybase SQL Anywhere 8►MobiLink►MobiLink Monitor.
- 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 is 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. 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. The allowed values are the same as when you connect using `dbmlsync`, except that you do not need to supply a script version. For all MobiLink Monitor sessions, the script version is set to **for_ML_Monitor_only**.

 For more information about connecting to the MobiLink synchronization server, see "`-c` option" on page 384.

- 4 Start synchronizing.

The data appears in the Monitor as it is collected.

❖ **To stop the MobiLink Monitor:**

- 1 In the Monitor, choose Monitor ► Disconnect from MobiLink Server. This stops the collection of data.

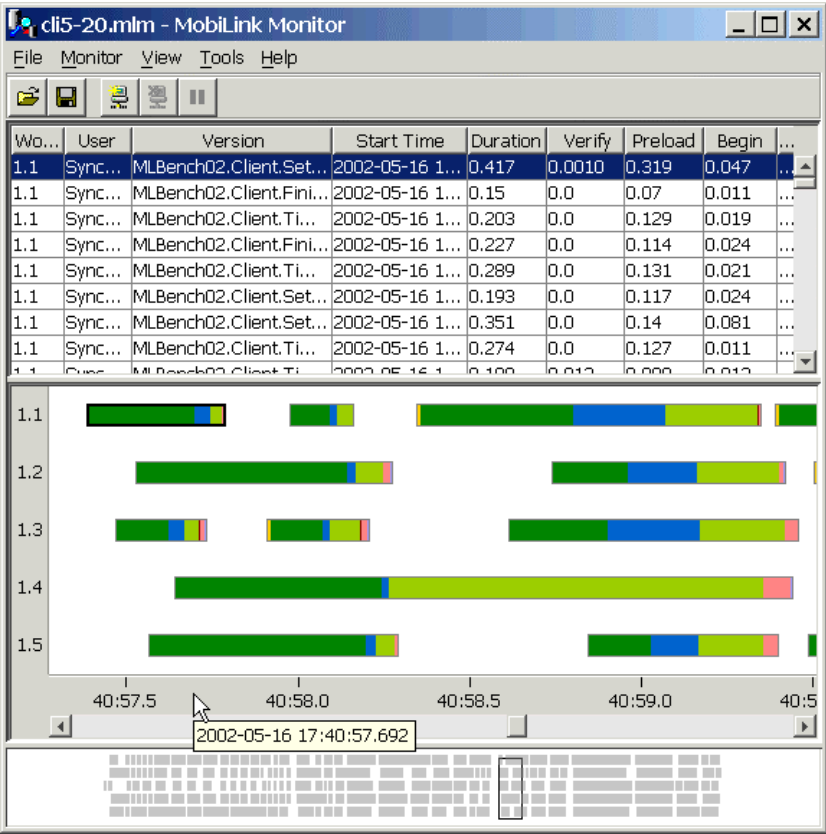
You can also stop collecting data by shutting down the MobiLink synchronization server.

- 2 When you are ready to close the Monitor, just click the X in the upper right corner of the screen, or choose File ► Close.

Before closing the Monitor, you can save the data for this session. For more information, see "Saving Monitor data" on page 243.

Using the MobiLink Monitor

Following is an example of the MobiLink Monitor when synchronization data has been logged:



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 scrollbar at the bottom of this pane represents time. You can zoom in on the data in the Chart by drawing a box around data in the Overview pane; or by choosing View►Go To.

In the screenshot above, the cursor is hovering over the time axis, and so a box is apparent that shows the complete date-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 and properties dialogs that you can use to customize the data. 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 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 61.
- ◆ **Start Time** The date and time when the MobiLink synchronization server started the synchronization. (This is not the same as 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 242.

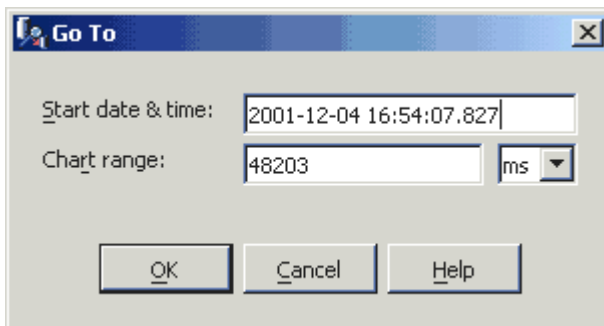
Grouping data by thread or user

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.

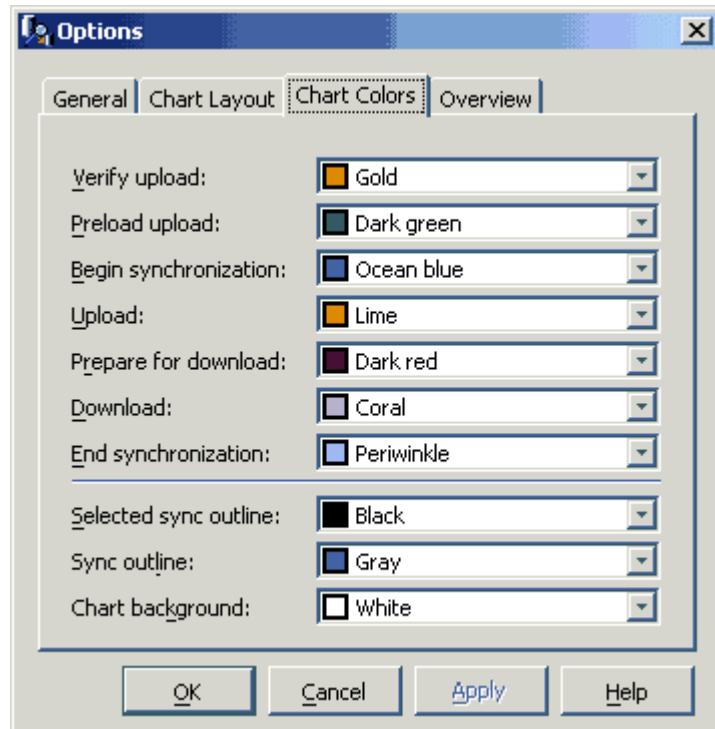
- ◆ **Overview Pane** The box in the Overview pane allows you to quickly select a portion of data to view. You can easily resize or move the box to see different data, or see data at different granularity. If you make the box smaller you shorten the interval of the visible data, which makes more detail visible. Click to move the current box without changing the zoom. Drag the box to resize it, changing the zoom.

Time axis

At the bottom of the Chart pane there is an axis 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 axis.

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 is as follows:



The default scheme 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 240.

Overview pane

The Overview pane shows you an overview of the entire synchronization 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 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.

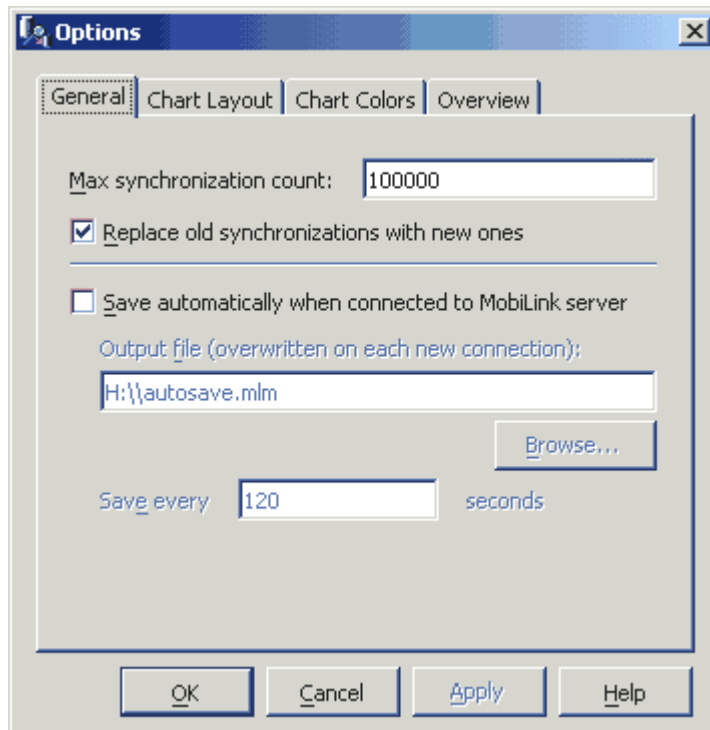
For more information, see "Options" on page 240.

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:



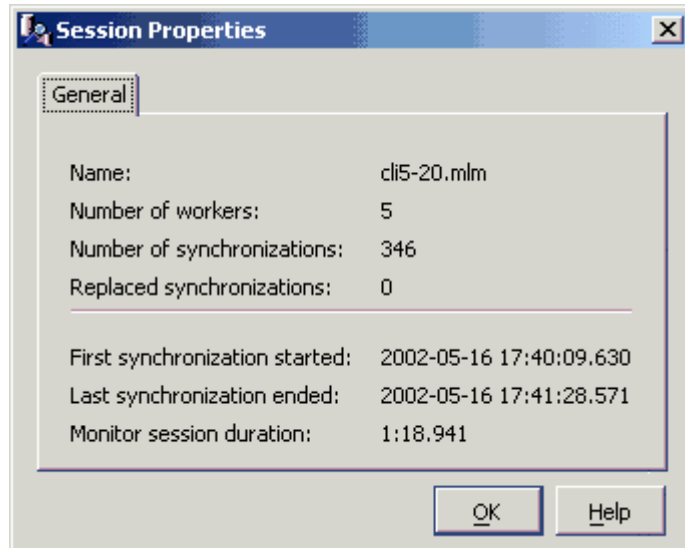
Restoring defaults

To restore default settings, delete the file *.mlMonitorSettings*. This file is stored in your user profiles directory.

Session Properties

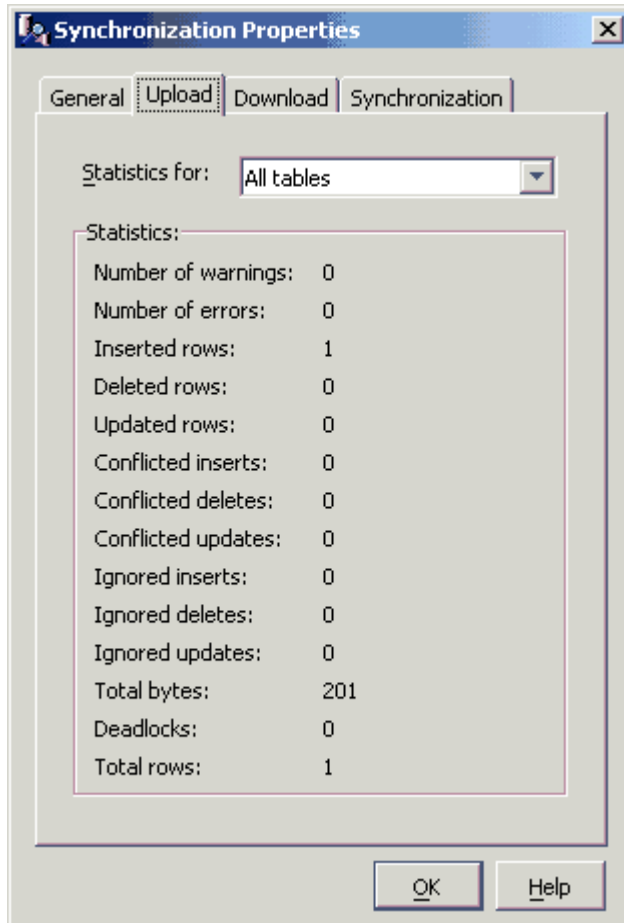
The Session Properties dialog provides basic information about the monitoring session.

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 "Statistical Properties" on page 247.

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.

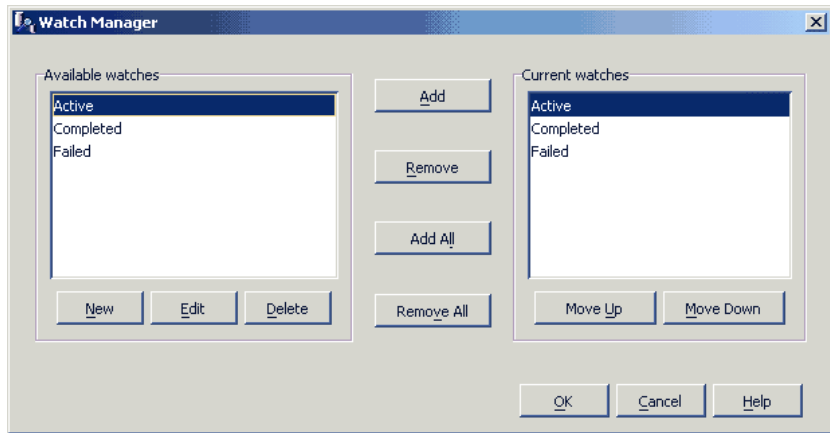
In the .csv file format, time durations are stored in milliseconds.

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 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:

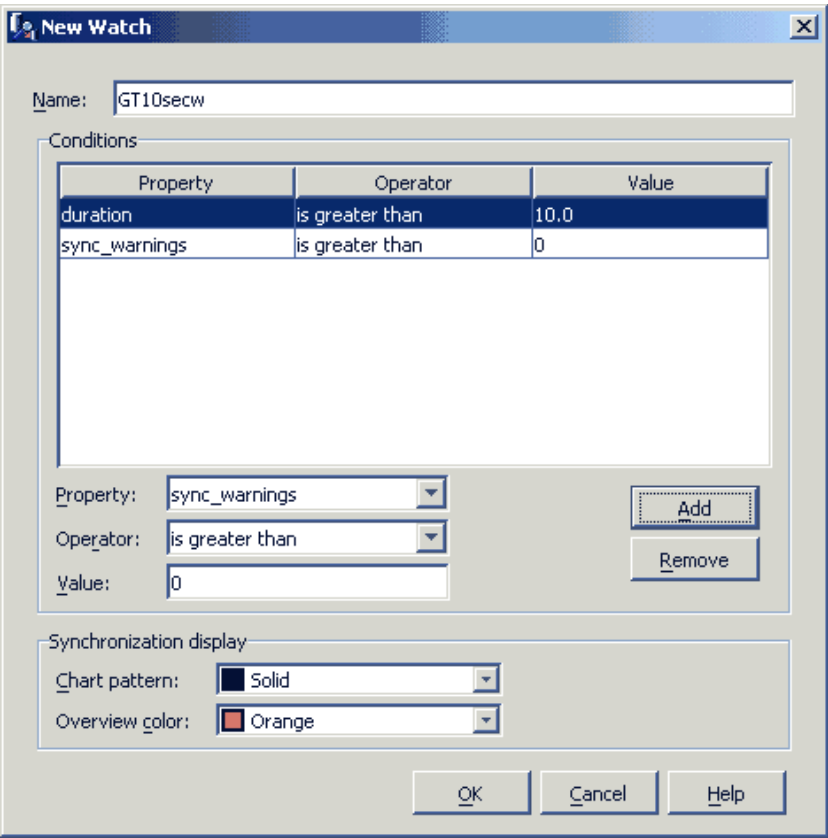
- 1 In the Watch Manager, click New.

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 "Statistical Properties" on page 247.
- 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.)

Example

The following screen shot shows a user-defined watch called GT10secw. This watch specifies that the Monitor should track synchronizations with a duration greater than 10 seconds that receive synchronization warnings (defined as the number of warnings being greater than zero), and that it should display them in the Overview pane as solid orange.



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). |

| Property | Notes |
|------------------------|--|
| download_filtered_rows | Number of fetched rows that were not downloaded to the MobiLink client because they matched rows that the client uploaded. |
| 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. |
| 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. |

| Property | Notes |
|----------------------|---|
| upload | Time for data to be uploaded to the consolidated database. |
| upload_bytes | Number of bytes uploaded from the synchronization client. |
| 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 <i>n.m</i> , where <i>n</i> is the stream and <i>m</i> is the thread number. |

C H A P T E R 1 0

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 | 252 |
| Choosing a user authentication mechanism | 254 |
| User authentication architecture | 255 |
| Providing initial passwords for users | 257 |
| Synchronizations from new users | 258 |
| Prompting end users to enter passwords | 259 |
| Changing passwords | 260 |
| Custom user authentication mechanisms | 261 |

About MobiLink users

MobiLink user names and passwords are separate from the user names and passwords needed to gain access to either the remote or the consolidated database. MobiLink user names and passwords are used only within the MobiLink system. They 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 users different rows, based on their user name.

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" on page 618.

- Use Sybase Central.
- Specify the `-zu+` command line option with dbmlsrv8. 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" on page 405.

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]" on page 335 of the book *ASA SQL Reference Manual*.

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

 For more information, see "user_name synchronization parameter" on page 397 of the book *UltraLite User's Guide*, and "password synchronization parameter" on page 384 of the book *UltraLite User's Guide*.

The MobiLink user name is stored in the *ml_user* MobiLink system table in the consolidated database. To ensure correct behavior of the MobiLink system, each client must be assigned a unique MobiLink user name.

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 "Adding user authentication to your application" on page 85 of the book *UltraLite User's Guide*.

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 on custom user authentication mechanisms, see "Custom user authentication mechanisms" on page 261.

🔗 For information on other security-related features of MobiLink and its related products, see the following locations:

- ◆ "Transport-Layer Security" on page 283
- ◆ "Encrypting an UltraLite database" on page 46 of the book *UltraLite User's Guide*
- ◆ "Keeping Your Data Secure" on page 387 of the book *ASA Database Administration Guide*.

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" on page 618.

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 258.

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 User 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* command line option specifies an ODBC connection to the consolidated database. The *-f* option specifies the file containing the user names and passwords.

🔗 For information on *dbmluser*, see "MobiLink user authentication utility" on page 618.


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" on page 405.
- ◆ **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:

- ◆ 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 parameters" on page 380 of the book *UltraLite User's Guide*.

- ◆ **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" on page 410.

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:

- ◆ 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 "Synchronization parameters" on page 380 of the book *UltraLite User's Guide*.

- ◆ **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" on page 410.

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 257 and "Synchronizations from new users" on page 258.

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


Custom user authentication mechanisms

You can choose to use a user authentication mechanism other than the built-in MobiLink mechanism.

To implement your own user authentication mechanism, you must write an *authenticate_user* script. The script then overrides the default MobiLink user authentication mechanism. It does not supplement the default 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, not present in the built-in MobiLink mechanism.

The *authenticate_user* script is executed immediately before, and in the same transaction as, the *begin_synchronization* script. The transaction is ended immediately after the *begin_synchronization* script.

 For more information, see "authenticate_user connection event" on page 446.

Java user authentication

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

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 11

Synchronizing Through a Web Server

About this chapter

This chapter describes how to route MobiLink synchronization through a Web server. The method is particularly useful for synchronizing across a firewall.

Contents

| Topic | Page |
|---|-------------|
| Introduction | 264 |
| Configuring MobiLink clients and servers for the Redirector | 266 |
| Configuring the Redirector (all versions) | 268 |
| Configuring NSAPI Redirector for Netscape Web servers | 270 |
| Configuring ISAPI Redirector for Microsoft Web servers | 272 |
| Configuring the servlet Redirector | 273 |

Introduction

This chapter describes how 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.

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.

MobiLink includes a Web server extension called the **Redirector** which 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 an MobiLink synchronization server is running again and resumes sending requests at that time.

Plug-ins are provided for the following Web servers:


- ◆ Netscape iPlanet Web servers
- ◆ Microsoft Web servers
- ◆ Web servers that support the Java Servlet API 2.2

The following sections describe how to configure your Web server to manage synchronization requests. First, there are generic instructions, and then there are specific instructions for each type of Web server.


Overview

❖ 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 266.


- 2 Ensure that the Redirector configuration file is on the same computer as the Web server.

 See "Copy redirector.config to the Web server" on page 268.

- 3 Modify the Redirector configuration file.

 See "Set up the Redirector configuration file" on page 268.

- 4 Perform Web server-specific configuration.

 See "Configuring ISAPI Redirector for Microsoft Web servers" on page 272, or "Configuring NSAPI Redirector for Netscape Web servers" on page 270, or "Configuring the servlet Redirector" on page 273.

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.

Configuring MobiLink clients

Set the following HTTP synchronization stream parameters on the MobiLink client:

- ◆ **host** the name or IP of the Web server.
- ◆ **port** the Web server port accepting HTTP 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 more information, see the following:

- ◆ "HTTP stream parameters" on page 403 of the book *UltraLite User's Guide*.
- ◆ "CREATE SYNCHRONIZATION USER statement [MobiLink]" on page 335 of the book *ASA SQL Reference Manual*

Configuring the MobiLink synchronization server

The MobiLink server must be started with the HTTP protocol. The following synchronization stream parameters apply to requests directed through Web servers:

- ◆ **port** for the HTTP protocol, MobiLink defaults to port 80. If the MobiLink synchronization server is running on the same machine as the Web server, this port 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.
- ◆ **unknown_timeout** This is the number of seconds to wait to receive the HTTP headers on a new (unknown) connection before it is closed. This setting is optional, and has a default value of 30 seconds.
- ◆ **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.

☞ For more information, see "-x option" on page 396.

Configuring the Redirector (all versions)

This section describes generic Web server configuration steps to set up the Redirector:

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 directory.

If 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.

Set up 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:

- ◆ **ML** used to list the computers running MobiLink synchronization server, in the form *ML=host:port*. *ML* is case sensitive .
- ◆ **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 .
- ◆ **REDIRECTOR_HOST** used to specify the machine name of the Web server running the Redirector. For example, *myCompany.com*.
- ◆ **REDIRECTOR_PORT** used to specify the port of the Web server running the Redirector. For example, *80*.
- ◆ **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.

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.

Note

You must also follow configuration procedures for the Web server that you are planning to use. Those procedures follow.

Example

Following is a sample *redirector.config* file. This file:

- ◆ specifies that the Redirector should check every 1800 seconds that the servers are functioning.
- ◆ defines three computers running MobiLink synchronization server that are able to process requests.
- ◆ specifies 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 NSAPI Redirector for Netscape Web servers

This section describes setup steps specific to Netscape iPlanet Web servers.

❖ To configure NSAPI Redirector for iPlanet:

- 1 Complete the steps in "Configuring the Redirector (all versions)" on page 268.
- 2 If necessary, copy the file *iaredirect.dll* to the computer that holds the Web server. This file is installed with MobiLink synchronization server, in the *MobiLink\redirector\nsapi* subdirectory of your SQL Anywhere directory.
- 3 Update the iPlanet Web server configuration file *obj.conf* as follows.

Sample file provided

A complete sample copy of *obj.conf*, preconfigured for 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.

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"  
funcs="redirector,initialize_redirector"  
Init fn="initialize_redirector"  
configFile="<location>/redirector.config"
```

- ◆ Specify the URL paths for MobiLink requests.

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.

These streams are sent using chunked HTTP. The default buffer size of 8 Kb is too small for some uploads.

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.

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>
```

Configuring ISAPI Redirector for Microsoft Web servers

This section contains instructions for configuring the Redirector for Microsoft Web servers to work with MobiLink.

❖ **To configure ISAPI Redirector for Microsoft Web servers:**

- 1 Complete the steps in "Configuring the Redirector (all versions)" on page 268.
- 2 Copy the file *iaredirect.dll*, which is included with the MobiLink synchronization server installation, 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.

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.

Configuring the servlet Redirector

The servlet version of the Redirector can be installed on a variety of Web servers that support Java Servlet API 2.2. For this release, it is supported for Apache Tomcat.

❖ To configure the servlet Redirector for Tomcat:

- 1 If necessary, change the Tomcat HTTP port.

Tomcat binds to port 8080 by default. If another Web server is using 8080 or there is some other conflict, you need to change the port number. To do this, open the file: `%CATALINA_HOME%/conf/server.xml` and search for 8080 (you will find it in a `<Connector>` tag). Change it to a port that isn't in use.

- 2 Install the servlet Redirector as a Web application:

- ◆ Copy the file *iaredirect.war* to `%CATALINA_HOME%/webapps`.
- ◆ Shut down and restart Tomcat. It will expand the war file and create the directory *iaredirect* for the Redirector Web application.
- ◆ Open the file `%CATALINA_HOME%/webapps/iaredirect/WEB-INF/web.xml`, and search for `redirector.config` (you'll find it under the `<init-param>` tag). Enter the correct path for the *redirector.config* file. For example, change *redirector.config* to `d:/redirector.config`. You must use a forward slash.
- ◆ Shut down and restart Tomcat. This allows the changes to take effect.
- ◆ Delete the file *iaredirect.war*. (It has been deployed and is no longer needed.)

The redirector will now be invoked with the following HTTP settings. Note that the `url_suffix` must exactly match the one below.

```
host=<tomcat host name>;port=<tomcat port
number>;url_suffix=iaredirect/servlet/redirect/ml/;
```


Running MobiLink Outside the Current Session

About this chapter This chapter describes how to run the MobiLink synchronization server as a daemon or service.

| Contents | <table><tr><th>Topic</th><th>Page</th></tr><tr><td>Running the UNIX MobiLink server as a daemon</td><td>276</td></tr><tr><td>Running the Windows MobiLink server as a service</td><td>277</td></tr><tr><td>Troubleshooting MobiLink server startup</td><td>282</td></tr></table> | Topic | Page | Running the UNIX MobiLink server as a daemon | 276 | Running the Windows MobiLink server as a service | 277 | Troubleshooting MobiLink server startup | 282 |
|--|--|-------|------|--|-----|--|-----|---|-----|
| Topic | Page | | | | | | | | |
| Running the UNIX MobiLink server as a daemon | 276 | | | | | | | | |
| Running the Windows MobiLink server as a service | 277 | | | | | | | | |
| Troubleshooting MobiLink server startup | 282 | | | | | | | | |

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.

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:**

- ◆ Use the `-ud` command line option when starting the MobiLink server.
For example:

```
dbmlsrv8 -c "dsn=ASA 8.0 Sample;uid=DBA;pwd=SQL" -ud
```

🔗 For more information, see "-ud option" on page 393.

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 in the Services folder 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, open the Services folder.
- 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" on page 499 of the book *ASA Database Administration Guide*.

❖ To remove a service (Sybase Central):

- 1 In Sybase Central, open the Services folder.
- 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, open the Services folder.
- 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 Adaptive Server Anywhere 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 8.0 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" on page 380.

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 *Local/System*, 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 *Local/System*, 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, open the Services folder.
- 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 an Adaptive Server Anywhere service from the Windows Service Manager. You can use Sybase Central to carry out all the service management for Adaptive Server Anywhere.

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 database server has started, it fails because it cannot find the database server. The sequence must be such that the database server is running when you start the MobiLink server.

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, as listed in the following table.

| Service | Default group |
|---------------------------------|---------------|
| Network server | ASANYServer |
| MobiLink Synchronization Server | 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 Open the Services folder.
- 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 member of each of a list of service groups 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 Open the Services folder.
- 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. On the server, you use the `-z` command line option. The startup information appears on the server window: you can use the `-o` option to log the results to an output file.

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>.

Contents

| Topic | Page |
|--|------|
| About transport-layer security | 284 |
| Invoking transport-layer security | 293 |
| Certificate authorities | 298 |
| Certificate chains | 299 |
| Enterprise root certificates | 300 |
| Globally signed certificates | 305 |
| Obtaining server-authentication certificates | 307 |
| Verifying certificate fields | 310 |

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

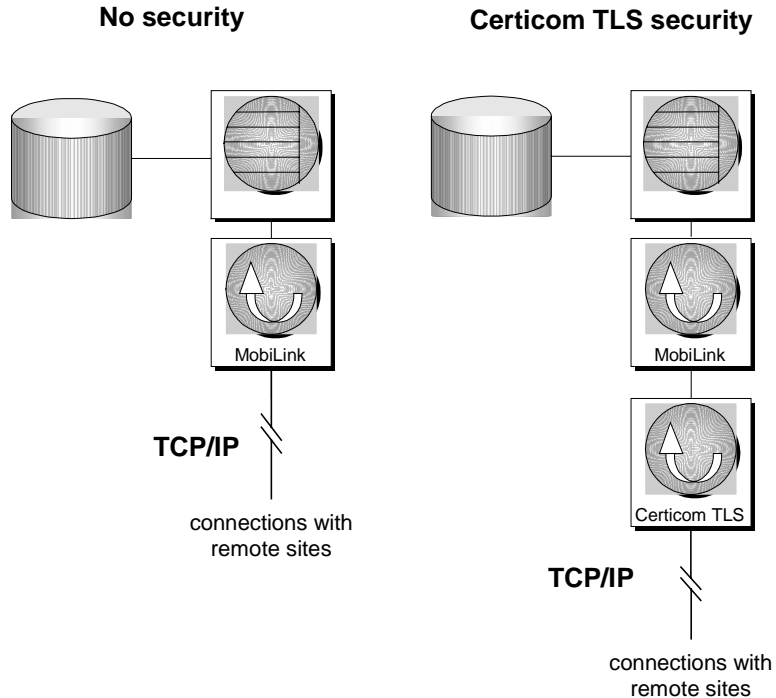
Public-key cryptography has many advantages. Using the public key, anyone can send a message that can be read only by the person who knows the matching private key. Likewise, someone can prove that they know a private key by using it to encrypt a message. To verify the identity of a key owner, you can 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:

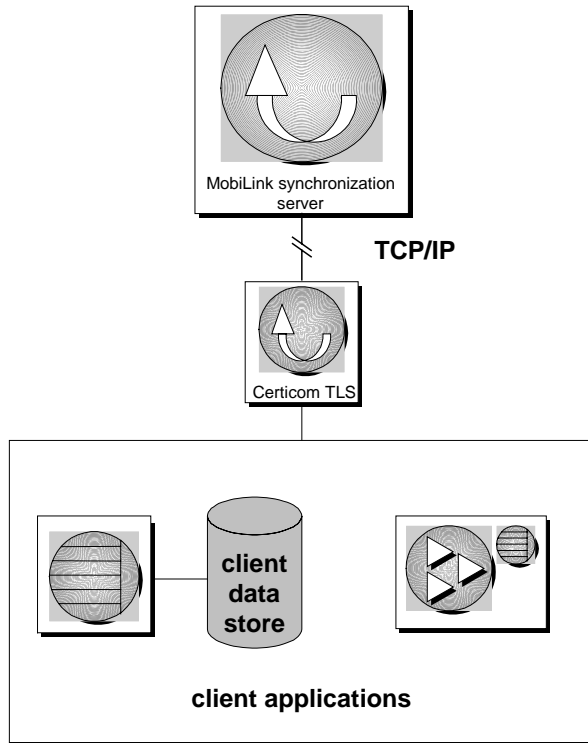


Transport-level security requires additional communication between a MobiLink client and the MobiLink synchronization server *before* the upload stream is sent. When a client initiates synchronization, it passes a message to the server. The client encrypts this message using the server's public key. 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
YmFzZSBjbmMuMQ8wDQYDVQQLEFAZTZWJhc2UxEjAQBjNVBAMUCVN5YmFzZSBBDQTAe
Fw05OTExMTcxODAlMzZaFw0wOTExMTcxODAlMzZaMGcxDDAKBgNVBAYTA1VTQTEl
MAkGALUECBMCQ0ExEzARBgNVBACtCkVtZXJ5dm1sbGUxPDASBgNVBAoUC1N5YmFz
ZSBjbmMuMQ8wDQYDVQQLEFANRUMxETAPBgNVBAMUC1vYm1saW5rMCAwEAYHKoZI
zj0CAQYFK4EEAAEDFwACAx0L37T06bGehBN1RVJcma/Y0h5xoyYwJDAOBgNVHQ8B
Af8EBAMCAf4wEgYDVR0TAQH/BAgwBgEB/wIBCAjALBgqhkJOPQBBQADMqAwLgIV
Ad+4I1uT7/1URk7SfZTTiYqnR/rAAhUCCRGc62100Mt69TxusuBvI2OY=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIBrDCCAWigAwIBAgIFMTIzNDUwCwYHKoZIzj0EAQUAMGsxDDAKBgNVBAYTA1VT
QTElMAkGALUECBMCQ0ExEzARBgNVBACtCkVtZXJ5dm1sbGUxPDASBgNVBAoUC1N5
YmFzZSBjbmMuMQ8wDQYDVQQLEFAZTZWJhc2UxEjAQBjNVBAMUCVN5YmFzZSBBDQTAe
Fw05OTExMTcxODAlMzZaFw0wOTExMTcxODAlMzZaMGcxDDAKBgNVBAYTA1VTQTEl
MAkGALUECBMCQ0ExEzARBgNVBACtCkVtZXJ5dm1sbGUxPDASBgNVBAoUC1N5YmFz
ZSBjbmMuMQ8wDQYDVQQLEFAZTZWJhc2UxEjAQBjNVBAMUCVN5YmFzZSBBDQTAxMBAg
ByqGSM49AgEGBSuBBAABAAcAAgFUVb7gQh0cy6XgxsRQUPaMCMiYk6MmMCQwDgYD
VR0PAQH/BAQDAgH+MBIGALUEwEwEB/wQIMAYBAf8CAQowCwYHKoZIzj0EAQUAAzEA
MC4CFQITrvY7k6c3jy37KyC4iDj6UNGWnQIVA/qAja8SA2W7SyAfQ23oCY7n29Ss
-----END CERTIFICATE-----
-----BEGIN ENCRYPTED PRIVATE KEY-----
ME4wGgYJKoZIhvcNAQUDMA0ECL+NqY7WeMr/AgEFBDAZTKkSudCw2sUC45GKQaTr
xclepizwr9g5jm6wK8cCqOBfgZxs/Ne8eC2sn2klqlM=
-----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.

Security tips

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, but certificates ensure MobiLink clients that they are talking to a trusted MobiLink synchronization server.

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 can 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 then invoke server authentication on the dbmlsrv8 command line.

🔗 For information about how to invoke transport-layer security on Adaptive Server Anywhere clients, see "CREATE SYNCHRONIZATION USER statement [MobiLink]" on page 335 of the book *ASA SQL Reference Manual*.

🔗 For information about how to invoke server authentication for UltraLite clients, see "Adding synchronization to your application" on page 94 of the book *UltraLite User's Guide*.

🔗 For information about how to invoke server authentication for Adaptive Server Anywhere, see "-x option" on page 396.

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.

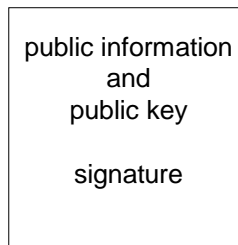
Self-signed certificates

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



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: MEC
Common Name: MobiLink
Serial Number: 2000.02.29.01
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
```

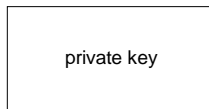
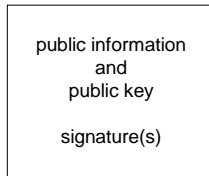
The response to each question should be a string, except for the number of years to the expiry date, which must be an integer.

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.

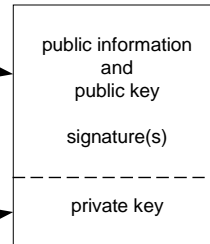
- ◆ **serv1.crt** This file contains the same information as the above two files, combined into one file. It is intended for use with a MobiLink synchronization server. The server sends the public information to identify itself to clients. It requires the private key to decode messages returned by the clients. You must restrict access to this file. It, too, contains a copy of the private key, protected only by the password.

Public certificate



Private key file

Server identity



You can create a server identity certificate by concatenating a public certificate and the matching private file.

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 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
ADDRESS '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 8.0 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:

```
dbmlsrv8 -c "dsn=UltraLite 8.0 Sample;uid=DBA;pwd=SQL"
-x tcpip ( security=ecc_tls ( certificate=serv1.crt;
certificate_password=password ) )
```

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.

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.

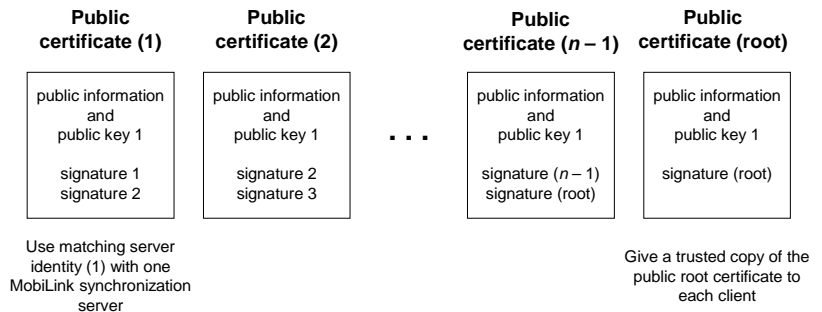
To obtain valid copies of a public key, users of this system typically obtain copies of a self-signed certificate owned by the certificate authority. To foil impostors, the certificate must be obtained by reliable means.

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 comprises 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

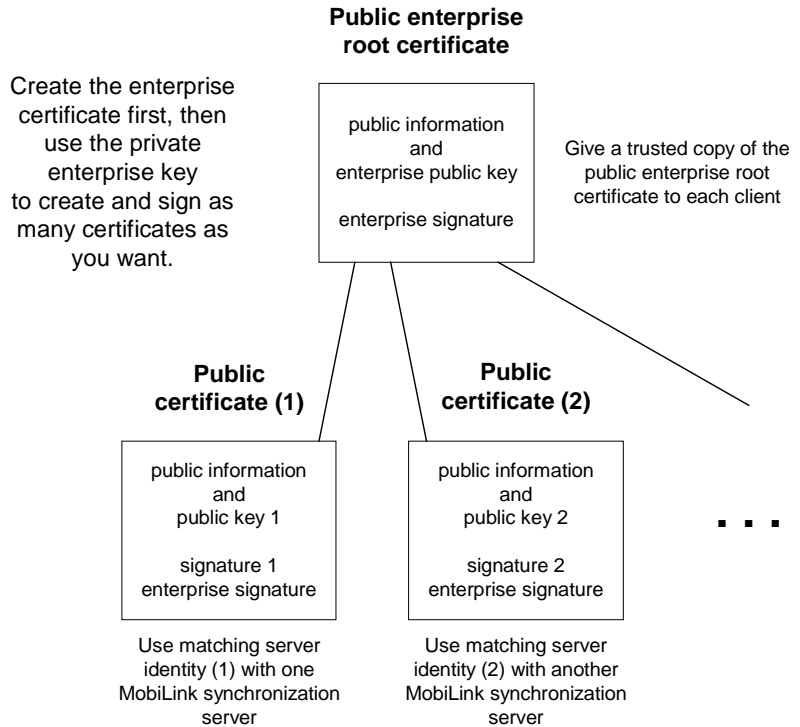
A deployment of MobiLink that involves multiple servers can be improved by assigning each server a unique certificate also signed by a common root certificate. A certificate authority within the enterprise holds the root certificate.

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: MEC
Common Name: MobiLink
Serial Number: 2000.02.29.02
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: MEC
Common Name: MobiLink
Serial Number: 2000.02.29.03
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: MEC
Common Name: MobiLink
Serial Number: 2002.02.29.04
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: 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 '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 8.0 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.

```
dbmlsrv8 -c "dsn=UltraLite 8.0 Sample;uid=DBA;pwd=SQL"
-x tcpip ( port=3333;
    security=ecc_tls ( certificate=serv1.crt;
    certificate_password=password3 ) )

dbmlsrv8 -c "dsn=UltraLite 8.0 Sample;uid=DBA;pwd=SQL"
-x tcpip ( port=4444;
    security=ecc_tls ( certificate=serv2.crt;
    certificate_password=password4 ) )
```

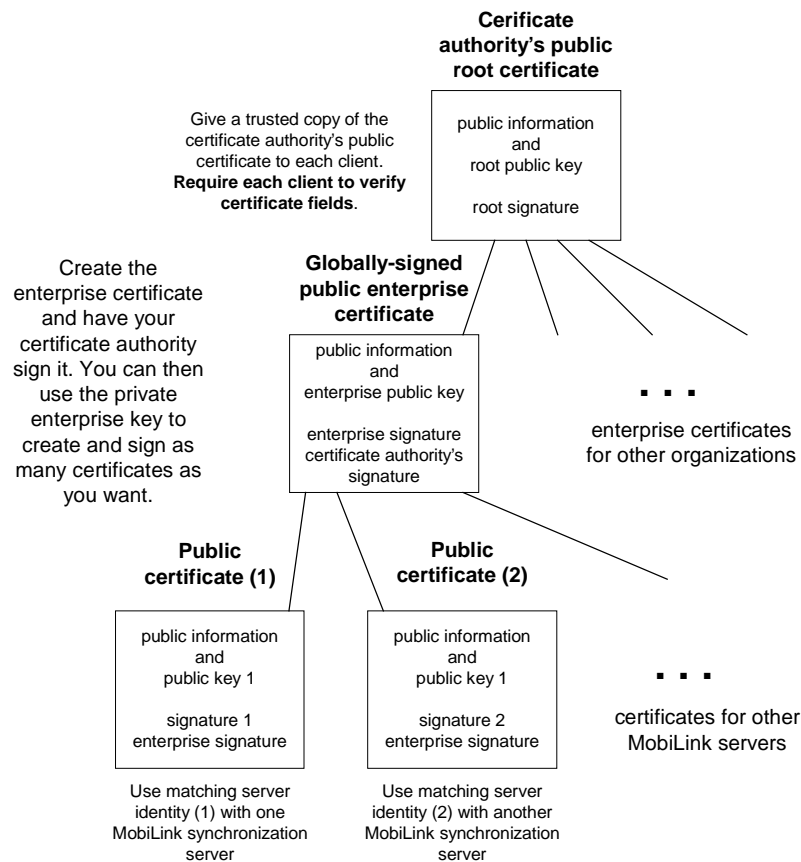
Globally signed certificates

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 8 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 ec163a02. Generating key pair (please
wait)...
Country: CA
State: Ontario
Locality: Waterloo
Organization: Sybase, Inc.
Organizational Unit: MEC
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:

```
dbmlsrv8 -c "dsn=UltraLite 8.0 Sample;uid=DBA;pwd=SQL"
-x tcpip ( security=ecc_tls( certificate=global2.crt;
certificate_password=password5 ) )
```

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
ADDRESS '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 8.0 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: MEC
Common Name: MobiLink
Serial Number: 2000.02.29.06
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: MEC
Common Name: MobiLink
Serial Number: 2000.02.29.07
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.

```
dbmlsrv8 -c "dsn=UltraLite 8.0 Sample;uid=DBA;pwd=SQL"
-x tcpip ( port=3333;security=ecc_tls (
certificate=serv6.crt;
certificate_password=password6 ) )

dbmlsrv8 -c "dsn=UltraLite 8.0 Sample;uid=DBA;pwd=SQL"
-x tcpip ( port=4444;security=ecc_tls (
certificate=serv7.crt;
certificate_password=password7 ) )
```

In addition, you must ensure that each client trusts your certificate authority's root certificate.

PART TWO

MobiLink Tutorials

This part provides hands-on tutorials that introduce you to the basic techniques of creating MobiLink synchronization systems.

C H A P T E R 1 4

**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 | <table><tr><th>Topic</th><th>Page</th></tr><tr><td>Introduction</td><td>316</td></tr><tr><td>Lesson 1: Creating and populating your databases</td><td>318</td></tr><tr><td>Lesson 2: Running the MobiLink synchronization server</td><td>322</td></tr><tr><td>Lesson 3: Running the MobiLink synchronization client</td><td>324</td></tr><tr><td>Tutorial cleanup</td><td>326</td></tr><tr><td>Summary</td><td>327</td></tr><tr><td>Further reading</td><td>328</td></tr></table> | Topic | Page | Introduction | 316 | Lesson 1: Creating and populating your databases | 318 | Lesson 2: Running the MobiLink synchronization server | 322 | Lesson 3: Running the MobiLink synchronization client | 324 | Tutorial cleanup | 326 | Summary | 327 | Further reading | 328 |
|---|---|-------|------|--------------|-----|--|-----|---|-----|---|-----|------------------|-----|---------|-----|-----------------|-----|
| Topic | Page | | | | | | | | | | | | | | | | |
| Introduction | 316 | | | | | | | | | | | | | | | | |
| Lesson 1: Creating and populating your databases | 318 | | | | | | | | | | | | | | | | |
| Lesson 2: Running the MobiLink synchronization server | 322 | | | | | | | | | | | | | | | | |
| Lesson 3: Running the MobiLink synchronization client | 324 | | | | | | | | | | | | | | | | |
| Tutorial cleanup | 326 | | | | | | | | | | | | | | | | |
| Summary | 327 | | | | | | | | | | | | | | | | |
| Further reading | 328 | | | | | | | | | | | | | | | | |

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" on page 75 of the book *ASA Getting Started*.

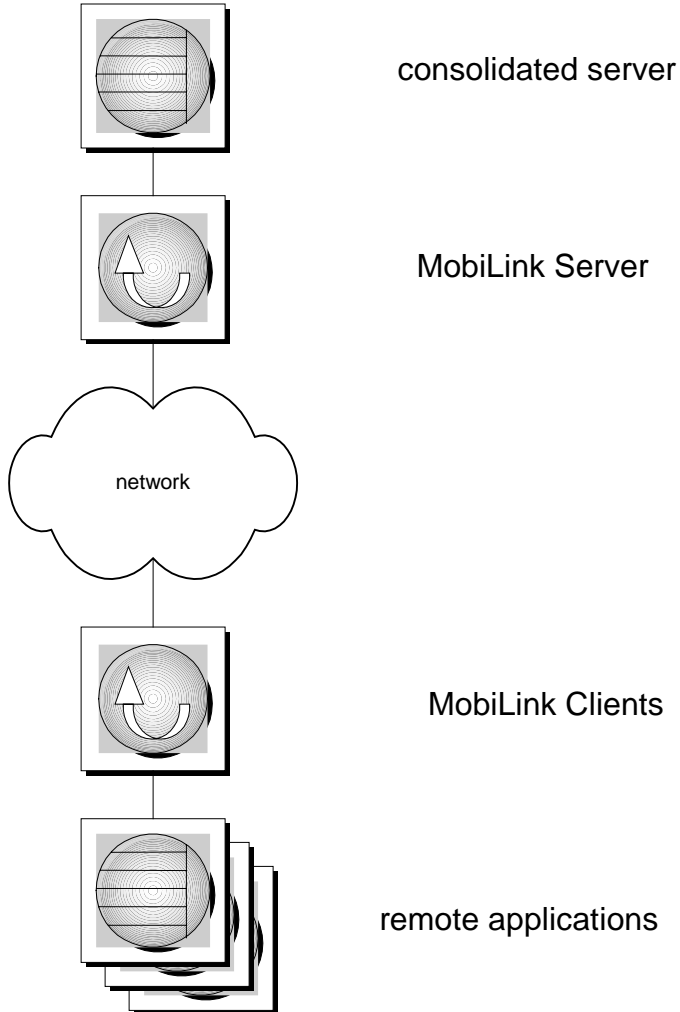
Goals

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 8 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" on page 451 of the book *ASA Database Administration Guide*.

❖ **To create ODBC data sources:**

- 1 Open a command prompt and navigate to the *Samples\MobiLink\AutoScripting* subdirectory of your SQL Anywhere 8 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" on page 453 of the book *ASA Database Administration Guide*.

- 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 8▶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 8 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 118.

☞ For more information about creating subscriptions and publications, see "Publishing data" on page 119.

☞ For more information about creating databases, see "The Initialization utility" on page 465 of the book *ASA Database Administration Guide* and "Creating a database using the dbinit command-line utility" on page 466 of the book *ASA Database Administration Guide*.

☞ For more information about running Interactive SQL, see "The Interactive SQL utility" on page 472 of the book *ASA Database Administration Guide* and "Using Interactive SQL" on page 75 of the book *ASA Getting Started*.

☞ For more information about SELECT statements, see "SELECT statement" on page 526 of the book *ASA SQL Reference Manual*.

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 8 installation.
- 2 Run the following command line:

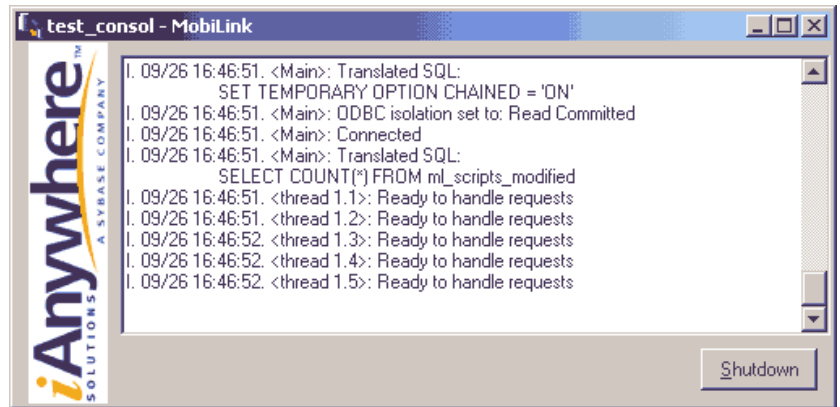
```
dbmlsrv8 -c "dsn=test_consol" -o mlserver.mls -v+  
-dl -za -zu+
```

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" on page 384.
- ◆ **-o** The `-o` option is used to specify the message log file. For more information, see "-o option" on page 387.
- ◆ **-v+** The `-v+` option sets verbose logging on. For more information, see "-v option" on page 393.
- ◆ **-dl** The `-dl` option sets the display log feature ON. For more information, see "-dl option" on page 386.
- ◆ **-za** The `-za` option turns automated scripting ON. For more information, see "-za option" on page 402.
- ◆ **-zu+** The `-zu+` option automates the user authentication process. For more information, see "-zu option" on page 405.

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 *dbmlsrv8*, 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 18.

For a complete list of *dbmlsrv8* options, see "MobiLink Synchronization Server Options" on page 379.

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:

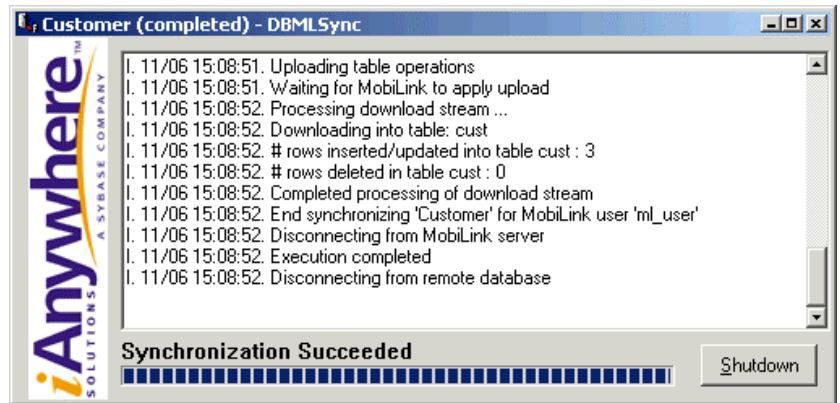
- 1 Open a command prompt and navigate to the *Samples\MobiLink\AutoScripting* subdirectory of your SQL Anywhere 8 installation.
- 2 Run the following command line:

```
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" on page 384.
- ◆ **-o** Specify the message log file. For more information, see "-o option" on page 387.
- ◆ **-v** Verbose operation. For more information, see "-v option" on page 393.
- ◆ **-e** Extended options. Specifying "SendColumnNames=ON" sends column names to MobiLink. This is required when you use -za in the dbmlsrv8 command line. For more information, see "SendColumnNames" on page 420.

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 dbmlsync, see "MobiLink synchronization client" on page 410.

For more information about remote clients, see "MobiLink clients" on page 21.

For more information about dbmlsync, see "Initiating synchronization" on page 138.

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 8►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 8 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 117.

 For more information about running the MobiLink synchronization server, see "Synchronization Basics" on page 9.

 For more information about synchronization scripting, see "Writing Synchronization Scripts" on page 47.

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 | 330 |
| | Lesson 1: Creating your databases | 331 |
| | Lesson 2: Creating scripts for your synchronization | 335 |
| | Lesson 3: Running the MobiLink synchronization server | 338 |
| | Lesson 4: Running the MobiLink synchronization client | 340 |
| | Lesson 5: Monitoring your MobiLink synchronization using log files | 342 |
| | Tutorial cleanup | 344 |
| | Further reading | 345 |

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" on page 49 of the book *Introducing SQL Anywhere Studio*.

🔗 For more information about synchronization, see "Tutorial: Synchronizing Adaptive Server Anywhere Databases" on page 315.

🔗 For more information about synchronization scripts, see "Introduction to synchronization scripts" on page 48.

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 8 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 8►Sybase Central.
Sybase Central appears.
- 2 In Sybase Central, choose Tools►Adaptive Server Anywhere 8►Create Database.
The Create Database wizard appears. Click Next.
- 3 Leave the default of Create a Database on this Machine. 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 8 databases.

❖ To create ODBC data sources:

- 1 In Sybase Central, choose Tools►Adaptive Server Anywhere 8►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 8.0 and click Finish.

The ODBC Configuration for Adaptive Server Anywhere 8.0 dialog appears.

- 3 Click the ODBC tab and enter **test_consol** as the Data Source Name.
- 4 Click the Login tab and enter a User ID **dba** and password **SQL**.
- 5 Click the Database tab. Click Browse to locate *test_consol.db*. Clear the Automatically Shut Down Database After Last Disconnect checkbox.
- 6 Enter **test_consol** under Server Name.
- 7 Click OK to return to the ODBC Data Source Administrator.
- 8 Repeat steps 2 through 7 using the name **test_remote** instead of **test_consol** for the ODBC connection name, database file name, and server name.
- 9 Click OK to close the ODBC Data Source Administrator.

❖ **To verify your new data sources:**

- 1 In Sybase Central, choose Tools►Adaptive Server Anywhere 8►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 8►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 build a 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 "The consolidated database" on page 12.

🔗 For more information about remote databases, see "MobiLink clients" on page 21.

🔗 For more information about Interactive SQL, see "The Interactive SQL utility" on page 472 of the book *ASA Database Administration Guide*.

🔗 For more information about creating ODBC data sources, see "The Data Source utility" on page 451 of the book *ASA Database Administration Guide*.

🔗 For more information about defining publications and subscriptions, see "Publishing data" on page 119.

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:

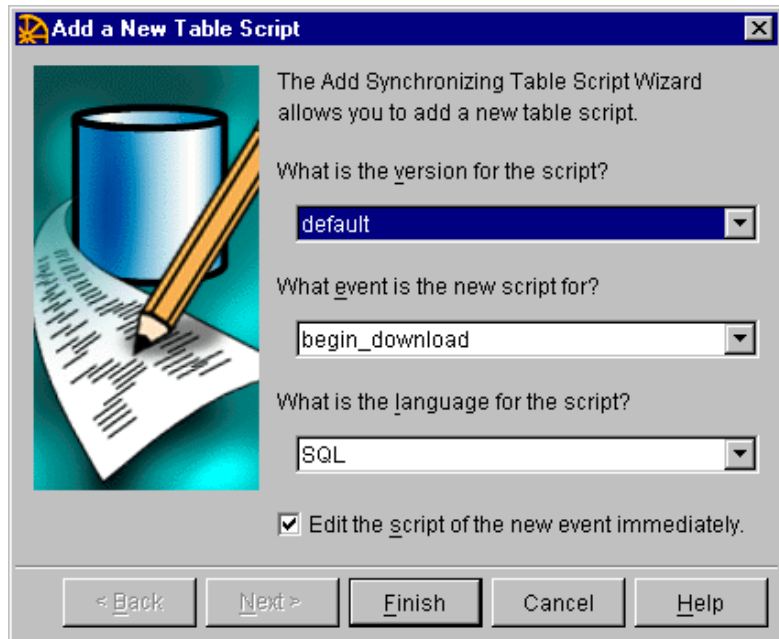
- 1 In the MobiLink Synchronization plug-in of Sybase Central, select the Tables folder and double-click DBA.
You will see two tables, *emp* and *cust*.
- 2 Right-click each table and choose Add to Synchronized Tables.

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" on page 549 and "download_cursor cursor event" on page 474.

🔗 For more information about script versions, see "Script versions" on page 61.

🔗 For more information about adding scripts, see "Adding and deleting scripts in your consolidated database" on page 63.

🔗 For more information about writing table scripts, see "Table scripts" on page 58.

🔗 For more information about writing synchronization scripts, see "Writing Synchronization Scripts" on page 47.

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:

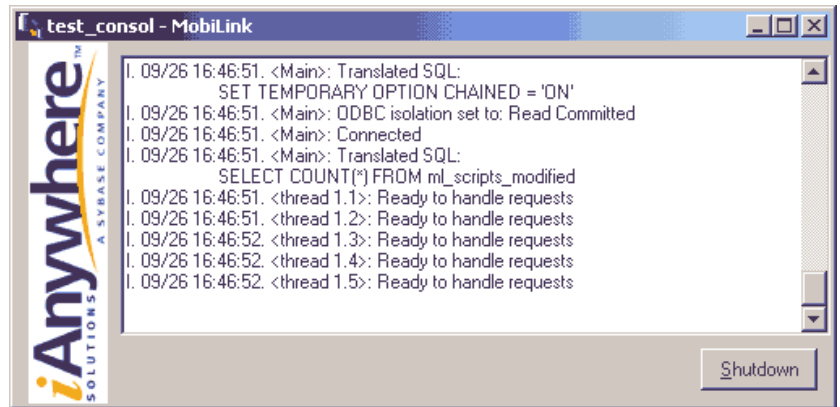
```
dbmlsrv8 -c "dsn=test_consol" -o mlserver.mls -v+  
-dl -zu+
```

This command line specifies the following options:

- ◆ **-c** Supply database connection parameters. For more information, see "-c option" on page 384.
- ◆ **-o** Specify the message log file. For more information, see "-o option" on page 387.
- ◆ **-v+** Sets verbose logging on. For more information, see "-v option" on page 393.
- ◆ **-dl** Sets the display log feature ON. For more information, see "-dl option" on page 386.
- ◆ **-zu+** Automates the user authentication process. For more information, see "-zu option" on page 405.

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 *dbmlsrv8*, 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 *dbmlsrv8*, see "The MobiLink synchronization server" on page 18.

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:

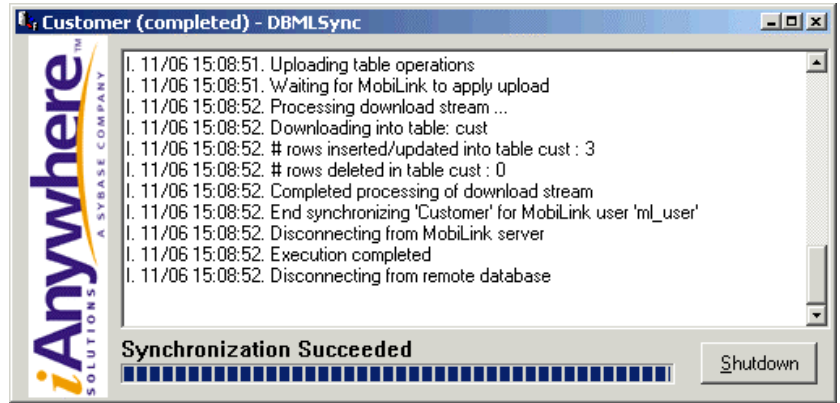
- 1 Ensure the MobiLink synchronization server is running by looking for the MobiLink icon in the system tray.
- 2 Open a command prompt and navigate to the *Samples\MobiLink\AutoScripting* subdirectory of your SQL Anywhere 8 installation.
- 3 Run the following command line:

```
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" on page 384.
- ◆ **-o** Specify the message log file. For more information, see "-o option" on page 387.
- ◆ **-v+** Verbose operation. For more information, see "-v option" on page 393.

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 dbmlsync, see "MobiLink synchronization client" on page 410.

For more information about synchronization, see "The synchronization process" on page 24.

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 19.

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 8►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 8 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 18.

 For more information about synchronization scripting, see "Writing Synchronization Scripts" on page 47 and "Synchronization Events" on page 433.

 For an introduction to other methods of synchronization such as timestamp, see "Synchronization Techniques" on page 83.

 For information about testing your scripts in Sybase Central, see "Testing script syntax" on page 78.

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 | 348 |
| Lesson 1: Create your databases | 349 |
| Lesson 2: Starting the MobiLink synchronization server | 355 |
| Lesson 3: Running the MobiLink synchronization client | 356 |
| Summary | 357 |
| Further reading | 358 |

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 329.

✍ For more information about running the MobiLink synchronization server, see "Synchronization Basics" on page 9.

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.

Completed SQL scripts for this tutorial are available in the *Samples/MobiLink/Autoscripting/Oracle* subdirectory of your SQL Anywhere 8 installation.

Create a directory called *OracleTut* for this tutorial, for example *c:/OracleTut*. Copy *build_consol.sql* and *build_remote.sql* from the location above into your *OracleTut* directory.

The following SQL statements, contained in *build_consol.sql*, 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, 'emp1' );
insert into emp ( emp_id, emp_name ) values (
emp_sequence.nextval, 'emp2' );
insert into emp ( emp_id, emp_name ) values (
emp_sequence.nextval, 'emp3' );
commit;
insert into cust ( cust_id, emp_id, cust_name ) values (
cust_sequence.nextval, 1, 'cust1' );
insert into cust ( cust_id, emp_id, cust_name ) values (
cust_sequence.nextval, 1, 'cust2' );
insert into cust ( cust_id, emp_id, cust_name ) values (
cust_sequence.nextval, 2, 'cust3' );
commit;
```

The following SQL statements, contained in *build_remote.sql*, 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 8►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 8 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 8►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" on page 472 of the book *ASA Database Administration Guide*.

❖ 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:

```
dbmlsrv8 -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" on page 384.
- ◆ **-o** Specifies the message log file. For more information, see "-o option" on page 387.
- ◆ **-v+** Sets verbose logging on. For more information, see "-v option" on page 393.
- ◆ **-dl** Sets the display log feature ON.
- ◆ **-za** Turns automated scripting ON. For more information, see "-za option" on page 402.
- ◆ **-zu+** Automates the user authentication process. For more information, see "-zu option" on page 405.

Further reading

 For more information about dbmlsrv8, see "MobiLink synchronization server" on page 380.

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" on page 384.
- ◆ **-o** Specify the message log file. For more information, see "-o option" on page 387.
- ◆ **-v+** Verbose operation. For more information, see "-v option" on page 393.
- ◆ **-e** Extended options. Specifying "SendColumnNames=ON" sends column names to MobiLink. For more information, see "-e option" on page 386.

Further reading

 For more information about *dbmlsync*, see "MobiLink synchronization client" on page 410.

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 18.

 For more information about synchronization scripting, see "Writing Synchronization Scripts" on page 47, and "Synchronization Events" on page 433.

 For an introduction to other methods of synchronization such as timestamp, see "Synchronization Techniques" on page 83.

C H A P T E R 1 7

Using MobiLink Sample Applications

About this chapter This chapter uses MobiLink sample applications to describe 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 in Java or .NET synchronization logic.

Contents

| Topic | Page |
|--------------------|------|
| Introduction | 360 |
| The CustDB sample | 361 |
| The Contact sample | 365 |

Introduction

This chapter introduces you to two MobiLink sample applications, CustDB and Contact. These samples are a valuable resource for the MobiLink developer. They provide you with examples of how to implement many of the techniques you will need to develop MobiLink applications.

For example, 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 remote databases.

🔗 For more information, see "Partitioning rows among remote databases" on page 91.

- ◆ **Row subsets** Different remote users get different sets of rows from the *ULOrder* table.

🔗 For more information, see "Partitioning rows among remote databases" on page 91.

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

🔗 For more information about this technique, see "Timestamp-based synchronization" on page 86.

- ◆ **Snapshot synchronization** The *ULProduct* table is synchronized using a simple method that downloads all rows each time synchronization occurs.

🔗 For more information about this technique, see "Snapshot synchronization" on page 88.

- ◆ **Primary key pools to maintain unique primary keys** Ensuring that primary key values are unique across a complete MobiLink installation is essential. The primary key pool method used in this application is a practical method for ensuring unique primary keys.

🔗 For more information about this technique, see "Maintaining unique primary keys using key pools" on page 100.

The CustDB sample

The CustDB sample application includes an UltraLite MobiLink client and a consolidated database. The application has been designed to illustrate several common techniques. To get the most out of this chapter, you should study the sample application as you read.

☞ Other samples are based on the Contact sample. For more information, see "The Contact sample" on page 365.

This section describes the pieces that make up the code for the CustDB sample application and database. The sample SQL scripts are in the *Samples\MobiLink\CustDB* subdirectory of your SQL Anywhere installation, and the application code is in *Samples\UltraLite\CustDB*. Platform-specific user interface code is in subdirectories of *Samples\UltraLite\CustDB* named for each operating system.

The CustDB sample database

The CustDB sample database is an Adaptive Server Anywhere database file. CustDB serves both as the MobiLink consolidated database and as an UltraLite reference database.

In addition, SQL scripts are provided so that you can rebuild the consolidated database or use a Sybase Adaptive Server Enterprise, Microsoft SQL Server, Oracle, or IBM DB2 database as a consolidated database.

☞ For more information about preparing a database for use as a consolidated database, see "Creating a consolidated database" on page 13.

The SQL scripts for the CustDB sample application perform the following operations. On databases other than Adaptive Server Anywhere, these scripts can be run only after the MobiLink system tables have been added.

- ◆ **Create the CustDB tables** The table definitions are included in the scripts.
- ◆ **Add the data** The data for the CustDB database is added.
- ◆ **Add the synchronization scripts** The synchronization scripts control what happens when an application synchronizes. These scripts are added to the database.

The scripts implement the same features, but in a way that is appropriate for each database management system.

Creating a DB2 consolidated database for CustDB

The files needed to create a DB2 consolidated database for the CustDB sample application are as follows:

- ◆ *Samples\MobiLink\CustDB\custdb2.sql*.
- ◆ *Samples\MobiLink\CustDB\custdb2.java* and its compiled form, *custdb2.class*.
- ◆ *Samples\MobiLink\CustDB\custdb2setup.java* and its compiled form, *custdb2setup.class*.

The *custdb2.class* file must be copied to the *SQLLIB\FUNCTION* directory on the DB2 server machine. This file contains the implementations of the procedures created in *custdb2.sql*.

❖ To set up a DB2 consolidated database for the sample application:

- 1 Create a DB2 database on the DB2 server. Ensure that the default tablespace (usually called USERSPACE1) uses 8 Kb pages. You may have to delete the default tablespace and re-create it with 8 Kb pages. Consult your DB2 documentation for more information.
- 2 If necessary, change the connect command in *custdb2.sql*.
- 3 Start a DB2 Command Window from either the server or client machine. Enter the following command at the prompt:

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

- 4 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.

If you use a name other than CustDB, you must change the connection code in *custdb2setup.java* and recompile it. In the following example, *jdk11dir* is the path to your JDK 1.1 installation.

```
javac -g -classpath  
jdk11dir\lib\classes.zip;%db2path%\java\db2java.zip;  
%db2path%\java\runtime.zip custdb2setup.java
```

- 6 Run the *custdb2setup Java* application on the DB2 client machine. In the following example, *username* and *password* are the username and password for connecting to the CustDB ODBC data source.

```
java custdb2setup [username password]
```


The *custdb2setup.class* file contains a Java application that 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.

Application logic source code

The application logic for UltraLite clients is held in the *Samples\UltraLite\CustDB* directory.

The embedded SQL logic is held in *Samples\UltraLite\CustDB\custdb.sql*. It holds the SQL statements needed to query and modify information from the UltraLite database and also holds the calls needed to start synchronization with the consolidated database.

The C++ API logic is held in *Samples\UltraLite\CustDB\custdbapi.cpp*.

The user-interface features are held separately, in platform-specific subdirectories.


 For more information, see "Source file locations" on page 18 of the book *UltraLite User's Guide*.

Synchronization logic source code

The source for the synchronization features of the sample application is the *Samples\MobiLink\CustDB\custdb.sql* file, with *custase.sql*, *custmss.sql*, *custora.sql*, and *custdb2.sql* holding versions for other database-management systems. The scripts in *custdb.sql* are included in the *custdb.db* Adaptive Server Anywhere database.


Cursor-based upload scripts

The synchronization logic for CustDB uses cursor-based upload scripts. Statement-based uploads are recommended for most purposes. You can find examples of statement-based uploads in the Contact sample.

 For more information, see "The Contact sample" on page 365.

Inspecting the
synchronization
scripts

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

 For information about running Sybase Central, see "Lesson 8: Browse the consolidated database" on page 35 of the book *UltraLite User's Guide*.

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*.

Examples

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
```

Some table-level scripts, such as this one, are associated with cursor events. These scripts are primarily responsible for the movement of data. As they play such a central role, this kind of table script is described separately as a **cursor script**.

Further reading

🔗 For more information about script types, see "Script types" on page 55.

🔗 For reference material, including detailed information about each script and its parameters, see "Synchronization Events" on page 433.

The Contact sample

The Contact sample contains an Adaptive Server Anywhere consolidated database and two Adaptive Server Anywhere remote databases. It illustrates many of the synchronization techniques discussed in this chapter.

Although the consolidated database is an Adaptive Server Anywhere database, the synchronization scripts are made up of simple SQL statements that should work with minimal changes on other database management systems.

The Contact sample is held in the *Samples\MobiLink\Contact* subdirectory of your SQL Anywhere directory. For an overview, see *Samples\MobiLink\Contact\readme.txt*.

Building the Contact sample

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 file named *consol.db* and loads the database schema, some data, synchronization scripts and MobiLink user names into the database.
- ◆ Creates subdirectories named *remote_1* and *remote_2*. Each subdirectory holds a single remote database.
- ◆ Creates a remote database file named *remote_1\remote.db* and loads information common to all remote databases into the file. This database is used as a template for the other remote database.
- ◆ Copies *remote_1\remote.db* into *remote_2\remote.db*.
- ◆ Applies customizations to each remote database. These customizations include a global database identifier, a MobiLink user name, and subscriptions to two publications.

❖ To run the build batch file (Command line):

- 1 At a command prompt, navigate to the *Samples\MobiLink\Contact* subdirectory of your SQL Anywhere installation.

- 2 Execute the following command:

```
build
```

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 these batch files in a text editor.

❖ To run the Contact sample:

- 1 Start the MobiLink synchronization server.

- ◆ At a command prompt, navigate to the *Samples\MobiLink>Contact* directory and execute the following command:

```
step1
```

This is 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.

- ◆ At a command prompt, navigate to the *Samples\MobiLink>Contact* directory and execute the following command:

```
step2
```

This is a batch file that synchronizes both remote databases.

- 3 Shut down the MobiLink synchronization server.

- ◆ At a command prompt, navigate to the *Samples\MobiLink>Contact* directory and execute the following command:

```
step3
```

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

To explore how the synchronization scripts work 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 held in the following files:

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

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

- ◆ **SalesRep** This table holds one row for each sales representative. Each remote database belongs to a single sales representative.

In each remote database, the table holds the following columns:

- ◆ **rep_id** A primary key column that holds 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, the table holds 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** The identifier of the sales representative for the customer.

In the consolidated database, the table has the following additional 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 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, the table holds 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** This table holds a row for each product sold by the company. It is held in a separate publication so that remote databases can synchronize the table separately.

In each remote database, the table holds 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 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**. In a security-conscious environment, these user IDs and passwords must be changed.

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 ID 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 user 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.

Synchronizing sales representatives in the Contact sample

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 holds a single row from the SalesRep table, corresponding to the remote database owner's MobiLink user name.

These stringent rules make synchronization of this table simple.

Download

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. No **download_delete_cursor** script is needed.

Upload

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

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 holds only the customers they are assigned to.

Download

The following **download_cursor** script downloads all customers belonging to the sales representative for which information has changed since the last time the sales representative downloaded information. It also downloads only customers marked active.

```
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
```

The following **download_delete_cursor** script downloads all customers for which information has changed since the last time the sales representative downloaded information. It deletes from the remote database all customers marked as inactive or who do not belong 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 inactive.

When territories are realigned, this script downloads those customers no longer assigned to the sales representative. It also downloads unnecessary deletes of customers that were transferred between 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 just 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 with customers assigned to some other sales representative.

Upload

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

- ◆ **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 )
```

- ◆ **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.

- ◆ **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

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 holds 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

The **download_cursor** script 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.

The **download_delete_cursor** 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 therefor 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:

- ◆ **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 )
```

- ◆ **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.

- ◆ **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

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
```

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 the upload scripts is a conflict detection and resolution procedure.

If two sales representatives take orders and then synchronize, each one decrements the *quantity* column of the *Product* table. For example, if Sam Singer takes an order for 20 baseball hats (product ID 400), he will decrement the quantity from 90 to 70. If Pam Savarino takes an order for 10 baseball hats before receiving this change, she will decrement 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 script is a straightforward UPDATE at the consolidated database:

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

- ◆ **upload_fetch** The following 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' )
```

- ◆ **resolve_conflict script** 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*.

PART THREE

MobiLink Reference

This part contains reference material that describes in detail the various commands and components specific to MobiLink synchronization technology.

C H A P T E R 1 8

MobiLink Synchronization Server Options

About this chapter This chapter describes the options that can be set when starting the MobiLink synchronization server, dbmlsrv8.

| | | |
|----------|---------------------------------|-------------|
| Contents | Topic | Page |
| | MobiLink synchronization server | 380 |

MobiLink synchronization server

The MobiLink synchronization server lets you synchronize remote databases or applications with an ODBC-compliant consolidated database.

Function Start a MobiLink synchronization server.

Syntax **dbmlsrv8 -c "connection-string" [options]**

| Option | Description |
|-------------------------|---|
| -a | Disable automatic reconnection upon synchronization error. See "-a option" on page 383. |
| -bc size | Specify the amount of memory to reserve for blob caching. See "-bc option" on page 384. |
| -bn size | Specify the maximum number of bytes to consider when comparing blobs. See "-bn option" on page 384. |
| -c "keyword=value; ..." | Supply ODBC database connection parameters for your reference database. See "-c option" on page 384. |
| -cn connections | Set maximum number of connections with the consolidated database server. See "-cn option" on page 385. |
| -cr count | Set maximum number of database connection retries. See "-cr option" on page 385. |
| -ct connection-timeout | Set the length of time a connection may be unused before it is timed out. See "-ct option" on page 385. |
| -d number | Specify the size of the download cache. See "-d option" on page 386. |
| -dl | Display log messages on the console. See "-dl option" on page 386; |
| -e filename | Store remote error logs sent into a named file. See "-e option" on page 386. |
| -et filename | Truncate the file and append remote synchronization logs to the new file. See "-et option" on page 387. |
| -f | Assume synchronization scripts do not change. See "-f option" on page 387. |
| -o logfile | Log messages to a file. See "-o option" on page 387. |

| Option | Description |
|---|--|
| -oq | Prevent the popup dialog on startup error. See "-oq option" on page 388. |
| -os <i>size</i> | Maximum size of output file. See "-os option" on page 388. |
| -ot <i>logfile</i> | Log messages to a file, but truncate it first. See "-ot option" on page 389. |
| -ps <i>num</i> | Set maximum number of prepared statements to cache per connection. See "-ps option" on page 389. |
| -q | Minimize the synchronization server window. See "-q option" on page 389. |
| -r <i>retries</i> | Retry deadlocked uploads at most this many times. See "-r option" on page 389. |
| -rd <i>delay</i> | Set maximum delay, in seconds, before retrying a deadlocked transaction. See "-rd option" on page 390. |
| -s <i>count</i> | Specify the maximum number of rows to be fetched or sent at once. See "-s option" on page 390. |
| -sl dnnet <i>script-options</i> | Set the .NET CLR options and force loading of the virtual machine on startup. See "-sl dnnet option" on page 390. |
| -sl java <i>script-options</i> | Set the Java virtual machine options and force loading of the virtual machine on startup. See "-sl java option" on page 391. |
| -t <i>ODBC-output-file</i> | Log ODBC statements issued by MobiLink to this file. See "-t option" on page 392. |
| -tt <i>ODBC-output-file</i> | Log ODBC statements issued by MobiLink to this file, but first delete the file if it exists. See "-tt option" on page 393. |
| -u <i>size</i> | Specify the amount of memory to reserve for caching upload streams. See "-u option" on page 393. |
| -ud | On UNIX platforms, run as a daemon. See "-ud option" on page 393. |
| -v [<i>levels</i>] | Controls the type of messages written to the log file. See "-v option" on page 393. |
| -w <i>count</i> | Set the number of worker threads. See "-w option" on page 394. |
| -wu <i>count</i> | Set the maximum number of worker threads |

| Option | Description |
|--|--|
| | permitted to process uploads concurrently. See "-wu option" on page 395. |
| -x <i>protocol</i> [(<i>network-parameters</i>)] | Specify the communications protocol. Optionally, specify network parameters in form <i>parameter=value</i> , with multiple parameters separated by semicolons. See "-x option" on page 396. |
| -za | Allow generation of active scripts. See "-za option" on page 402. |
| -zac | Allow generation of active cursored scripts. See "-zac option" on page 403. |
| -zd | Pass the <i>last_download</i> timestamp last. See "-zd option" on page 403. |
| -ze | Allow generation of sample scripts. See "-ze option" on page 403. |
| -zec | Allow generation of example cursor scripts. See "-zec option" on page 404. |
| -zp | In the event of a timestamp conflict between the consolidated and remote database, this option allows timestamp values with a precision higher than the lowest-precision to be used for conflict detection purposes. See "-zt option" on page 405. |
| -zs <i>name</i> | Identifying name for dbmlstop . See "-zs option" on page 405. |
| -zt <i>number</i> | Set number of operating system threads to run concurrently. See "-zt option" on page 405. |
| -zu { + - } | Allow automatic addition of users when the <i>authenticate_user</i> script is undefined. See "-zu option" on page 405. |
| -zw <i>1,...5</i> | Controls which levels of warning message to display. See "-zw option" on page 405. |
| -zwd <i>code</i> | Disables specific warning codes. See "-zwd option" on page 406. |
| -zwe <i>code</i> | Enables specific warning codes. See "-zwe option" on page 407. |

Description


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

The MobiLink synchronization server is compatible with a variety of database-management systems, including Adaptive Server Anywhere, Adaptive Server Enterprise, Oracle, Microsoft SQL Server, and IBM DB2.

You must supply connection parameters for the consolidated database using the `-c` option. The command line options may be presented in any order. The `-c` option is shown here as the first item in a command string as a convention only. It can be anywhere in a list of options, but must precede a connection string.

The MobiLink server requires the consolidated database to be running. Unless your ODBC data source is configured to automatically start the consolidated database, the database must be running before you start the MobiLink server.

You can put `dbmlsrv8` command line options in a configuration file and optionally use the File Hiding utility, `dbfhide`, to add simple encryption to the configuration file.

 For more information, see "Using configuration files" on page 10 of the book *ASA Database Administration Guide* and "The File Hiding utility" on page 446 of the book *ASA Database Administration Guide*.

dbmlsrv8 options

This section lists all MobiLink synchronization server command line options.

-a option

Function

Instructs the MobiLink synchronization server not to reconnect on synchronization error.

Syntax

dbmlsrv8 -c "connection-string" -a ...

Description

Should an error occur during synchronization, the MobiLink synchronization server automatically disconnects from the consolidated database, and then re-establishes the connection. Reconnecting ensures that the following synchronization starts from a known state. When this behavior is not required, you can use this option to disable it. The maintenance of state information depends on programmer requirements and may vary depending on the ways in which the programmer configures MobiLink scripting to work with the DBMS. This applies even if that database is an Oracle, Adaptive Server Anywhere database or other supported product. Some status information may need to be re-initialized depending on the client application.

-bc option

| | |
|--------------------|--|
| Function | Sets the blob cache size. |
| Syntax | dbmlsrv8 -c "connection-string" -bc size ... |
| Description | <p>The amount of memory to use for caching blobs. If more memory is required, the MobiLink synchronization server uses disk space, instead. For this reason, too small a value can degrade performance. To calculate the minimum recommended size, multiply the maximum size of all blob data in any one row by the number of worker threads, then multiply the result by 4, which provides for a large reserve of memory.</p> <p>The <i>size</i> is the amount of memory to reserve in bytes. Use the suffix "k", "m" or "g" to specify units of kilobytes, megabytes or gigabytes, respectively.</p> |

-bn option

| | |
|--------------------|--|
| Function | Sets the maximum number of blob bytes to compare. |
| Syntax | dbmlsrv8 -c "connection-string" -bn size... |
| Description | <p>When two blobs contain similar or identical values, the operation of comparing them for filtering or conflict detection can be expensive due to the amount of data involved. This option tells the MobiLink synchronization server to consider only the first <i>size</i> bytes of two blobs when making the comparison. The default is to consider the entire blobs, no matter how big they are.</p> <p>Under some situations, limiting the maximum amount of data compared can speed synchronization substantially; however, it can also cause errors. For example, if two large blobs differ only in the last few bytes, the MobiLink synchronization server may decide that they are identical when in fact they are not.</p> |

-c option

| | |
|--------------------|--|
| Function | Specifies connection parameters for the consolidated database. |
| Syntax | dbmlsrv8 -c "connection-string" ... |
| Description | The connection string must give the MobiLink synchronization server information sufficient to connect to the consolidated database. The connection string is required. |

The connection string must specify connection parameters in the form *keyword=value*, separated by semicolons, with no spaces between parameters.

Connection parameters must be included in the ODBC data source specification if not given in the command line. Check your RDBMS and ODBC data source to determine required connection data.

✍ For a complete list of SQL Anywhere connection parameters, see "Connection parameters" on page 164 of the book *ASA Database Administration Guide*.

-cn option

Function

Sets the maximum number of consolidated database connections.

Syntax

dbmlsrv8 -c "connection-string" -cn value...

Description

Specifies the maximum number of connections that the MobiLink synchronization server should make to the consolidated database. The minimum and the default value are one greater than the number of worker threads. A warning is issued if the supplied value is too small.

A value larger than the number of worker threads may speed performance, particularly if connecting to the consolidated database is slow or if multiple script versions are in use. The optimum maximum number of database connections is the number of script versions times the number of worker threads. Connections above this optimum value will not necessarily speed synchronization, and will needlessly consume resources in both the MobiLink synchronization server and the consolidated database server.

See also

"MobiLink database connections" on page 227

-cr option

Function

Sets the maximum number of database connection retries.

Syntax

dbmlsrv8 -c "connection-string" -cr value...

Description

Set the maximum number of times that the MobiLink synchronization server will attempt to connect to the database, before quitting, when a connection goes bad. The default value is three connection retries.

-ct option

Function

Sets the length of time a connection may be unused before it is timed out and disconnected by the MobiLink synchronization server.

| | |
|--------------------|---|
| Syntax | dbmlsrv8 -c "connection-string" -ct connection-timeout... |
| Description | MobiLink database connections that go unused for a specified amount of time are freed by the server. The timeout can be set using the -ct option. A default timeout period of 60 minutes is used. |
| -d option | |
| Function | Sets the size of the download cache. |
| Syntax | dbmlsrv8 -c "connection-string" -d number... |
| Description | When no download acknowledgement is required, MobiLink buffers the download stream in a download cache. Since no acknowledgement is required from the client to commit the download transaction, the buffered download stream is sent to the client after the commit. The size of the download cache can be specified with the d command line option in units of kilobytes (k), megabytes (m), or gigabytes (g). The default size for the download cache is 0.5 megabytes. |
| -dl option | |
| Function | Displays all log messages on screen. |
| Syntax | dbmlsrv8 -c "connection-string" -v -dl ... |
| Description | Display messages in the MobiLink synchronization server window if specified by the -v command line option. By default, only a subset of all messages is shown in the window when a log file is being output (using -o). In circumstances with many messages, this option can degrade performance. |
| -e option | |
| Function | Stores error logs from Adaptive Server Anywhere MobiLink clients in a named file. |
| Syntax | dbmlsrv8 -c "connection-string" -e filename... |
| Description | With no -e option, error logs from Adaptive Server Anywhere MobiLink clients are stored in a file named <i>dbmlsrv.mle</i> . The -e option instructs the MobiLink synchronization server to store the error logs in a named file. By default, <i>dbmlsync</i> will send, on the occurrence of an error on the remote site, up to 32 kilobytes of remote log messages to a MobiLink synchronization server. |
| See also | "-et option" on page 387 |

-et option

| | |
|--------------------|---|
| Function | Stores error logs from Adaptive Server Anywhere MobiLink clients in a named file after truncating the existing file. |
| Syntax | dbmlsrv8 -c "connection-string" -et filename... |
| Description | <p>The amount of information delivered from a remote site can be controlled by a dbmlsync extended option, ErrorLogSendLimit. The valid setting for this extended option is <i>n</i>, <i>nk</i>, <i>nK</i>, <i>nm</i>, or <i>nM</i>, where <i>n</i> is zero or a positive integer and the units are bytes. Here, <i>n</i> means <i>n</i> bytes, <i>nk</i> or <i>nK</i> means <i>n</i> kilobytes and <i>nm</i> or <i>nM</i> means <i>n</i> megabytes. If you don't want to send any dbmlsync output log messages, a zero value should be set for this extended option.</p> <p>If ErrorLogSendLimit is set to be large enough, dbmlsync sends the entire output log messages from the current session up to MobiLink synchronization server. For instance, if the output log messages were appended to an old output log file, dbmlsync will only send the new messages generated in the current session to MobiLink synchronization server. If the total length of new messages is greater than ErrorLogSendLimit, dbmlsync will only log the last part of the newly generated error and log messages up to the specified size.</p> <p>The -et option is the same as the -e option, except that the error log file is truncated before any new errors are added to it.</p> |
| See also | "-e option" on page 386 |

-f option

| | |
|--------------------|---|
| Function | Loads synchronization scripts only on startup, for better performance. |
| Syntax | dbmlsrv8 -c "connection-string" -f... |
| Description | Without the -f option, the MobiLink synchronization server checks to see if synchronization scripts have changed during regular operation. This checking is helpful during development, but can have an unnecessary performance impact in a production environment. With the -f option, the MobiLink synchronization server loads the synchronization scripts at startup time only. |

-o option

| | |
|-----------------|--|
| Function | Logs output messages to a message log file. |
| Syntax | dbmlsrv8 -c "connection-string" -o logfile... |

Description

Write all log messages to the specified file. Note that the MobiLink synchronization server window, if present, usually shows a subset of all messages logged.

The MobiLink synchronization server gives the full error context in its output file if errors occur during synchronization. The error context may include the following information:

- ◆ **User Name** This is the actual user name that is provided by MobiLink Adaptive Server Anywhere applications during synchronization.
- ◆ **Modified User Name** This is the user name modified by the `modify_user` script.
- ◆ **Transaction** This lists the transaction the error occurs. The transaction could be `authenticate_user`, `begin _synchronization`, `upload`, `prepare_for_download`, `download`, or `end_synchronization`.
- ◆ **Table Name** This shows the table name if it is available or NULL.
- ◆ **Row** The operation could be INSERT, UPDATE, DELETE or FETCH.
- ◆ **Row Data** This shows all the column values of the row that caused the error.
- ◆ **Script Version** This is the script version currently used for synchronization.
- ◆ **Script** This is the actual script used for the synchronization.

You can see contextual information in the log regardless of your chosen level of verbosity.

-oq option

Function

Prevents the appearance of the dialog when a startup error occurs.

Syntax

dbmlsrv8 -c "connection-string" -oq ...

Description

By default, the MobiLink synchronization server displays a message box dialog if a startup error occurs. The `-oq` option prevents this dialog from being displayed.

-os option

Function

Sets the maximum size of the message log file.

Syntax

dbmlsrv8 -c "connection-string" -os size...

Description

The *size* is the maximum file size for logging output messages, specified in units of bytes. Use the suffix "k" or "m" to specify units of kilobytes or megabytes, respectively. By default, there is no size limit. The minimum size limit is 10 kb.

Before the MobiLink synchronization server logs output messages to a file, it checks the current file size. If the log message will make the file size exceed the specified size, the MobiLink synchronization server renames the message log file to *yymmddxx.mls*. In this instance, *xx* are sequential characters ranging from AA to ZZ, and *yymmdd* represents the final two digits of the current year, the month, and the day of the month.

You can use this option to prune old message log files to free up disk space. The latest output is always appended to the file specified by *-o* or *-ot*.

-ot option

Function

Logs the output message to the message log file, but truncate it first.

Syntax

dbmlsrv8 -c "connection-string" -ot logfilename ...

Description

Truncate the message log file and then append output messages to it. The default is to send output to the screen.

-ps option

Function

Sets the maximum number of prepared statements to cache per connection.

Syntax

dbmlsrv8 -c "connection-string" -ps num ...

Description

Controls the maximum number of ODBC prepared statements kept in the prepared statement cache.

-q option

Function

Instructs MobiLink to run in a minimized window.

Syntax

dbmlsrv8 -c "connection-string" -q ...

Description

Minimize the MobiLink synchronization server window.

-r option

Function

Sets the maximum number of deadlock retries.

Syntax

dbmlsrv8 -c "connection-string" -r retries ...

Description

By default, MobiLink synchronization server retries uploads that are deadlocked for a maximum of 10 attempts. If the deadlock is not broken, synchronization fails, since there is no guarantee that the deadlock can be overcome. This option allows an arbitrary retry limit to be set. To stop the server from retrying deadlocked transactions, specify **-r 0**. The upper bound on this setting is 2 to the power 32, minus one.

-rd option

Function

Sets the maximum delay time between deadlock retries.

Syntax

dbmlsrv8 -c "connection-string" -rd delay ...

Description

When upload transactions are deadlocked, the MobiLink synchronization server waits a random length of time before retrying each transaction. The random nature of the delay greatly increases the likelihood that future attempts will succeed. This option allows you to specify the maximum delay in units of seconds. The value 0 (zero) makes retries instantaneous, but larger values are recommended because they yield more successful retries. The default and maximum delay value is 30.

-s option

Function

Sets the number of rows fetched, inserted, updated, or deleted at once.

Syntax

dbmlsrv8 -c "connection-string" -s count ...

Description

Set the number of rows fetched or transferred at one time from the database to *count*. Set this option to no less than the number of rows specified in the ODBC prefetch option, if this option is set. The default value is 10. The number of rows fetched at once can be viewed in the log file as **rowset size**.

-sl dnet option

Function

Sets the .NET Common Language Runtime (CLR) options and forces the CLR to load on startup.

Syntax

dbmlsrv8 -c "connection-string" -sl dnet options ...

Description

Sets options to pass directly to the .NET CLR. The available options are:

| Option | Description |
|-----------------------------------|--|
| -Dname=value | Set an environment variable. For example, <div>-Dsychtype=far -Dextra_rows=yes</div> For more information, see the .NET framework class System.Environment. |
| -MLAutoLoadPath=path | Set the location of base assemblies. Only works with private assemblies. To tell MobiLink where assemblies are located, use this option or -MLDomConfigFile, but not both. When you use this option, you cannot specify a domain in the event script. The default is the current directory. |
| -MLDomConfigFile=file | Set the location of base assemblies. Use when you have shared assemblies, or you don't want to load all assemblies in the directory, or you can't use MLAutoLoadPath for some other reason. To tell MobiLink where assemblies are located, use this option or -MLAutoLoadPath, but not both. |
| -MLStartClasses=classnames | At server startup, load user-defined start classes in the order listed. |
| -clrConGC | Enable concurrent garbage collection in the CLR. |
| -clrFlavor=(wks svr) | Flavor of the .NET CLR to load. The flavor is svr for server and wks for workstation. By default, wks is loaded. |
| -clrVersion=version | Version of the .NET CLR to load. This must be prefixed with v . For example, v1.0.3705 loads the directory WINNT\Microsoft.NET\Framework\v1.0.3705. |

To display this list of options, use the following command:

```
dbmlsrv8 -sl dnet (?)
```

See also

"Loading assemblies" on page 191
"Writing Synchronization Scripts in .NET" on page 187

-sl java option

Function

Sets the Java virtual machine options and forces the virtual machine to load on startup.

Syntax

```
dbmlsrv8 -c "connection-string" -sl java options ...
```

| | | |
|--|--|--|
| Description | | Sets <code>-jrepath</code> and sets other options to pass directly to the Java virtual machine. The available options are: |
| Option | Description | |
| <code>(-hotspot -server -classic)</code> | Override the default choice for the Java VM to use. | |
| <code>{ -cp -classpath } location;...</code> | Specify a set of directories or jar files in which to search for classes. | |
| <code>-Dname=value</code> | Set a system property. For example, <code>-Dsynctype=far -Dextra_rows=yes</code> | |
| <code>-DMLStartClasses=class, ...</code> | At server startup, load user-defined start classes in the order listed. | |
| <code>-jrepath path</code> | Override the default JRE path, which is <code>sun\jre131</code> directory under the <code>Sybase\shared</code> directory. | |
| <code>-verbose:(class gc jni)</code> | Enable verbose output. | |
| <code>-X vm-option</code> | Set a VM-specific option as described in the file <code>Xusage.txt</code> , which by default is installed to <code>Sybase\Shared\Sun\jre131\bin\hotspot</code> . | |

To display this list of options, use the following command:

```
dbmlsrv8 -sl java (?)
```

See also "Writing Synchronization Scripts in Java" on page 165
"User-defined start classes" on page 174

-t option

Function Creates a file containing all the ODBC statements issued by MobiLink.

Syntax **dbmlsrv8 -c "connection-string" -t ODBC-output-file ...**

Description This option can be used to create a file containing all of the ODBC statements issued by MobiLink. If used on UNIX, with the Adaptive Server Anywhere driver used as a driver manager, this feature is ignored. The feature is useful for tracing what was called, passed, and retrieved. It has a severe impact on performance.

To prevent the file from becoming large, use the "-tt option" on page 393.

See also "-tt option" on page 393

-tt option

| | |
|--------------------|---|
| Function | Logs ODBC statements issued by MobiLink to a file. If the file already exists, it first deletes it. |
| Syntax | dbmlsrv8 -c "connection-string" -tt ODBC-output-file ... |
| Description | This option is used to create a file containing all of the ODBC statements issued by MobiLink. If used on UNIX with the Adaptive Server Anywhere driver used as a driver manager, this feature is ignored. The feature is useful for tracing what was called, passed, and retrieved. It has a severe impact on performance. |
| See also | "-t option" on page 392 |

-u option

| | |
|--------------------|--|
| Function | Sets the upload cache size. |
| Syntax | dbmlsrv8 -c "connection-string" -u size ... |
| Description | The amount of space, in units of bytes, to reserve for caching upload streams that are being processed. Use the suffix k, m, or g to specify units of kilobytes, megabytes, or gigabytes respectively. You should consider enlarging this value if your clients upload large streams, or many clients synchronize at once, or both. The suggested size is the maximum expected size of an upload stream multiplied by the number of worker threads. The default value is 500 Kb. |

-ud option

| | |
|--------------------|--|
| Function | Instructs MobiLink to run as a daemon. |
| Syntax | dbmlsrv8 -c "connection-string" -ud ... |
| Description | UNIX platforms only. |
| See also | "Running MobiLink Outside the Current Session" on page 275 |

-v option

| | |
|--------------------|---|
| Function | Allows you to specify what information is logged to the message log file and displayed in the synchronization window. |
| Syntax | dbmlsrv8 -c "connection-string" -v[levels] ... |
| Description | This option controls the type of messages written to the message log file. |

If you specify `-v` alone, the MobiLink synchronization server writes a minimal amount of information about each synchronization.

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

- ◆ **+** Turn on all logging options that increase verbosity. Negative verbosity options are not turned on by this option.
- ◆ **c** Show the content of each synchronization script when it is invoked. This level implies **s**.
- ◆ **h** Show the remote schema as uploaded during synchronization.
- ◆ **n** Show row-count summaries.
- ◆ **r** Display the column values of each row uploaded or downloaded.
- ◆ **s** Show the name of each synchronization script as it is invoked.
- ◆ **t** Show the translated SQL that results from scripts that are written in ODBC canonical format. This level implies **c**. The following example shows the automatic translation of a statement for Adaptive Server Anywhere.

```
I. 02/11 11:02:14. [102]: begin_upload synch2
    { call SynchLogLine( ?, ?, 'begin_upload' ) }

I. 02/11 11:02:14. [102]: Translated SQL:
    call SynchLogLine( ?, ?, 'begin_upload' )
```

The following example shows the translation of the same statement for Microsoft SQL Server.

```
I. 02/11 11:03:21. [102]: begin_upload synch2
    { call SynchLogLine( ?, ?, 'begin_upload' ) }

I. 02/11 11:03:21. [102]: Translated SQL:
    EXEC SynchLogLine ?, ?, 'begin_upload'
```

- ◆ **u** Show undefined table scripts. This may help new users understand the synchronization process.

-w option

Function

Sets the number of worker threads.

Syntax

dbmlsrv8 -c "connection-string" -w count ...

Description

Each worker thread accepts synchronization requests one at a time. Each worker thread is associated with a network protocol. If you have more than one protocol defined, the worker threads are divided evenly among the protocols.

Each worker thread uses one connection to the consolidated database. The MobiLink synchronization server opens one additional connection for administrative purposes. Hence, the minimum number of connections from the MobiLink synchronization server to the consolidated database is *count* + 1.

The number of worker threads has a strong influence on MobiLink synchronization throughput, and you need to run tests to determine the optimum number for your particular synchronization setup. The number of worker threads determines how many synchronizations can be active simultaneously; the rest will be queued waiting for worker threads to become available. Thus adding worker threads should increase throughput, but it will also increase the possibility of contention between the active synchronizations. At some point adding more worker threads will decrease throughput, because the increased contention outweighs the benefit of overlapping synchronizations.

☞ For more information, see the MobiLink Performance whitepaper at <http://my.sybase.com/detail?id=1009664>, and "MobiLink Performance" on page 219.

The value set for this option is also the default setting for the `-wu` option, which can be used to limit the number of threads that can simultaneously upload. This is useful if the optimum number of worker threads for downloading is larger than the optimum number for uploading, as is typically the case with remote databases on slow computers or with slow connections to the MobiLink server. Tests have shown that for slow synchronization clients (such as Palm devices or computers connected by dialup), the best throughput is achieved with a large number of worker threads (via `-w`) with a small number allowed to apply uploads simultaneously (via `-wu`). In general, the optimum number for `-wu` depends on the consolidated database, and is relatively independent of the processing or network speeds for the remote databases. Therefore, when you increase the number of threads with `-w`, you may want to use `-wu` to restrict the number that can upload simultaneously. For more information, see "`-wu` option" on page 395.

The default number of worker threads is 5.

-wu option

Function

Sets the maximum number of worker threads that can apply uploads simultaneously.

Syntax

dbmlsrv8 -c "connection-string" -wu count ...

Description

Use the `-wu` option to limit the number of worker threads that can simultaneously apply uploads. When the limit is reached, a worker thread that is ready to apply its upload must wait until another finishes its upload. The excess worker threads are still free to receive uploads or to download.

The most common cause of contention in the consolidated database is having too many worker threads applying uploads simultaneously. This can be an issue when the network connection is slow, or when the client device has low processing speed. For example, when working over a wide-area wireless network or using a Palm device you may want to increase the total number of threads (`-w`) but limit the number that can apply uploads simultaneously.

Consider the following example. In a pilot setup using a LAN and remote databases on PCs, you find that the optimum number of worker threads is approximately 10 for both upload-only and download-only synchronizations, and that corresponds to 100% CPU utilization on the consolidated database. With fewer worker threads you find that throughput is less and the CPU utilization for the consolidated database is lower. With more worker threads, throughput does not increase because the consolidated database is already processing as fast as it can with 10 workers. When you switch to using a dialup network with 10 MobiLink worker threads, you will probably find that throughput is lower and the consolidated CPU utilization has dropped. You may find that you can get throughput (and consolidated CPU utilization) to approach the values obtained with the LAN by increasing the number of worker threads (via `-w`) while keeping the number that apply uploads simultaneously limited to 10 (via `-wu`).

By default, all worker threads can apply uploads simultaneously. The number of worker threads that are used is set by the `-w` option. The default is 5.

-x option

Function

Sets communications protocol and parameters for MobiLink clients. These are used by the MobiLink synchronization server to listen for synchronization requests.

Syntax

dbmlsrv8 -c "connection-string" -x protocol [(network-parameters;...)]...

Description

Specify communications protocol through which to communicate with client applications.

Note for UltraLite users

If you are using an UltraLite Java application *and* you are using TLS security, the syntax of `-x` is slightly different. For details, see "Using transport-layer security from UltraLite Java applications" on page 353 of the book *UltraLite User's Guide*.

The allowed values of *protocol* are as follows:

- ◆ **tcipip** Accept connections from applications via TCP/IP.
- ◆ **http** Accept connections via the standard Web protocol. Client applications can pick their HTTP version and the MobiLink synchronization server adjusts on a per-connection basis.
- ◆ **https** Accept connections via a variant of HTTP that handles secure transactions. The HTTPS stream implements HTTP over SSL/TLS using RSA encryption, and is compatible with any other HTTPS server.

Optionally, you can also specify network parameters, in the form *parameter=value*. Separate multiple parameters with semicolons. Which parameters you specify depends on the protocol you choose.

- ◆ **TCP/IP parameters** If you specify the **tcipip** protocol, you can optionally specify the following *network-parameters*:
 - ◆ **client_port=nnnnn or client_port=nnnnn-mmmmm** A range of client ports for communication. If only one value is specified, the end of the range is 100 greater than the initial value, for a total of 101 ports.

The option can be useful for clients inside a firewall communicating with a MobiLink synchronization server outside the firewall.
 - ◆ **host=hostname** The host name or IP number on which the MobiLink synchronization server should listen. The default value is **localhost**.
 - ◆ **port=portnumber** The socket port number on which the MobiLink synchronization server should listen. The port number must be a decimal number that matches the port the MobiLink synchronization server is setup to monitor. The default port is 2439, which is the IANA registered port number for the MobiLink synchronization server.

- ◆ **keep_alive=[0|1]** In some circumstances, MobiLink worker threads become unavailable when TCP/IP-based connections disappear during synchronization. These blocked worker threads are waiting for replies from the MobiLink client. If all worker threads reach this state, MobiLink cannot process synchronizations. Similarly, MobiLink clients can become blocked if the connection disappears. The TCP/IP-based streams that are used during MobiLink synchronization accept a parameter, both on the client and server side, to manage liveness. The default is 1 (On).
- ◆ **security=cipher(keyword=value;...)** All communication through this connection is to be encrypted using the cipher suite specified. The cipher can be one of **ecc_tls** or **rsa_tls**. These refer to elliptic-curve and RSA certification. For backwards compatibility, **ecc_tls** can also be specified as **certicom_tls**.

You can optionally specify the security keywords **certificate** (the certificate that is to be used for server authentication), and **certificate_password**. You must use a certificate that matches the cipher suite you choose. If you do not specify a certificate, MobiLink uses a sample certificate appropriate to the cipher suite you chose.

Your installation includes a sample elliptic-curve certificate called `sample.crt` with password `tJ1#m6+W`, and a sample RSA certificate called `rsaserver.crt` with password `test`. The sample certificates are for testing and development only. They provide no security because the same certificates and passwords are distributed to all Adaptive Server Anywhere customers. New certificates are available from several companies, including Entrust Technologies and VeriSign.

Separately licensable option required

Use of Certicom technology requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ For more information about security, see "Transport-Layer Security" on page 283.

- ◆ **HTTP parameters** If you specify the **http** protocol, you can optionally specify the following *network-parameters*:
 - ◆ **client_port=nnnnn or client_port=nnnnn-mmmmm** A range of client ports for communication. If only one value is specified, the end of the range is 100 greater than the initial value, for a total of 101 ports.

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

- ◆ **contd_timeout=seconds** The number of seconds the MobiLink synchronization server waits to receive the next part of a partially completed synchronization before the synchronization is abandoned. You can tune this option to free MobiLink worker threads when the wait time indicates that the client will never continue the connection. The default value is 30 seconds.
- ◆ **host=hostname** The host name or IP number on which the MobiLink synchronization server should listen. The default value is **localhost**.
- ◆ **port=portnumber** The socket port number on which the MobiLink synchronization server should listen. The port number must be a decimal number that matches the port the MobiLink synchronization server is setup to monitor. The default port is 80.
- ◆ **keep_alive=[0|1]** In some circumstances, MobiLink worker threads become unavailable when TCP/IP-based connections disappear during synchronization. These blocked worker threads are waiting for replies from the MobiLink client. If all worker threads reach this state, MobiLink cannot process synchronizations. Similarly, MobiLink clients can become blocked if the connection disappears. The TCP/IP-based streams that are used during MobiLink synchronization accept a parameter, both on the client and server side, to manage liveness. The default is 1 (On).
- ◆ **security=cipher(keyword=value;...)** All communication through this connection is to be encrypted using the cipher suite specified. The cipher can be one of **ecc_tls** or **rsa_tls**. These refer to elliptic-curve and RSA certification. For backwards compatibility, **ecc_tls** can also be specified as **certicom_tls**.

You can optionally specify the security keywords. **certificate** (the certificate that is to be used for server authentication), and **certificate_password**. You must use a certificate that matches the cipher suite you choose. If you do not specify a certificate, MobiLink uses a sample certificate appropriate to the cipher suite you chose.

Your installation includes a sample elliptic-curve certificate called **sample.crt** with password **tJ1#m6+W**, and a sample RSA certificate called **rsaserver.crt** with password **test**. The sample certificates are for testing and development only. They provide no security because the same certificates and passwords are distributed to all Adaptive Server Anywhere customers. New certificates are available from several companies, including Entrust Technologies and VeriSign.

Separately licensable option required

Use of Certicom technology requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

🔒 For more information about security, see "Transport-Layer Security" on page 283.

- ◆ **unknown_timeout=seconds** The number of seconds the MobiLink synchronization server waits to receive HTTP headers on a new connection before the synchronization is abandoned. You can tune this option to free MobiLink worker threads when the wait time indicates that a network failure has occurred. The default value is 30 seconds.
- ◆ **url_suffix=suffix** The suffix to add to the URL on the first line of each HTTP request. This parameter can be used to help ensure that a particular client connects to the intended server.
- ◆ **version=http-version** The MobiLink synchronization server automatically detects the HTTP version used by a client. This parameter is a string specifying the default version of HTTP to use in case the server cannot detect the method used by the client. You have a choice of **1.0** or **1.1**. The default value is **1.1**.
- ◆ **HTTPS parameters** The HTTPS communication stream uses Certicom RSA security.

Separately licensable option required

Use of Certicom technology requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

🔒 For more information about security, see "Transport-Layer Security" on page 283.

If you specify the **https** protocol, you can optionally specify the following *network-parameters*:


- ◆ **client_port=nnnnn or client_port=nnnnn-mmmmm** A range of client ports for communication. If only one value is specified, the end of the range is 100 greater than the initial value, for a total of 101 ports.

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

- ◆ **contd_timeout=seconds** The number of seconds the MobiLink synchronization server waits to receive the next part of a partially completed synchronization before the synchronization is abandoned. You can tune this option to free MobiLink worker threads when the wait time indicates that the client will never continue the connection. The default value is 30 seconds.
- ◆ **host=hostname** The host name or IP number on which the MobiLink synchronization server should listen. The default value is **localhost**.
- ◆ **keep_alive=[0|1]** In some circumstances, MobiLink worker threads become unavailable when TCP/IP-based connections disappear during synchronization. These blocked worker threads are waiting for replies from the MobiLink client. If all worker threads reach this state, MobiLink cannot process synchronizations. Similarly, MobiLink clients can become blocked if the connection disappears. The TCP/IP-based streams that are used during MobiLink synchronization accept a parameter, both on the client and server side, to manage liveness. The default is 1 (On).
- ◆ **port=portnumber** The socket port number on which the MobiLink synchronization server should listen. The port number must be a decimal number that matches the port the MobiLink synchronization server is setup to monitor. The default port is 443.
- ◆ **certificate** An optional parameter that specifies a certificate name. This must be an RSA certificate. If you do not specify a certificate, MobiLink uses the sample RSA certificate that is provided with Adaptive Server Anywhere. It is called `rsaserver.crt` and has the password test.

The sample certificates are for testing and development only. They provide no security because the same certificates and passwords are distributed to all Adaptive Server Anywhere customers. New certificates are available from several companies, including Entrust Technologies and VeriSign.

- ◆ **certificate_password** An optional parameter that specifies a password for the certificate.

 For more information about security, see "Transport-Layer Security" on page 283.

- ◆ **unknown_timeout=seconds** The number of seconds the MobiLink synchronization server waits to receive HTTP headers on a new connection before the synchronization is abandoned. You can tune this option to free MobiLink worker threads when the wait time indicates that a network failure has occurred. The default value is 30 seconds.

- ◆ **url_suffix=suffix** The suffix to add to the URL on the first line of each HTTP request. This parameter can be used to help ensure that a particular client connects to the intended server.
- ◆ **version=http-version** The MobiLink synchronization server automatically detects the HTTP version used by a client. This parameter is a string specifying the default version of HTTP to use in case the server cannot detect the method used by the client. You have a choice of **1.0** or **1.1**. The default value is **1.1**.

-za option

Function

Allows the generation of active statement-based scripts.

Syntax

dbmlsrv8 -c "connection-string" -za

Description

Generate statement-based scripts that perform a simple snapshot synchronization.

To generate scripts, you must also specify that the client sends column names. You do this when you initiate synchronization. For Adaptive Server Anywhere remotes, see "SendColumnNames" on page 420. For UltraLite remotes, see "send_column_names synchronization parameter" on page 389 of the book *UltraLite User's Guide*.

The scripts that are generated are called `upload_insert`, `upload_update`, `upload_delete`, and `download_cursor`. The generated scripts are used for the current synchronization, as well as for following those using the same script version.

The scripts are generated when `dbmlsync` synchronizes and apply to the script version specified on the `dbmlsync` command line, with a default value of default. The generated scripts perform one-to-one snapshot synchronization using the table and column names sent from the client. If the consolidated database has different table or column names, activating these scripts will cause an error during the synchronization.

Note:

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.

See also

"-zec option" on page 404

Example

The following `dbmlsrv8` command enables automatic script generation. The `dbmlsync` command sets the necessary `SendColumnNames` option.


```
dbmlsrv8 -c "dsn=YourDBDSN" -za
```

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

-zac option

Function

Allows the generation of active cursor-based scripts.

Syntax

dbmlsrv8 -c "connection-string" -zac

Description

Generate cursor-based scripts that perform a simple snapshot synchronization.

To generate scripts, you must also specify that the client sends column names. You do this when you initiate synchronization. For Adaptive Server Anywhere remotes, see "SendColumnNames" on page 420. For UltraLite remotes, see "send_column_names synchronization parameter" on page 389 of the book *UltraLite User's Guide*.

The scripts are named `upload_cursor` and `download_cursor`. The generated scripts are used for the current synchronization, as well as for following synchronizations using the same script version.

See also

"-za option" on page 402

-zd option

Function

Allows MobiLink to pass the read-only last download timestamp last.

Syntax

dbmlsrv8 -c "connection-string" -zd

Description

Pass the `last_download` timestamp as an argument last. If `-zd` is not used, the `last_download` timestamp is sent first.

-ze option

Function

Allows the generation of sample statement-based scripts.

Syntax

dbmlsrv8 -c "connection-string" -ze

Description

Generate sample scripts that, if activated, perform a simple snapshot synchronization.

To generate scripts, you must also specify the `SendColumnNames` extended option on the `dbmlsync` command line. For more information, see "SendColumnNames" on page 420.

The generated scripts are named example_upload_insert, example_upload_update, example_upload_delete, and example_download_cursor. The synchronization is canceled once the scripts are generated.

Note:

Scripts are generated only the first time that a remote synchronizes, and only when the given script version does not exist. Otherwise, -ze has no effect.

See also "-za option" on page 402

-zec option

Function

Allows the generation of sample cursor-based scripts.

Syntax

dbmlsrv8 -c "connection-string" -zec

Description

Generate sample cursor-based scripts that, if activated, perform a simple snapshot synchronization. You must also specify the SendColumnNames extended option on the dbmlsync command line to generate the scripts.

The generated scripts are named example_upload_cursor and example_download_cursor.

See also "-za option" on page 402

-zp option

Function

Adjusts which timestamp values will be used for conflict detection purposes.

Syntax

dbmlsrv8 -c "connection-string" -zp

Description

In the event of a timestamp conflict between the consolidated and remote database, this option allows timestamp values with a precision higher than the lowest precision to be used for conflict detection purposes. The option is useful when the consolidated database is more precise than the remote, as updated timestamps on the remote can cause conflicts in the next synchronization. The option allows MobiLink to ignore these conflicts. When there is a precision mismatch and zp is not used, a per synchronization and a schema sensitive per table warning are written to the log to advertise the -zp option. Another per synchronization warning is also added to advise users to adjust the timestamp precision on the remote database where possible.

See also "MobiLink stop utility" on page 613

-zs option

| | |
|--------------------|--|
| Function | Specifies a MobiLink server name. |
| Syntax | dbmlsrv8 -c "connection-string" -zs name |
| Description | Specify a server name for the MobiLink synchronization server. If the MobiLink synchronization server is started using the -zs option, it must be shut down using the dbmlstop server-name command. Shutdown may only occur from the same machine as the server. |
| See also | "MobiLink stop utility" on page 613 |

-zt option

| | |
|--------------------|--|
| Function | Allows the given number of operating system threads that you specify to run concurrently. |
| Syntax | dbmlsrv8 -c "connection-string" -zt number |
| Description | Run a number of operating system threads concurrently. This option may be required for some ODBC drivers. It also gives you fine control of processor resources. |

-zu option

| | |
|--------------------|---|
| Function | Allows the automatic addition of users when the authenticate_user script is undefined. |
| Syntax | dbmlsrv8 -c "connection-string" -zu{ + - } ... |
| Description | If this is supplied as -zu+, then unrecognized MobiLink user names are added to the ml_user table on first synchronizing. If the argument is supplied as -zu-, or not supplied, unrecognized user names are prevented from synchronizing. Adaptive Server Anywhere versions 7.0.x and previous automatically added new users. |

-zw option

| | |
|--------------------|--|
| Function | Controls which levels of warning message to display. |
| Syntax | dbmlsrv8 -c "connection-string" -zw levels |
| Description | MobiLink has five levels of warning messages: |

| Level | Description |
|-------|--|
| 0 | Suppress all warning messages |
| 1 | Server and high ODBC level: warning messages when the MobiLink synchronization server starts |
| 2 | Synchronization and user level: warning messages when a synchronization starts |
| 3 | Schema level: warning messages when a MobiLink synchronization server is processing a client schema |
| 4 | Script and lower ODBC level: warning messages when a MobiLink synchronization server fetches, prepares, or executes scripts |
| 5 | Table or row level: warning messages when a MobiLink synchronization server performs table operations in an upload or download |

To specify the level of warning messages you want reported, you can separate levels with a comma, or separate a range with two dots. For example, **-zw 1..3,5** is the same as **-zw 1,2,3,5**.

The reporting of messages has a slight impact on performance. Levels with a higher number tend to produce more messages.

If **-zw** is used more than once in the same command line, MobiLink recognizes only the last instance. If settings of **-zw**, **-zwd**, and **-zwe** conflict, MobiLink gives priority to **-zwe**, then **-zwd**, then **-zw**.

The default is **1,2,3,4,5** which indicates that all levels of warning message should be displayed.

-zwd option

Function Disables specific warning codes.

Syntax **dbmlsrv8 -c "connection-string" -zwd code,...**

Description You can disable specific warning codes so that they will not be reported, even though other codes of the same level are reported.

☞ For a complete list of warning message codes, see "MobiLink synchronization server Warning Messages" on page 683.

If `-zwd` is used more than once in the same command line, MobiLink accumulates the settings. If settings of `-zw`, `-zwd`, and `-zwe` conflict, MobiLink gives priority to `-zwe`, then `-zwd`, then `-zw`.

-zwe option

Function

Enables specific warning codes.

Syntax

dbmlsrv8 -c "connection-string" -zwe code,...

Description

You can enable specific warning codes so that they will be reported even though you have disabled other codes of the same level using `-zw`.


☞ For a complete list of warning message codes, see "MobiLink synchronization server Warning Messages" on page 683.

If `-zwe` is used more than once on the same command line, MobiLink accumulates the settings. If settings of `-zw`, `-zwd`, and `-zwe` conflict, MobiLink gives priority to `-zwe`, then `-zwd`, then `-zw`.

MobiLink Synchronization Client

About this chapter

This chapter describes details of the MobiLink synchronization client, dbmlsync. It is used to synchronize Adaptive Server Anywhere remote databases with a consolidated database.

 The dbmlsync utility only works with Adaptive Server Anywhere remote databases. To synchronize UltraLite remote databases, see "Introduction to UltraLite" on page 3 of the book *UltraLite User's Guide*.

Contents

| Topic | Page |
|---------------------------------|------|
| MobiLink synchronization client | 410 |
| dbmlsync options | 413 |

MobiLink synchronization client

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

Syntax

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

| Option | Description |
|--|---|
| -a | Do not prompt for input again on error. |
| -c <i>connection-string</i> | Supply database connection parameters in the form <i>parm1=value1; parm2=value2,...</i> . If you do not supply this option, a dialog will appear and you must supply connection information. See "-c option" on page 413. |
| -d | Drop any other connections to the database whose locks conflict with the articles to be synchronized. See "-d option" on page 413. |
| -dl | Display log messages on the console. See "-dl option" on page 413. |
| -e " <i>keyword=value;...</i> " | Specify extended options. See "-e extended options" on page 414. |
| -eh | Ignore errors that occur in hook functions. |
| -ek <i>key</i> | Specify encryption key. See "-ek option" on page 423. |
| -ep | Prompt for encryption key. See "-ep option" on page 423. |
| -eu | Specify extended options for upload defined by most recent -n option. See "-eu option" on page 423. |
| -i <i>filename</i> | Execute file containing SQL statements immediately after synchronization. See "-i option" on page 424. |
| -is | Ignore schedule. See "-is option" on page 424. |
| -k | Close window on completion. See "-k option" on page 424. |
| -l | List available extended options. See "-l option" on page 424. |
| -mn <i>password</i> | Specify new MobiLink password. See "-mn option" on page 425. |
| -mp <i>password</i> | Specify MobiLink password. See "-mp option" on page 425. |
| -n <i>name</i> | Specify synchronization publication name. See "-n option" on page 425. |
| -o <i>logfile</i> | Log output messages to this file. See "-o option" on |

| Option | Description |
|---|--|
| | page 426. |
| -os <i>size</i> | Maximum size of output file. See "-os option" on page 426. |
| -ot <i>logfile</i> | Truncate file and log output messages to file. See "-ot option" on page 426. |
| -p | Disable logscan polling. See "-p option" on page 427. |
| -pi | Test that you can connect to MobiLink. See "-pi option" on page 427. |
| -pp <i>number</i> [h m s] | Set logscan polling period. See "-pp option" on page 428. |
| -q | Run in minimized window. See "-q option" on page 428. |
| -r [a b] | Upload retry on client progress. See "-r option" on page 429. |
| -u <i>ml_username</i> | Allows you to specify the MobiLink user to synchronize. See "-u option" on page 429. |
| -urc <i>row_estimate</i> | Allows you to specify and estimate of the rows that will be uploaded. See "-urc option" on page 430. |
| -v [<i>levels</i>] | Verbose operation. See "-v option" on page 430. |
| -wc <i>classname</i> | Specify a Windows class name for ActiveSync synchronization (Windows CE only). See "-wc option" on page 431. |
| -x | Rename and restart the transaction log. See "-x option" on page 431. |
| <i>transaction-logs-directory</i> | Specify the location of the transaction log. See "Transaction log files" on page 140. |

Description

Run `dbmlsync` on the command line to synchronize an Adaptive Server Anywhere remote database with a consolidated database. `Dbmlsync` uses the information on the publication, synchronization user or synchronization subscription to locate and connect to the MobiLink synchronization server.

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

- ◆ the working log file has been truncated and renamed since you last synchronized
- ◆ you run the `dbmlsync` utility from a directory other than the one where the renamed log files are stored

TableOrder You must specify the extended option TableOrder in some cases. For more information, see "TableOrder" on page 421 and "Referential integrity and synchronization" on page 35.

Using a configuration file You can put dbmlsync command line options in a configuration file and optionally use the File Hiding utility, dbfhide, to add simple encryption to the configuration file. For more information, see "Using configuration files" on page 10 of the book *ASA Database Administration Guide* and "The File Hiding utility" on page 446 of the book *ASA Database Administration Guide*.

Dbmlsync event hooks There are also dbmlsync client stored procedures that can help you customize the synchronization process. For more information, see "Customizing the client synchronization process" on page 157 and "Client event-hook procedures" on page 592.

Using dbmlsync For more information about using dbmlsync, see "Initiating synchronization" on page 138.

dbmlsync options

This section lists MobiLink synchronization client command line options.

-c option

Function

Specifies connection parameters for the remote database.

Syntax

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

Description

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

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

If you do not specify **-c**, a DBMLSync Setup dialog appears. You can specify the remaining command line options in the fields of the connection dialog.

For a complete list of connection parameters for connecting to Adaptive Server Anywhere databases, see "Connection parameters" on page 164 of the book *ASA Database Administration Guide*.

-d option

Function

Drops conflicting locks to the remote database.

Syntax

dbmlsync -d ...

Description

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

-dl option

Function



Displays messages in the log file.

| | |
|--------------------|--|
| Syntax | dbmlsync -dl ... |
| Description | Normally when output is logged to a file, more messages are written to the file than to the dbmlsync window. This option forces dbmlsync to write information normally only written to the file to the window as well. |

-e extended options


| | |
|--------------------|--|
| Function | Specifies extended options. |
| Syntax | dbmlsync -e <i>keyword=value</i>; ... |
| Description | Specify one or more of the following extended options. |

| Extended option | Short name | Default | Description |
|----------------------|------------|-----------|--|
| CommunicationAddress | adr | localhost | <p>Specifies the communication address for connecting to the MobiLink server.</p> <p>When using <code>adr</code> on the command line, you must ensure that all subscriptions for a single synchronization user are always synchronized to only one consolidated database. Using this extended option to synchronize publications for one user to more than one consolidated database will result in data loss and strange behavior.</p> <p>For allowed values, see "CREATE SYNCHRONIZATION USER statement [MobiLink]" on page 335 of the book <i>ASA SQL Reference Manual</i>.</p> |

| Extended option | Short name | Default | Description |
|-------------------|------------|----------------------------|--|
| CommunicationType | ctp | tcpip | <p>Specifies the communication type for connecting to the MobiLink server.</p> <p>When using ctp on the command line, you must ensure that all subscriptions for a single synchronization user are always synchronized to only one consolidated database. Using this extended option to synchronize publications for one user to more than one consolidated database will result in data loss and strange behavior.</p> <p> For allowed values, see "CREATE SYNCHRONIZATION USER statement [MobiLink]" on page 335 of the book <i>ASA SQL Reference Manual</i>.</p> |
| ConflictRetries | cr | -1 (Continue indefinitely) | <p>Number of retries if download fails because of conflicts.</p> <p> For more information, see "Concurrency during synchronization" on page 140.</p> |
| DisablePolling | p | OFF | <p>Disable automatic logscan polling when scheduling is enabled.</p> |

| Extended option | Short name | Default | Description |
|--------------------|------------|--|---|
| DownloadBufferSize | dbs | 32 Kb on Windows CE, 1 Mb on all other operating systems. | <p>The valid settings are n, nK, and nM, where n is zero or a positive integer and the units are in bytes. If the setting is zero, dbmlsync will not buffer the download stream. If the setting is greater than zero, but less than 4K, dbmlsync will give a warning message and automatically use 4K memory for buffering the download stream.</p> <p>Download buffering at the client side is advisable for use with no download acknowledgement. Enable download buffering at the client to get the most benefit from eliminating the download ack. That way the worker thread can send the download faster, as it won't have to wait while the client applies it.</p> |

| Extended option | Short name | Default | Description |
|-------------------|------------|---------|--|
| ErrorLogSendLimit | EL | 32 Kb | <p>Sends the dbmlsync remote output log to MobiLink server when an error occurs on the remote. The option controls the number of bytes dbmlsync sends of its output log to the MobiLink server when errors occurred during synchronization.</p> <p>The valid settings for this extended option are <i>n</i>, <i>nk</i>, <i>nK</i>, <i>nm</i>, or <i>nM</i>, where <i>n</i> is zero or an integer, and k, K, m, or M specify megabytes or kilobytes. If you don't want to send any dbmlsync output log messages, a zero value should be set for this extended option.</p> <p>The contents of the log are determined by the verbosity settings: either the -v command line option or extended options starting with "verbose".</p> |
| FireTriggers | ft | ON | Fire triggers on download. |
| IgnoreHookErrors | eh | OFF | Ignore errors that occur in hook function. |
| IgnoreScheduling | isc | OFF | Ignore scheduling information. |

| Extended option | Short name | Default | Description |
|------------------|------------|-------------|---|
| Increment | inc | NULL | Send the upload stream in chunks of roughly the specified size. Once the specified size is reached, a chunk is sent at the next point in the stream at which there are no outstanding partial transactions. MobiLink does not send a download stream to the remote until the last chunk has been received. If you do not specify an increment size, the whole upload stream is sent in a single transfer (the default). |
| LockTables | lt | ON | Set to OFF to allow modifications during synchronization  For more information, see "Concurrency during synchronization" on page 140. |
| Memory | mem | 1 Mb | Memory used for building upload stream. |
| MobiLinkPwd | MP | NULL | MobiLink Password. |
| NewMobiLinkPwd | mn | NULL | New MobiLink password. |
| OfflineDirectory | dir | NULL | Path containing offline transaction logs. |
| PollingPeriod | pp | 1 minute | Logscan polling period. |
| Schedule | sch | No schedule | Schedule string. For more information, see "Schedule option syntax" on page 347 of the book <i>ASA SQL Reference Manual</i> . |
| ScriptVersion | sv | default | Tells the MobiLink synchronization server to use the scripts for the named schema version. |

| Extended option | Short name | Default | Description |
|--------------------------------|------------|---------|---|
| SendColumnNames | scn | OFF | ON tells dbmlsync to send column names from the remote database to the server. Used typically for generating scripts automatically using the -za or -ze option on dbmlsrv8. |
| SendDownloadACK | sa | ON | ON tells <i>dbmlsync</i> to send a download acknowledgement from the client to the server. An acknowledgement line appears in the client log in verbose mode. OFF means dbmlsync does not send the acknowledgement. Turning the acknowledgement off can lead to less contention in the consolidated database and also increased throughput due to shorter download transactions. Download transactions are shorter because they are committed or rolled back as soon as possible, since MobiLink doesn't need to keep these transactions open for as long as it takes the remote client to apply the download. Enable client side download buffering to get the most performance out of eliminating the download acknowledgement. |
| SendTriggers | st | OFF | Send trigger actions on upload. |
| SiteScriptName | sn | NULL | File containing SQL statements to be executed after synchronization. See "-i option" on page 424. |
| StreamCompression (deprecated) | sc | MEDIUM | This option has been deprecated and is ignored. |

| Extended option | Short name | Default | Description |
|------------------|------------|---------|--|
| TableOrder | to | default | Allows users to specify the ordering of table upload during synchronization for referential integrity resolution. Use with a comma delimited list. All tables that are to be uploaded in the synchronization must be listed. Other tables not involved in the synchronization may be included and will be ignored. |
| Verbose | v | OFF | Full verbosity. This is the same as setting the command line option <code>-v+</code> . |
| VerboseHooks | vs | OFF | Log messages related to hook scripts. |
| VerboseMin | vm | OFF | Log a minimal amount of information. |
| VerboseOptions | vo | OFF | Log a list of the extended options you have specified. |
| VerboseRowCounts | vn | OFF | Log the number of rows that were uploaded or downloaded. |
| VerboseRowValues | vr | OFF | Log the values of rows that were uploaded or downloaded. |
| VerboseUpload | vu | OFF | Log information about the upload stream. |

Description

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

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

In contrast, options specified on the command line with the `-eu` option apply only to the synchronization specified by the `-n` option they follow.

Extended options can be specified on the dbmlsync command line using the -e or -eu options, or they can be stored in the database. You store extended options in the database using Sybase Central; or by using the **OPTIONS** clause in any of the following statements:

- ◆ **CREATE SYNCHRONIZATION SUBSCRIPTION**
- ◆ **ALTER SYNCHRONIZATION SUBSCRIPTION**
- ◆ **CREATE SYNCHRONIZATION USER**
- ◆ **ALTER SYNCHRONIZATION USER**
- ◆ **CREATE SYNCHRONIZATION SUBSCRIPTION** without specifying a synchronization user. (This associates extended options with a publication.)

When you store extended options and connection parameters in the database, dbmlsync reads the information from the database. 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.

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

You should ensure that the extended options you provide in your synchronization subscription are ones you want for your synchronization. You can review extended options in the log and the *syssync* system table.

When using *adr* or *ctp* on the command line, you must ensure that all subscriptions for a single synchronization user are always synchronized to only one consolidated database. Using these extended options to synchronize publications for one user to more than one consolidated database will result in data loss and strange behavior.

See also

"-eu option" on page 423

-eh option

| | |
|-----------------|--|
| Function | Ignores errors that occur in hook functions. |
| Syntax | dbmlsync -eh ... |

-ek option

| | |
|--------------------|--|
| Function | Allows you to specify the encryption key for strongly encrypted databases directly on the command line. |
| Syntax | dbmlsync -ek key ... |
| Description | If you have a strongly encrypted database, you must provide the encryption key to use the database or transaction log in any way, including offline transactions. For strongly encrypted databases, you must specify either -ek or -ep, but not both. The command will fail if you do not specify a key for a strongly encrypted database. |

-ep option

| | |
|--------------------|--|
| Function | Prompt for the encryption key. |
| Syntax | dbmlsync -ep ... |
| Description | This option causes a dialog box to appear, in which you enter the encryption key. It provides an extra measure of security by never allowing the encryption key to be seen in clear text. For strongly encrypted databases, you must specify either -ek or -ep, but not both. The command will fail if you do not specify a key for a strongly encrypted database. |

-eu option

| | |
|--------------------|--|
| Function | Specifies extended upload options. |
| Syntax | dbmlsync -n publication-name -eu keyword=value;... |
| Description | Options specified on the command line with the -eu option apply only to the synchronization specified by the -n option they follow. For example, on the following command line, the extended option sv=test applies only to the synchronization of pub2. |

```
dbmlsync -n pub1 -n pub2 -eu sv=test
```

If extended options are specified in more than one way, the specified options are combined.

☞ For more information, see "-e extended options" on page 414.

-i option

| | |
|--------------------|--|
| Function | Executes the SQL script contained in the named file. |
| Syntax | dbmlsync -i filename ... |
| Description | Immediately upon completing synchronization, and before releasing the table locks, execute the SQL script contained in the named file. This option is intended for upgrading applications and making schema changes to deployed, remote databases. Schema changes must be accomplished in this manner to ensure all changes are in a compatible format. The script specified is executed if dbmlsync received confirmation from MobiLink that the upload was applied even if an error occurs during the download. You should be explicit about commit/rollback operations when using the -i option. Failure to do so may cause inconsistent results. |

-is option

| | |
|--------------------|---|
| Function | Ignores scheduling instructions so that synchronization is immediate. |
| Syntax | dbmlsync -is ... |
| Description | Ignore extended options that schedule synchronization. ☞ For information about scheduling, see "Scheduling synchronization" on page 162. |

-k option

| | |
|--------------------|--|
| Function | Closes window on completion. |
| Syntax | dbmlsync -k ... |
| Description | Close window on completion, if used together with the -o option. |

-l option


| | |
|-----------------|-----------------------------------|
| Function | Lists available extended options. |
| Syntax | dbmlsync -l ... |

Description When used with the `dbmlsync` command line it shows you available extended options.

-mn option

Function Supplies a new password for the user being synchronized.


Syntax **dbmlsync -mn** *password* ...

Description Changes the MobiLink user's password.
 For more information, see "Authenticating MobiLink Users" on page 251.

-mp option

Function Supplies the password of the user being synchronized.

Syntax **dbmlsync -mp** *password* ...

Description Supplies the password for MobiLink user authentication.
 For more information, see "Authenticating MobiLink Users" on page 251.

-n option

Function Names the synchronization publication.

Syntax **dbmlsync -n** *pubname* ...

Description Name of synchronization publication. You can supply more than one `-n` option to synchronize more than one synchronization publication.

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

- ◆ Specify `-n pub1, pub2, pub3` to upload `pub1`, `pub2` and `pub3` in one upload stream.

In this case, if you have set extended options on the publications, only the options set on the first publication in the list are used. Extended options set on subsequent publications are ignored.

- ◆ Specify `-n pub1 -n pub2 -n pub3` to upload `pub1` in one upload stream, `pub2` in another, and `pub3` in a third upload stream.

There are cases where dbmlsync never terminates after synchronizing the first publication, so the second will never be synchronized:

If an `sp_hook_dbmlsync_end` hook is defined, and the hook always sets the `restart` parameter to true, then dbmlsync repeatedly synchronizes the first publication and never synchronizes the second. You must be aware of whether or not the `sp_hook_dbmlsync_end_hook` is defined before specifying multiple publications with `-n`.

-o option

| | |
|--------------------|---|
| Function | Sends output to a log file. |
| Syntax | dbmlsync -o <i>filename</i> ... |
| Description | Append output to a log file. Default is to send output to the screen. |

-os option

| | |
|--------------------|--|
| Function | Specifies the maximum size of the output log messages. |
| Syntax | dbmlsync -os <i>size</i> ... |
| Description | <p>The <i>size</i> is the maximum file size for logging output messages, specified in units of bytes. Use the suffix k, m or g to specify units of kilobytes, megabytes or gigabytes, respectively. By default, there is no size limit. The minimum size limit is 10 kb.</p> <p>Before the dbmlsync utility logs output messages to a file, it checks the current file size. If the log message will make the file size exceed the specified size, the dbmlsync utility renames the output file to <code>yymmddxx.dbr</code>. Here, <code>xx</code> are sequential characters ranging from <code>AA</code> to <code>ZZ</code>, and <code>yymmdd</code> represents the year, month, and day.</p> <p>This option allows you to manually delete old log files and free up disk space.</p> |

-ot option

| | |
|--------------------|---|
| Function | Truncates the log file and appends output messages to it. |
| Syntax | dbmlsync -ot <i>logfile</i> ... |
| Description | The functionality is the same as the <code>-o</code> option except the log file is truncated before any messages are written to it. |

-p option

| | |
|--------------------|---|
| Function | Disables log scanning. |
| Syntax | dbmlsync -p ... |
| Description | When scheduling is enabled, the log is scanned by default once per minute. If changes have been made to any of the synchronized tables, then at the next scheduled synchronization time, a synchronization session is initiated and the affected rows uploaded. This option disables this feature. If present, the client synchronization utility performs one synchronization session, then exits. Polling improves performance by polling and caching the contents. Upon the next poll, there is less log content available to scan, thus performance on the subsequent scan is improved. |

-pi option

| | |
|--------------------|--|
| Function | Pings a MobiLink synchronization server. |
| Syntax | dbmlsync -pi -c <i>connection_string</i> -e <i>sv=script_version</i> [-n <i>pubname</i>] [-u <i>ml_username</i>] |
| Description | <p>The ping option allows you to test that your connection information is correct. When you use -pi, dbmlsync does not initiate synchronization.</p> <p>In order to be able to ping, dbmlsync must have a unique address for the MobiLink synchronization server. This means that you must include connection parameters and a script version, as well as the publication name, MobiLink user name, or both. The publication or user hold connection information for the remote. You need to specify both when the user is subscribed to multiple publications or the publication has multiple users. For example, if there is only one subscription to the publication, you can specify the publication without the user.</p> <p>When the MobiLink synchronization server receives a ping request, it connects to the consolidated database, authenticates the user, and then sends the authenticating user status and value back to the client (dbmlsync or UltraLite).</p> <p>If the ping succeeds, the MobiLink server issues an information message. If the ping does not succeed, it issues an error message.</p> <p>If the MobiLink user name cannot be found in the ml_user system table and the MobiLink server is running with the command line option -zu+, the MobiLink server adds the user to ml_user.</p> |

The MobiLink synchronization server may execute the following scripts, if they exist:

- ◆ begin_connection
- ◆ authenticate_user
- ◆ authenticate_user_hashed
- ◆ end_connection

The client cannot synchronize while it is pinging the server.

-pp option

Function

Specifies the frequency of log scans.

Syntax

dbmlsync -pp *frequency* ...

Description

When scheduling is enabled, the log is scanned by default once per minute. If changes have been made to any of the synchronized tables, a synchronization session is initiated and the affected rows uploaded. This option allows you to specify how frequently log scans are to be initiated. Use the suffix h, m, or s to specify units of hours, minutes or seconds. Minutes are assumed if no suffix is present. If a scan takes longer than the specified period, or the specified period is zero, a new scan is initiated as soon as the previous one completes. The default scan period is 1 minute.

☞ For more information, see "-p option" on page 427.

-q option

Function

Starts the MobiLink synchronization client in a minimized window.

Syntax

dbmlsync -q ...

Description

For Windows operating systems only, starts dbmlsync with a minimized window.

-r option

Function

When you use `-ra` the upload continues from the offset recorded in the remote database as long as the offset recorded in the remote is after that recorded for the consolidated database. When you use `-rb`, the upload continues from the offset recorded in the remote database as long as the offset recorded in the remote is before that recorded for the consolidated database.

Syntax

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

Description

When either the client or consolidated database needs to be restored from a backup, the records of the most recently transmitted offset in the client transaction log can mismatch between the client and the consolidated database. In this case, the options are to continue upload synchronization from the remote database or the consolidated database's record of the last synchronization point. Which one is appropriate depends in part on the synchronization scripts you use.

By default, uploads continue from the offset recorded by the consolidated database. If you use `-ra` or `-rb`, the upload continues from the offset recorded in the remote database as long as the offset recorded in the consolidated database is ahead of that at the remote database.

If you use `-ra`, the upload is retried starting from the offset recorded in the remote database even if the offset recorded in the remote database is ahead of that recorded in the consolidated database. This option should be used with care. If the offset mismatch is the result of a restore of the consolidated database, changes that happened in the remote database in the gap between the two recorded offsets are lost. The `-ra` option may be useful when the remote database transaction log has been truncated and the synchronization definition recreated after the last successful synchronization.

-u option

Function

Specifies the MobiLink user name.

Syntax

dbmlsync `-u ml_username` ...

Description

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

This option should be used in conjunction with `-n publication` to identify the subscription on which `dbmlsync` should operate. Each subscription is uniquely identified by an `ml_username`, `publication` pair.

You can only specify one user name on the command line. All subscriptions to be synchronized in a single run must involve the same user. The -u option can be omitted if each publication specified on the command line has only one subscription.

-urc option

| | |
|--------------------|---|
| Function | Specifies an estimate of the number of rows to be uploaded in a synchronization. |
| Syntax | dbmlsync -urc <i>row_estimate</i> ... |
| Description | <p>To improve performance, you can specify an estimate of the number of rows that will be uploaded in a synchronization. In general, a higher estimate results in faster uploads but more memory usage.</p> <p>Synchronization will proceed correctly regardless of the estimate that is specified.</p> |


-v option

| | |
|--------------------|---|
| Function | Allows you to specify what information is logged to the message log file and displayed in the synchronization window. A high level of verbosity may affect performance and should normally be used in the development phase only. |
| Syntax | dbmlsync -v [<i>levels</i>] ... |
| Description | <p>The -v options affect the message log file and synchronization window. You only have a message log if you specify -o or -ot on the dbmlsync command line.</p> <p>If you specify -v alone, a minimal amount of information is logged.</p> <p>The values of <i>levels</i> are as follows. You can use one or more of these options at once; for example, -vnrsu or -v+cp.</p> <ul style="list-style-type: none">◆ + Turn on all logging options except for c and p.◆ c Expose the connect string in the log.◆ p Expose the password in the log.◆ n Log the number of rows that were uploaded and downloaded.◆ o Log information about the command line options and extended options that you have specified.◆ r Log the values of rows that were uploaded and downloaded. |

- ◆ **s** Log messages related to hook scripts.
- ◆ **u** Log information about the upload stream.

There are extended options that have similar functionality to the `-v` options. If you specify both `-v` and the extended options and there are conflicts, the `-v` option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used.

When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the `-v` options and the extended options.

 For information about the extended options, see extended options starting with the word "verbose" in "-e extended options" on page 414.

-wc option

| | |
|--------------------|---|
| Function | For Windows CE only, this option specifies a Windows class name for use with ActiveSync synchronization. |
| Syntax | dbmlsync -wc <i>class-name</i> ... |
| Description | This option specifies a class name that identifies the application for ActiveSync synchronization. The class name must be given when registering the application for use with ActiveSync synchronization. |
| See also | "Registering Adaptive Server Anywhere clients for ActiveSync" on page 146 "Using ActiveSync synchronization" on page 143 |

-x option

| | |
|--------------------|--|
| Function | Renames and restarts the transaction log after it has been scanned for outgoing messages. |
| Syntax | dbmlsync -x ... |
| Description | In some circumstances, synchronizing data to a consolidated database can take the place of backing up remote databases, or renaming the transaction log when the database server is shut down. |

If backups are not routinely performed at the remote database, the transaction log continues to grow. As an alternative to using the -x option to control transaction log size, you can use an Adaptive Server Anywhere event handler to control the size of the transaction log. For example, the following event handler renames the transaction log at the remote database when its size exceeds 5 Mb. You can use such an event handler together with the DELETE_OLD_LOGS database option to control the space taken up by transaction logs.

```
CREATE EVENT RenameLogLimit
TYPE GrowLog
WHERE event_condition( 'LogSize' ) > 5
AT REMOTE
HANDLER
BEGIN
    BACKUP DATABASE DIRECTORY backupdir
    TRANSACTION LOG ONLY
    TRANSACTION LOG RENAME
END
```

☞ For more information, see "Automating Tasks Using Schedules and Events" on page 231 of the book *ASA Database Administration Guide*, and "DELETE_OLD_LOGS option" on page 566 of the book *ASA Database Administration Guide*.

CHAPTER 20

Synchronization Events

About this chapter

This chapter provides information about the MobiLink synchronization events and the SQL scripts, Java methods, or .NET methods that handle these events. You implement scripts to handle one or more of these events to control the actions of the MobiLink synchronization server.

Contents

| Topic | Page |
|---|------|
| Overview of MobiLink events | 436 |
| authenticate_user connection event | 446 |
| authenticate_user_hashed connection event | 450 |
| begin_connection connection event | 452 |
| begin_download connection event | 454 |
| begin_download table event | 456 |
| begin_download_deletes table event | 458 |
| begin_download_rows table event | 460 |
| begin_synchronization connection event | 462 |
| begin_synchronization table event | 464 |
| begin_upload connection event | 466 |
| begin_upload table event | 468 |
| begin_upload_deletes table event | 470 |
| begin_upload_rows table event | 472 |
| download_cursor cursor event | 474 |
| download_delete_cursor cursor event | 477 |
| download_statistics connection event | 479 |
| download_statistics table event | 482 |
| end_connection connection event | 485 |
| end_download connection event | 487 |
| end_download table event | 489 |

| | |
|--|-----|
| end_download_deletes table event | 491 |
| end_download_rows table event | 493 |
| end_synchronization connection event | 495 |
| end_synchronization table event | 497 |
| end_upload connection event | 499 |
| end_upload table event | 502 |
| end_upload_deletes table event | 504 |
| end_upload_rows table event | 506 |
| example_upload_cursor table event | 508 |
| example_upload_delete table event | 509 |
| example_upload_insert table event | 510 |
| example_upload_update table event | 511 |
| handle_error connection event | 512 |
| handle_odbc_error connection event | 515 |
| modify_last_download_timestamp connection event | 517 |
| modify_next_last_download_timestamp connection event | 519 |
| modify_user connection event | 521 |
| new_row_cursor cursor event | 523 |
| old_row_cursor cursor event | 525 |
| prepare_for_download connection event | 527 |
| report_error connection event | 529 |
| report_odbc_error connection event | 531 |
| resolve_conflict table event | 533 |
| synchronization_statistics connection event | 535 |
| synchronization_statistics table event | 537 |
| time_statistics connection event | 539 |
| time_statistics table event | 541 |
| upload_cursor cursor event | 543 |
| upload_delete table event | 545 |
| upload_fetch table event | 547 |
| upload_insert table event | 549 |
| upload_new_row_insert table event | 551 |

| | |
|------------------------------------|-----|
| upload_old_row_insert table event | 553 |
| upload_statistics connection event | 554 |
| upload_statistics table event | 557 |
| upload_update table event | 560 |

Overview of MobiLink events

The following pseudo code provides an overview of the sequence in which events, and hence the script of the same name, are invoked.

Synchronization events in pseudo-code.

- Variables are shown with MixedCase and assigned with:
var <- value

CONNECT to consolidated database
begin_connection
COMMIT
for each synchronization request {
 <synchronize>
}
end_connection
COMMIT
DISCONNECT from consolidated database

synchronize

<authenticate>
begin_synchronization
COMMIT
<upload>
<prepare_for_download>
<download>
end_synchronization
synchronization_statistics
time_statistics
COMMIT

authenticate

```
Status <- 1000
UseDefaultAuthentication <- TRUE
if( authenticate_user is defined ) {
    UseDefaultAuthentication <- FALSE
    TempStatus <- authenticate_user
    if( TempStatus > Status ) {
        Status <- TempStatus
    }
}
if( authenticate_user_hashed is defined ) {
    UseDefaultAuthentication <- FALSE
    TempStatus <- authenticate_user_hashed
    if( TempStatus > Status ) {
        Status <- TempStatus
    }
}
if( UseDefaultAuthentication ) {
    if( the user exists in the ml_user table ) {
        if( ml_user.hashed_password column is not NULL ) {
            if( password matches ml_user.hashed_password ) {
                Status <- 1000
            } else {
                Status <- 4000
            }
        } else {
            Status <- 1000
        }
    } else if( -zu+ was on the command line ) {
        Status <- 1000
    } else {
        Status <- 4000
    }
}
if( Status >= 3000 ) {
    ROLLBACK
    // Abort the synchronization.
} else {
    // UserName defaults to the MobiLink user name.
    if( modify_user script is defined ) {
        UserName <- modify_user
    }
    // The value of UserName is later passed to all
    // scripts that expect the MobiLink user name.
    if( authenticate_user script is defined or
        authenticate_user_hashed script is defined or
        modify_user script is defined ) {
        if( no error in calling these scripts ) {
            COMMIT
        }
    }
}
```

```
        } else {  
            ROLLBACK  
            // Abort synchronization  
        }  
    }  
}
```

☞ For the details of upload stream processing, see "Events during upload" on page 438.

☞ For the details of download stream processing, see "Events during download" on page 444.

Notes

- ◆ MobiLink synchronization allows multiple clients to synchronize concurrently.
- ◆ A single connection to the consolidated database can be used for several synchronizations from different clients, one after the other.
- ◆ The `begin_connection` and `end_connection` events are **connection-level events**. They are independent of any single synchronization and have no parameters.
- ◆ Some events are invoked once per synchronization for each table being synchronized. Scripts associated with these events are called **table-level scripts**.

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.
- ◆ A database error event can occur at any point within the synchronization process. Database errors are handled using the following script.

```
handle_error( error_code, error_message,  
             user_name, table_name )
```

- ◆ An ODBC error can be handled by the `handle_odbc_error` event.

Events during upload

The following pseudo code illustrates how upload events and upload scripts are invoked.

These events take place at the `process_upload_stream` location in the complete event model. For more information, see "Overview of MobiLink events" on page 436.

upload

```
begin_upload
  for each table being synchronized {
    begin_upload_rows
      for each uploaded INSERT or UPDATE for this table {
        if( INSERT ) {
          <upload_inserted_row>
        }
        if( UPDATE ) {
          <upload_updated_row>
        }
      }
    end_upload_rows
  }
  for each table being synchronized IN REVERSE ORDER {
    begin_upload_deletes
      for each uploaded DELETE for this table {
        <upload_deleted_row>
      }
    end_upload_deletes
  }
if( no error in calling any of these scripts ) {
  end_upload
  upload_statistics
  COMMIT
} else {
  ROLLBACK
  // Abort the synchronization
}
```

<upload_inserted_row>

NOTE: Only table scripts for the current table are involved.

```
UploadUsingStatements <- ( upload_insert is defined
                           or upload_update is defined
                           or upload_delete is defined
                           or upload_fetch is defined
                           or upload_new_row_insert is defined
                           or upload_old_row_insert is defined )
if( UploadUsingStatements ) {
  ConflictsAreExpected <- ( upload_new_row_insert is
                           defined or upload_old_row_insert is defined
                           or resolve_conflict is defined )
  if( upload_insert is defined ) {
    upload_insert
  } else if( ConflictsAreExpected
             and upload_update is not defined
             and upload_insert is not defined
             and upload_delete is not defined ) {
    // Forced conflict.
    upload_new_row_insert
    resolve_conflict
  }
} else {
  // Upload with cursors.
  ConflictsAreExpected <- ( new_row_cursor is defined
                           or old_row_cursor is defined
                           or resolve_conflict is defined )
  if( upload_cursor is defined ) {
    INSERT using upload_cursor
  } else if( ConflictsAreExpected ) {
    INSERT using new_row_cursor
    resolve_conflict
  }
}
```

upload_updated_row

Only table scripts for the current table are involved. Both the old (original) and new rows are uploaded for each update.

```
UploadUsingStatements <- ( upload_insert is defined
                           defined(upload_update)
                           or defined(upload_delete)
                           or upload_fetch is defined
                           or upload_new_row_insert is defined
                           or upload_old_row_insert is defined )
if( UploadUsingStatements ) {
  ConflictsAreExpected <- ( upload_new_row_insert is
                           defined or upload_old_row_insert is defined
                           or resolve_conflict is defined )
  Conflicted <- FALSE
  if( upload_update is defined ) {
    if( ConflictsAreExpected
        and upload_fetch is defined ) {
      FETCH using upload_fetch INTO current_row
      if( current_row <> old_row ) {
        Conflicted <- TRUE
      }
    }
    if( not Conflicted ) {
      upload_update
    }
  } else if( upload_update is not defined
             and upload_insert is not defined
             and upload_delete is not defined ) {
    // Forced conflict.
    Conflicted <- TRUE
  }
  if( ConflictsAreExpected and Conflicted ) {
    upload_old_row_insert
    upload_new_row_insert
    resolve_conflict
  }
} else {
  // Upload with cursors.
  ConflictsAreExpected <- ( new_row_cursor is defined
                           or old_row_cursor is defined
                           or resolve_conflict is defined )
  Conflicted <- FALSE
  if( upload_cursor is defined ) {
    FETCH using upload_cursor INTO current_row
    if( ConflictsAreExpected and
        current_row <> old_row )
  {
    Conflicted <- TRUE
  }
}
```

```
        } else {  
            UPDATE using upload_cursor  
        }  
    } else {  
        // Forced conflict.  
        Conflicted <- TRUE  
    }  
    if( ConflictsAreExpected and Conflicted ) {  
        INSERT using new_row_cursor  
        INSERT using old_row_cursor  
        resolve_conflict  
    }  
}
```

upload_deleted_row

NOTE: Only table scripts for the current table are considered.

```

UploadUsingStatements <- ( upload_insert is defined
                          or upload_update is defined
                          or upload_delete is defined
                          or upload_fetch is defined
                          or upload_new_row_insert is defined
                          or upload_old_row_insert is defined )
if( UploadUsingStatements ) {
  ConflictsAreExpected <- ( upload_new_row_insert is
                           defined
                           or upload_old_row_insert is defined
                           or resolve_conflict is defined )
  if( upload_delete is defined ) {
    upload_delete
  } else if( ConflictsAreExpected
             and upload_update is not defined
             and upload_insert is not defined
             and upload_delete is not defined ) {
    // Forced conflict.
    upload_old_row_insert
    resolve_conflict
  }
} else {
  // Upload with cursors.
  ConflictsAreExpected <- ( new_row_cursor is defined
                           or old_row_cursor is defined
                           or resolve_conflict is defined )
  if( upload_cursor is defined ) {
    DELETE using upload_cursor
  } else if( ConflictsAreExpected ) {
    // Forced conflict.
    INSERT using old_row_cursor
    resolve_conflict
  }
}
}

```

Notes

- ◆ The upload starts and ends with connection events. Other events are table-level events.
- ◆ The begin_upload and end_upload scripts for each remote table hold logic that is independent of the individual rows being updated.

Events during download

The following pseudo code provides an overview of the sequence in which download events, and hence the script of the same name, are invoked.

These events take place at the `process_download_stream` location in the complete event model provided in "Overview of MobiLink events" on page 436.

```
-----  
prepare_for_download  
-----
```

```
if( prepare_for_download is defined ) {  
    prepare_for_download  
}  
if( modify_last_download_timestamp is defined ) {  
    call modify_last_download_timestamp script  
}  
if( prepare_for_download is defined or  
    modify_last_download_timestamp is defined ) {  
    if( no error in calling any of these scripts ) {  
        COMMIT  
    }  
    else {  
        ROLLBACK  
        // Abort the synchronization  
    }  
}
```

download

```

begin_download
for each table being synchronized {
  begin_download_deletes
  for each row in download_delete_cursor {
    send DELETE to remote
  }
  end_download_deletes
  begin_download_rows
  for each row in download_cursor {
    send INSERT WITH UPDATE to remote
  }
  end_download_rows
}
if (modify_next_last_download_timestamp is defined ) {
  call modify_next_download_timestamp
}
end_download
if( no error in calling any of these scripts and
  no error in sending the download stream ) {
  download_statistics
  COMMIT
} else {
  ROLLBACK
  // Abort the synchronization
}

```

Notes

- ◆ Like the upload stream, the download stream starts and ends with connection events. Other events are table-level events.
- ◆ If an acknowledgement is expected, and if no confirmation of the downloads is received from the client, the entire download transaction is rolled back in the consolidated database.
- ◆ 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 35.

authenticate_user connection event

Function Implements a custom user authentication mechanism.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------------|--------------------------------------|
| 1 | auth_status | INTEGER. This is an INOUT parameter. |
| 2 | ml_username | VARCHAR(128). |
| 3 | user_password | VARCHAR(128). |
| 4 | user_new_password | VARCHAR(128) |

Default action Use MobiLink built-in user authentication mechanism.

Description The MobiLink synchronization server executes this event upon starting each synchronization. It is executed before, and in the same transaction as, the begin_synchronization event.

You can use this event to replace the built-in MobiLink authentication mechanism with a custom mechanism. You may want to call into the authentication mechanism of your DBMS, or you may wish to implement features not present in the MobiLink built-in mechanism, such as password expiry or a minimum password-length.

The parameters used in an authenticate_user event are as follows:

- 1 auth_status** This required parameter is an INOUT parameter: a parameter that provides a value to the script, and could be given a new value by the script. The auth_status parameter indicates the overall success of the authentication, and can be set to one of the following values:

| Returned Value | Auth_status | Description |
|----------------------|-------------|---|
| $V \leq 1999$ | 1000 | Authentication succeeded. |
| $1999 < V \leq 2999$ | 2000 | Authentication succeeded: password expiring soon. |
| $2999 < V \leq 3999$ | 3000 | Authentication failed: password expired. |
| $3999 < V \leq 4999$ | 4000 | Authentication failed. |
| $4999 < V \leq 5999$ | 5000 | Authentication failed as user is already synchronizing. |
| $5999 < V$ | 4000 | If the returned value is greater than 5999, MobiLink interprets it as a returned value of 4000. |

- 2 **ml_username** This optional parameter indicates the user name for authentication purposes.
- 3 **user_password** This optional parameter indicates the password for authentication purposes. If the user does not supply a password, this is NULL.
- 4 **user_new_password** This optional parameter indicates a new password. If the user does not change their password, this is NULL.

SQL scripts for the `authenticate_user` event must be implemented as stored procedures.

See also

"Authenticating MobiLink Users" on page 251
 "Custom user authentication mechanisms" on page 261
 "authenticate_user_hashed connection event" on page 450
 "begin_synchronization connection event" on page 462

SQL example

A typical `authenticate_user` script is a call to a stored procedure. The order of the parameters in the call must match the order above. In an Adaptive Server Anywhere consolidated database, the script could be as follows.

```
call my_auth ( ?, ?, ?, ? )
```

In an Adaptive Server Enterprise consolidated database, the Transact-SQL script could be as follows.

```
execute ? = my_auth ?, ?, ?
```

The following Adaptive Server Anywhere stored procedure uses only the user name to authenticate—it has no password check. The procedure ensures only that the supplied user name is one of the employee IDs listed in the `ULEmployee` table.

```
CREATE PROCEDURE my_auth(in @user_name varchar(128))
begin
  if exists
  ( select * from ulemmployee
    where emp_id = @user_name )
  then
    message 'OK' type info to client;
    return 1000;
  else
    message 'Not OK' type info to client;
    return 4000;
  end if
end
```

Java example

The following stored procedure call registers a Java method called `authenticateUser` as the script for the `authenticate_user` event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',
  'authenticate_user',
  'ExamplePackage.ExampleClass.authenticateUser' )
```

Following is the sample Java method `authenticateUser`. It calls Java functions that check and, if needed, change the user's password.

```
public String authenticateUser
( ianywhere.ml.script.InOutInteger authStatus,
  String user, String pwd, String newPwd )
throws java.sql.SQLException
{ // in a real authenticate_user handler, we would
  // handle more auth code states
  _curUser = user;

  if( checkPwd( user, pwd ) )
  { // auth successful
    if( newPwd != null )
    { // pwd is being changed
      if( changePwd( user, pwd, newPwd ) )
      { // auth ok and pwd change ok use custom code
        authStatus.setValue( 1001 ); }
      else { // authorization ok but password
        // change failed. Use custom code.
        java.lang.System.err.println( "user: "
          + user + " pwd change failed!" );
        authStatus.setValue( 1002 ); } }
    else { authStatus.setValue( 1000 ); } }
  else { // auth failed

    authStatus.setValue( 4000 ); }

  return( null ); }
```


.NET example

The following stored procedure call registers a .NET method called `AuthUser` as the script for the `authenticate_user` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script(  
    'ver1', 'authenticate_user',  
    'TestScripts.Test.AuthUser'  
)
```

Following is the C# signature for the call `AuthUser`.

```
public void AuthUser( ref int authStatus, string user,  
    string pwd, string newPwd )
```

 For a more detailed example of an `authenticate_user` script written in C# in .NET, see ".NET synchronization example" on page 200.

authenticate_user_hashed connection event

Function Implements a custom user authentication mechanism.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|----------------------|---|
| 1 | auth_status | INTEGER. This is an INOUT parameter. |
| 2 | ml_username | VARCHAR(128). |
| 3 | hashed_user_password | BINARY(20). If the user does not supply a password, this is NULL. |
| 4 | hashed_new_password | BINARY(20). If the user does not change their password, this is NULL. |

Default action Use MobiLink built-in user authentication mechanism.

Description This event is identical to authenticate_user except for the passwords, which are in the same hashed form as those stored in the ml_user.hashed_password column. Passing the passwords in hashed form provides increased security.

When the two authentication scripts are both defined, and both scripts return different auth_status codes, the higher value is used.

See also "Authenticating MobiLink Users" on page 251
"Custom user authentication mechanisms" on page 261
"authenticate_user connection event" on page 446

Java example The following stored procedure call registers a Java method called authUserHashed as the script for the authenticate_user_hashed event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script(  
  'ver1', 'authenticate_user_hashed',  
  'ExamplePackage.ExampleClass.authUserHashed
```

Following is the sample Java method authUserHashed. It calls Java functions that check and, if needed, change the user's password.


```
public String authUserHashed(
    anywhere.ml.script.InOutInteger authStatus,
    String user, byte pwd[], byte newPwd[] )
    throws java.sql.SQLException
{ // in a real authenticate_user_hashed handler, we
  // would handle more auth code states
  _curUser = user;
  if( checkPwdHashed( user, pwd ) ) {
    // auth successful
    if( newPwd != null )
    { // pwd is being changed
      if( changePwdHashed( user, pwd, newPwd ) )
      { // auth ok and pwd change ok use custom code
        authStatus.setValue( 1001 ); }
      else
      { // auth ok but pwd change failed.
        // Use custom code
        java.lang.System.err.println( "user: " + user
          + " pwd change failed!" );
        authStatus.setValue( 1002 ); } }
    else { authStatus.setValue( 1000 ); } }
    else { // auth failed
      authStatus.setValue( 4000 ); }
    return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called `AuthUserHashed` as the script for the `authenticate_user_hashed` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script( 'ver1',
  'authenticate_user_hashed',
  'TestScripts.Test.AuthUserHashed'
)
```

Following is the C# signature for the call `AuthUserHashed`.

```
public void AuthUserHashed( ref int authStatus, string
  user, byte[] pwd, byte[] newPwd )
```

begin_connection connection event

| | |
|-----------------------|--|
| Function | Processes any statements at the time the MobiLink synchronization server connects to the consolidated database server. |
| Parameters | None. |
| Default action | None. |
| Description | The MobiLink synchronization server executes this event upon opening each worker-thread connection to the consolidated database server. The MobiLink synchronization opens connections on demand as synchronization requests come in. When an application forms or reforms a connection with the MobiLink synchronization server, the MobiLink synchronization server temporarily allocates one connection with the database server for the duration of that synchronization. |
| See also | "end_connection connection event" on page 485 |
| SQL example | <p>The following SQL script works in an Adaptive Server Anywhere database. Two variables are created, one for the last_download timestamp, and one for employee ID.</p> <pre>call ml_add_connection_script('custdb', 'begin_connection', 'create variable @LastDownload timestamp; create variable @EmployeeID integer;')</pre> |
| Java example | <p><i>Note:</i> This script is not generally used in Java, because instead of database variables you would use member variables in this class instance, and prepare the members in the constructor.</p> <p>The following stored procedure call registers a Java method called beginConnection as the script for the begin_connection event when synchronizing the script version <i>ver1</i>. This syntax is for Adaptive Server Anywhere consolidated databases.</p> <pre>call ml_add_java_connection_script ('ver1', 'begin_connection', 'ExamplePackage.ExampleClass.beginConnection')</pre> <p>Following is the sample Java method beginConnection. This returns SQL that will create a connection level variable.</p> <pre>public String beginConnection() { return("create variable @LastDownload timestamp;"); }</pre> |

.NET example

The following stored procedure call registers a .NET method called `BeginConnection` as the script for the `begin_connection` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script( 'ver1',  
    'begin_connection',  
    'TestScripts.Test.BeginConnection'  
)
```

Following is the signature for the call `BeginConnection`.

```
public void BeginConnection()
```

begin_download connection event

Function Processes any statements just before the MobiLink synchronization server commences preparing the download data stream.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|--------------|
| 1 | last_download | TIMESTAMP |
| 2 | ml_username | VARCHAR(128) |

Default action None.

Description The MobiLink synchronization server executes this event as the first step in the processing of downloaded information. Download information is processed in a single transaction. The execution of this event is the first action in this transaction.

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 user has never synchronized successfully, this value is set to 1900-01-01.

See also "end_download connection event" on page 487

SQL example The following example works in an Adaptive Server Anywhere installation.

```
call ml_add_connection_script ( 'Lab', 'begin_download',  
    'CALL SetDownloadParameters ( ?, ? )' )
```

Java example The following stored procedure call registers a Java method called beginDownloadConnection as the script for the begin_download connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'example_ver',  
    'begin_download',  
    'ExamplePackage.ExampleClass.beginDownloadConnection' )
```

Following is the sample Java method beginDownloadConnection. It calls a Java function that will prepare the delete tables using a JDBC synchronization that was set earlier.

```
public String beginDownloadConnection ( Timestamp ts,
String user )
    throws java.sql.SQLException
{   prepDeleteTables ( _syncConn, ts, user );
    return ( null ); }
```

.NET example

The following stored procedure call registers a .NET method called `BeginDownload` as the script for the `begin_download` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script(
    'ver1',
    'begin_download',
    'TestScripts.Test.BeginDownload'
)
```

Following is the C# signature for the call `BeginDownload`.

```
public void BeginDownload(
    DateTime timestamp,
    string user )
```

begin_download table event

Function Provides a location to process statements related to a specific table just before preparing the download stream of inserts, updates, and deletions.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|---------------|
| 1 | last_download | TIMESTAMP |
| 2 | ml_username | VARCHAR(128) |
| 3 | table | VARCHAR (128) |

Default action None.

Description The MobiLink synchronization server executes this event as the first step in preparing download information for a specific table. The download information is prepared in its own transaction. The execution of this event is the first table-specific action in the transaction.

You can have one begin_download script for each table in the remote database. The script is only invoked when that table is synchronized.

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 user has never synchronized successfully, this value is set to 1900-01-01.

See also "end_download table event" on page 489

SQL example The following example can be used on an Adaptive Server Anywhere 8 database. The first chunk of code calls the ml_add_table_script, and the second creates a BeginTableDownload procedure.

```
call ml_add_table_script ( 'version1', 'Leads',
'begin_download', 'call BeginTableDownLoad( ?, ?, ? ) );
```

```
create procedure BeginTableDownload ( LastDownload
timestamp, MLUser varchar(128), TableName varchar(128) )
begin
    exexcute immediate 'update ' || TableName ||
' set last_download_check = CURRENT_TIMESTAMP
    where Owner = ' || MLUser;
end
```

Java example

The following stored procedure call registers a Java method called `beginDownloadTable` as the script for the `begin_download` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',
'begin_download',
'ExamplePackage.ExampleClass.beginDownloadTable' )
```

Following is the sample Java method `beginDownloadTable`. It saves the name of the current table for use in a later member function call.

```
public String beginDownloadTable( Timestamp ts,
String user, String table )
{ _curTable = table;
  return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called `BeginTableDownload` as the script for the `begin_download` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(
'ver1', 'table1', 'begin_download',
'TestScripts.Test.BeginTableDownload'
)
```

Following is the C# signature for the call `BeginTableDownload`.

```
public void BeginTableDownload(
    DateTime timestamp,
    string user,
    string table )
```

begin_download_deletes table event

Function Processes statements related to a specific table just before fetching a list of rows to be deleted from the specified table in the remote database.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|---------------|
| 1 | last_download | TIMESTAMP |
| 2 | ml_username | VARCHAR (128) |
| 3 | table | VARCHAR (128) |

Default action None.

Description This event is executed immediately before fetching a list of rows to be deleted from the named table in the remote database.

You can have one begin_download_deletes script for each table in the remote database.

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 user has never synchronized successfully, this value is set to 1900-01-01.

See also "begin_download_rows table event" on page 460
"end_download_rows table event" on page 493

SQL example To minimize the amount of data on remotes, you can use this event to flag data that will be deleted when the download_delete_cursor is executed. The following example flags for deletion sales leads from the remote device that are over 10 weeks old. The example can be used on an Adaptive Server Anywhere 8 database. The code calls the ml_add_table_script, and then creates a beginDownloadDeletes procedure.

```
call ml_add_table_script ('version1', 'Leads',  
'begin_download_deletes',  
'call BeginDownloadDeletes (?, ?, ?)' );
```



```
create procedure BeginDownloadDeletes (LastDownload
timestamp, MLUser varchar(128), TableName varchar(128) )
begin
    execute immediate 'update ' || TableName ||
        ' set delete_flag = 1 where
        days(creation_time, CURRENT DATE) > 70 and Owner = '
        || MLUser;
end;
```

Java example

The following stored procedure call registers a Java method called `beginDownloadDeletes` as the script for the `begin_download_deletes` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script ( 'ver1', 'table1',
    'begin_download_deletes',
    'ExamplePackage.ExampleClass.beginDownloadDeletes' )
```

Following is the sample Java method `beginDownloadDeletes`. It saves the name of the current table for use in a later member function call.

```
public String beginDownloadDeletes( Timestamp ts,
String user, String table )
{
    _curTable = table;
    return( null ); } }
```

.NET example

The following stored procedure call registers a .NET method called `BeginDownloadDeletes` as the script for the `begin_download_deletes` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(
    'ver1', 'table1', 'begin_download_deletes',
    'TestScripts.Test.BeginDownloadDeletes'
)
```

Following is the C# signature for the call `BeginDownloadDeletes`.

```
public void BeginDownloadDeletes(
    DateTime timestamp,
    string user,
    string table )
```

begin_download_rows table event

Function Processes statements related to a specific table just before fetching a list of rows to be inserted or updated in the specified table in the remote database.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|---------------|
| 1 | last_download | TIMESTAMP |
| 2 | ml_username | VARCHAR (128) |
| 3 | table | VARCHAR (128) |

Default action None.

Description This event is executed immediately before fetching the stream of rows to be inserted or updated in the named table in the remote database.

You can have one begin_download_rows script for each table in the remote database.

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 user has never synchronized successfully, this value is set to 1900-01-01.

See also "begin_download_deletes table event" on page 458
"end_download_deletes table event" on page 491

SQL example You can use the begin_download_rows table event to flag rows that you no longer want to download for this table. The following example archives sales leads that are over seven days old.

```
call ml_add_table_script ('version1', 'Leads',  
    'begin_download_rows',  
    'call BeginDownloadRows (?, ?, ?)' );
```

```
create procedure BeginDownloadRows (
    LastDownload timestamp, MLUser varchar(128),
    TableName varchar(128) )
begin
    execute immediate 'update ' || TableName ||
        ' set download_flag = 0 where
        days(creation_time, CURRENT DATE) > 7 and Owner = '
        || MLUser;
end;
```

Java example

The following stored procedure call registers a Java method called `beginDownloadRows` as the script for the `begin_download_rows` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',
    'begin_download_rows',
    'ExamplePackage.ExampleClass.beginDownloadRows' )
```

Following is the sample Java method `beginDownloadRows`. It generates an UPDATE statement using the table and user. MobiLink will execute this SQL statement.

```
public String beginDownloadRows( Timestamp ts,
    String user, String table )
{
    return( "update " + table + " set download_flag = 0 "
        + " where days(creation_time, CURRENT DATE) > 7 " +
        " and Owner = '" + user + "'" ); }
}
```

.NET example

The following stored procedure call registers a .NET method called `BeginDownloadRows` as the script for the `begin_download_rows` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(
    'ver1', 'table1', 'begin_download_rows',
    'TestScripts.Test.BeginDownloadRows'
)
```

Following is the C# signature for the call `BeginDownloadRows`.

```
public void BeginDownloadRows(
    DateTime timestamp,
    string user,
    string table )
```

begin_synchronization connection event

Function Processes any statements at the time an application connects to the MobiLink synchronization server in preparation for the synchronization process.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |

Default action None.

Description The MobiLink synchronization server executes this event immediately after an application preparing to synchronize has formed a connection with the MobiLink synchronization server.

This event is executed within a separate transaction before the upload transaction. It is useful for maintaining statistics.

See also "end_synchronization connection event" on page 495
"begin_synchronization table event" on page 464

SQL example You may want to store the ml_username value in a temporary table or variable if you will be referencing that value many times in subsequent scripts.

```
Call ml_add_connection_script ( 'version1',  
    'begin_synchronization', 'set @EmployeeID = ?' );
```

Java example The following stored procedure call registers a Java method called beginSynchronizationConnection as the script for the begin_synchronization connection event when synchronizing the script version ver1. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1', 'begin_synchronization',  
    'ExamplePackage.ExampleClass.beginSynchronizationConnection' )
```

Following is the sample Java method beginSynchronizationConnection. It saves the name of the synchronizing user for later use.

```
public String beginSynchronizationConnection(  
    String user )  
{  
    _curUser = user;  
    return( null ); }  
}
```

.NET example

The following stored procedure call registers a .NET method called `BeginSync` as the script for the `begin_synchronization` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script( 'ver1',  
    'begin_synchronization',  
    'TestScripts.Test.BeginSync'  
)
```

Following is the C# signature for the call `BeginSync`.

```
public void BeginSync( string user )
```

begin_synchronization table event

Function Processes statements related to a specific table at the time an application connects to the MobiLink synchronization server in preparation for the synchronization process.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|---------------|
| 1 | ml_username | VARCHAR (128) |
| 2 | table | VARCHAR (128) |

Default action None.

Description The MobiLink synchronization server executes this event after an application that is preparing to synchronize has formed a connection with the MobiLink synchronization server, and after the begin_synchronization connection-level event.

You can have one begin_synchronization script for each table in the remote database. The event is only invoked when the table is synchronized.

See also "end_synchronization table event" on page 497
"begin_synchronization connection event" on page 462

Java example The following stored procedure call registers a Java method called beginSynchronizationTable as the script for the begin_synchronization table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1', 'begin_synchronization',
'ExamplePackage.ExampleClass.beginSynchronizationTable' )
```

Following is the sample Java method beginSynchronizationTable. It adds the current table name to a list of table names contained in this instance.

```
public String beginSynchronizationTable(String user,
String table )
{
    _tableList.add( table );
    return( null );
}
```

.NET example

The following stored procedure call registers a .NET method called `BeginTableSync` as the script for the `begin_synchronization` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1',  
    'begin_synchronization',  
    'TestScripts.Test.BeginTableSync'  
)
```

Following is the C# signature for the call `BeginTableSync`.

```
public void BeginTableSync( string user, string table )
```

begin_upload connection event

Function Processes any statements just before the MobiLink synchronization server commences processing the stream of uploaded inserts, updates, and deletes.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

| Item | Parameter | Description |
|------|-------------|---------------|
| 1 | ml_username | VARCHAR (128) |

Default action None.

Description The MobiLink synchronization server executes this event as the first step in the processing of uploaded information. Upload information is processed in a single transaction. The execution of this event is the first action in this transaction.

See also "end_upload connection event" on page 499
"begin_upload table event" on page 468

Java example The following stored procedure call registers a Java method called beginUploadConnection as the script for the begin_upload connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',  
    'begin_upload',  
    'ExamplePackage.ExampleClass. beginUploadConnection ' )
```

Following is the sample Java method beginUploadConnection. It prints a message to the MobiLink output log. (This might be useful at development time but would slow down a production server.)

```
public String beginUploadConnection( String user )  
{ java.lang.System.out.println(  
    "Starting upload for user: " + user );  
    return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called BeginUpload as the script for the begin_upload connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.


```
call ml_add_dnet_connection_script( 'ver1',  
    'begin_upload',  
    'TestScripts.Test.BeginUpload'  
)
```

Following is the C# signature for the call `BeginUpload`.

```
public void BeginUpload( string user )
```

The following C# example saves the current user name for use in a later event.

```
public void BeginUpload( string curUser )  
{  
    user = curUser;  
}
```

begin_upload table event

Function Processes statements related to a specific table just before the MobiLink synchronization server commences processing the stream of uploaded inserts, updates, and deletes.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | table | VARCHAR(128) |

Default action None.

Description The MobiLink synchronization server executes this event as the first step in the processing of uploaded information. Upload information is processed in a separate transaction. The execution of this event is the first table-specific action in this transaction.

You can have one begin_upload script for each table in the remote database. The script is only invoked when the table is actually synchronized.

See also "end_upload table event" on page 502
"begin_upload connection event" on page 466

SQL example When uploading rows from a remote you may want to place the changes in an intermediate table and manually process changes yourself. You can populate a global temporary table in this event.

```
call ml_add_table_script ('version1', 'Leads' ,
    'begin_upload', 'insert into T_Leads SELECT *
    FROM Leads WHERE Owner = @EmployeeID')
```

Java example The following stored procedure call registers a Java method called beginUploadTable as the script for the begin_upload table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',
    'begin_upload',
    'ExamplePackage.ExampleClass.a beginUploadTable ' )
```

Following is the sample Java method `beginUploadTable`. This example takes no action. MobiLink interprets NULL as no script.

```
public String beginUploadTable( String user,
    String table )
{ return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called `BeginTableUpload` as the script for the `begin_upload` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(
    'ver1', 'table1', 'begin_upload',
    'TestScripts.Test.BeginTableUpload'
)
```

Following is the C# signature for the call `BeginTableUpload`.

```
public void BeginTableUpload(
    string user,
    string table )
```

begin_upload_deletes table event

Function Processes statements related to a specific table just before uploading deleted rows from the specified table in the remote database.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | table | VARCHAR(128) |

Default action None.

Description This event runs immediately before applying the changes that result from rows deleted in the client table named in the second parameter.

You can have one begin_upload_deletes script for each table in the remote database. The script is only invoked when the table is actually synchronized.

See also "end_upload_deletes table event" on page 504

Java example The following stored procedure call registers a Java method called beginUploadDeletes as the script for the begin_upload_deletes table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',
    'begin_upload_deletes',
    'ExamplePackage.ExampleClass. beginUploadDeletes' )
```

Following is the sample Java method beginUploadDeletes. It prints a message to the MobiLink output log. (This might be useful at development time but would slow down a production server.)

```
public String beginUploadDeletes(String user,
String table )
    throws java.sql.SQLException
{   java.lang.System.out.println( "Starting upload
    deleted for table: " + table );
    return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called `BeginUploadDeletes` as the script for the `begin_upload_deletes` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script( 'ver1', 'table1',  
    'begin_upload_deletes',  
    'TestScripts.Test.BeginUploadDeletes'  
)
```

Following is the C# signature for the call `BeginUploadDeletes`.

```
public void BeginUploadDeletes( string user,  
    string table )
```

begin_upload_rows table event

Function Processes statements related to a specific table just before uploading inserts and updates from the specified table in the remote database.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | table | VARCHAR(128) |

Default action None.

Description This event is run immediately prior to applying the changes that result from inserts and deletes to the client table named in the second parameter.

You can have one begin_upload_rows script for each table in the remote database. The script is only invoked when the table is actually synchronized.

See also "end_upload_rows table event" on page 506

Java example The following stored procedure call registers a Java method called beginUploadRows as the script for the begin_upload_rows table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',
'begin_upload_rows',
'ExamplePackage.ExampleClass.beginUploadRows' )
```

Following is the sample Java method beginUploadRows. It prints a message to the MobiLink output log. (This might be useful at development time but would slow down a production server.)

```
public String beginUploadRows ( String user,
String table )
throws java.sql.SQLException
{ java.lang.System.out.println( "Starting upload rows
for table: " + table + " and user: " + user );
return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called `BeginUploadRows` as the script for the `begin_upload_rows` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1', 'begin_upload_rows',  
    'TestScripts.Test.BeginUploadRows'  
)
```

Following is the C# signature for the call `BeginUploadRows`.

```
public void BeginUploadRows(  
    string user,  
    string table )
```

download_cursor cursor event

Function Defines a cursor to select rows that are to be downloaded and inserted or updated in the remote database.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|--------------|
| 1 | last_download | TIMESTAMP |
| 2 | ml_username | VARCHAR(128) |

Default action None.

A default download_cursor SQL script can be generated using the MobiLink synchronization server -za option. Also, the UltraLite analyzer generates a SELECT statement based on your reference database that you can use to get started.

Description The MobiLink synchronization server opens a read-only cursor with which to fetch a list of rows to download to the remote database. This script should contain a suitable SELECT statement.

The parameters are the last_download timestamp and the user name. You can use these values if you choose by placing question marks in your SQL statement.

You can have one download_cursor script for each table in the remote database.

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 user has never synchronized successfully, this value is set to 1900-01-01.

To optimize performance of the download stage of synchronization to UltraLite clients, when the range of primary key values is outside the current rows on the device, you should order the rows in the download cursor by primary key. Downloads of large reference tables, for example, can benefit from this optimization.

Note that `download_cursor` allows for cascading deletes. Thus, you can delete records from a database.

For Java and .NET applications, this script must return valid SQL.

See also

"upload_cursor cursor event" on page 543

"download_delete_cursor cursor event" on page 477

SQL example

The following example comes from an Oracle installation, although the statement is valid against all supported databases. The example downloads all rows that have been changed since the last time the user downloaded data, and which match the user name in the *emp_name* column.

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

To write a `download_cursor` SQL script that does not use the first parameter (the `last_download` timestamp), but does use the second parameter (the MobiLink user name), add a dummy clause that affects no rows. For example:

```
call ml_add_table_script( 'Lab', 'ULOrder',
    'download_cursor', 'SELECT order_id, cust_id,
    prod_id, emp_id, disc, quant, notes, status
    FROM ULOrder WHERE ? IS NOT NULL AND emp_name = ?' )
```

You must still use both parameters, but the first `?` is a place holder that does nothing.

Java example

The following stored procedure call registers a Java method called `downloadCursor` as the script for the `download_cursor` cursor event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',
    'download_cursor',
    'ExamplePackage.ExampleClass. downloadCursor ' )
```

Following is the sample Java method `downloadCursor`. It dynamically creates the SQL statement for the download cursor.

```
public String downloadCursor( Timestamp ts,
    String user )
{ return( "SELECT order_id, cust_id, prod_id, emp_id,
    disc, " + " quant, notes, status " + "FROM ULOrder " +
    "WHERE emp_name = '" + user + "'" ); }
```

.NET example

The following stored procedure call registers a .NET method called DownloadCursor as the script for the download_cursor cursor event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1', 'download_cursor',  
    'TestScripts.Test.DownloadCursor'  
)
```

Following is the C# signature for the call DownloadCursor.

```
public string DownloadCursor(  
    DateTime timestamp,  
    string user )
```

The following C# example populates a temporary table with the contents of a file called *rows.txt*. It then returns a cursor that causes MobiLink to send the rows in the temporary table to the remote database.

```
public string DownloadCursor( DateTime ts, string user )  
{  
    DBCommand stmt = curConn.CreateCommand();  
    StreamReader input = new StreamReader( "rows.txt" );  
    string sql = input.ReadLine();  
  
    stmt.CommandText = "DELETE FROM dnet_dl_temp";  
    stmt.ExecuteNonQuery();  
  
    while( sql != null ){  
        stmt.CommandText = "INSERT INTO dnet_dl_temp VALUES "  
+ sql;  
        stmt.ExecuteNonQuery();  
        sql = input.ReadLine();  
    }  
    return( "SELECT * FROM dnet_dl_temp" );  
}
```

download_delete_cursor cursor event

Function Defines a cursor to select rows that are to be deleted in the remote database.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|--------------|
| 1 | last_download | TIMESTAMP |
| 2 | ml_username | VARCHAR(128) |

Default action None.

Description The MobiLink synchronization server opens a read-only cursor with which to fetch a list of rows to download, and then insert or update in the remote database. This script must contain a SELECT statement that returns the primary key values of the rows to be deleted from the table in the remote database.

The parameters are the last_download timestamp and the user name. You can use these values by placing a question mark in your SQL statement.

You can have one download_delete_cursor script for each table in the remote database.

If the download_delete_cursor has NULLs for the primary key columns for one or more rows in a table, then MobiLink deletes all the data in the table. For a complete description of this behavior, see "Deleting all the rows in a table" on page 73.

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 user has never synchronized successfully, this value is set to 1900-01-01.

Note that rows deleted from the consolidated database will not appear in a result set defined by a download_delete_cursor event, and so are not automatically deleted from the remote database. One technique for identifying rows to be deleted from remote databases is to add a column to the consolidated database table identifying a row as inactive.

For Java and .NET applications, this script must return valid SQL.

See also

"upload_cursor cursor event" on page 543
"download_cursor cursor event" on page 474

SQL example

This example is taken from the Contact sample and can be found in *Samples\MobiLink\Contact\build_consol.sql*. It deletes from the remote database any customers that:

- ◆ have been changed since the last time this user downloaded data (Customer.last_modified > ?), and either
- ◆ do not belong to the synchronizing user (SalesRep.ml_username != ?), or
- ◆ are marked as inactive in the consolidated database (Customer.active = 0).

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

Java example

The following stored procedure call registers a Java method called downloadDeleteCursor as the script for the download_delete_cursor event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',
'download_delete_cursor',
'ExamplePackage.ExampleClass.downloadDeleteCursor' )
```

Following is the sample Java method downloadDeleteCursor. It calls a Java method that generates the SQL for the download delete cursor.

```
public String downloadDeleteCursor( Timestamp ts,
String user )
{ return( getDownloadCursor( _curUser, _curTable ) ); }
```

.NET example

The following stored procedure call registers a .NET method called DownloadDeleteCursor as the script for the download_delete_cursor cursor event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(
'ver1', 'table1', 'download_delete_cursor',
'TestScripts.Test.DownloadDeleteCursor'
)
```

Following is the C# signature for the call DownloadDeleteCursor.

```
public string DownloadDeleteCursor(
    DateTime timestamp,
    string user )
```

download_statistics connection event

Function Tracks synchronization statistics for download operations.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|--|
| 1 | ml_username | VARCHAR(128). The MobiLink user name as specified in your SYNCHRONIZATION USER definition. |
| 2 | warnings | INTEGER. The number of warnings issued. |
| 3 | errors | INTEGER. The number of errors, including handled errors, that occurred. |
| 4 | fetches_rows | INTEGER. The number of rows fetched by the download_cursor script. |
| 5 | deleted_rows | INTEGER. The number of rows fetched by the download_deletes script. |
| 6 | filtered_rows | INTEGER. The number of rows from (5) actually sent to the remote. This reflects download filtering of uploaded values. |
| 7 | bytes | INTEGER. The number of bytes sent to the remote as the download. |

Default action None.

Description The download_statistics event allows you to gather, for any user, statistics on downloads. The download_statistics connection script is called just prior to the commit at the end of the download transaction.

Note:

Depending on the command line, not all warnings or errors are logged, so the warnings and errors counts may be more than the number of warnings or errors logged.

See also

"download_statistics table event" on page 482
"upload_statistics connection event" on page 554
"upload_statistics table event" on page 557
"synchronization_statistics connection event" on page 535
"synchronization_statistics table event" on page 537
"time_statistics connection event" on page 539
"time_statistics table event" on page 541
"MobiLink Monitor" on page 231

SQL example

The following example comes from an Oracle installation.

```
INSERT INTO download_audit (id, user_name, warnings,
errors, deleted_rows, fetched_rows, download_rows,
bytes) VALUES (d_audit.nextval, ?,?,?,?,?,,?)
```

Once vital statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

Java example

The following stored procedure call registers a Java method called `downloadStatisticsConnection` as the script for the `download_statistics` event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1', 'download_statistics',
'ExamplePackage.ExampleClass.downloadStatisticsConnection' )
```

Following is the sample Java method `downloadStatisticsConnection`. It prints the number of fetched rows to the MobiLink output log.

```
public String downloadStatisticsConnection( String user,
int warnings, int errors, int fetchedRows,
int deletedRows, int bytes )
{  java.lang.System.out.println( "download connection
stats fetchedRows: " + fetchedRows );
return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called `DownloadStats` as the script for the `download_statistics` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script(  
    'ver1',  
    'download_statistics',  
    'TestScripts.Test.DownloadStats'  
)
```

Following is the C# signature for the call DownloadStats.

```
public void DownloadStats(  
    string user,  
    int warnings,  
    int errors,  
    int deletedRows,  
    int fetchedRows,  
    int downloadRows,  
    int bytes )
```

download_statistics table event

Function Tracks synchronization statistics for download operations by table.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|--|
| 1 | ml_username | VARCHAR(128). This is the MobiLink user name as specified in your SYNCHRONIZATION USER definition. |
| 2 | table | VARCHAR(128). The table name. |
| 3 | warnings | INTEGER. The number of warnings issued. |
| 4 | errors | INTEGER. The number of errors, including handled errors, that occurred. |
| 5 | fetched_rows | INTEGER. The number of rows fetched by the download_cursor script. |
| 6 | deleted_rows | INTEGER. The number of rows fetched by the download_deletes script. |
| 7 | filtered_rows | INTEGER. The number of rows from (6) actually sent to the remote. This reflects download filtering of uploaded values. |
| 8 | bytes | INTEGER. The number of bytes sent to the remote as the download. |

Default action None.

Description The download_statistics event allows you to gather, for any user and table, statistics on downloads as they apply to that table. The download_statistics table script is called just prior to the commit at the end of the download transaction.

See also "download_statistics connection event" on page 479
"upload_statistics connection event" on page 554
"upload_statistics table event" on page 557
"synchronization_statistics connection event" on page 535
"synchronization_statistics table event" on page 537
"time_statistics connection event" on page 539

"time_statistics table event" on page 541

"MobiLink Monitor" on page 231

SQL example

The following example comes from an Oracle installation.

```
INSERT INTO download_audit ( id, user_name, table,
warnings, errors, deleted_rows, fetched_rows,
download_rows, bytes)
VALUES (d_audit.nextval,?,?,?,?,?,?,?,?)
```

Once vital statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

Java example

The following stored procedure call registers a Java method called `downloadStatisticsTable` as the script for the `download_statistics` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',
'download_statistics',
'ExamplePackage.ExampleClass.downloadStatisticsTable' )
```

Following is the sample Java method `downloadStatisticsTable`. It prints some statistics for this table to the MobiLink output log.

```
public String downloadStatisticsTable( String user,
String table, int warnings, int errors, int fetchedRows,
int deletedRows, int bytes )
{ java.lang.System.out.println( "download table stats "
+ "table: " + table + "bytes: " + bytes );
return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called `DownloadTableStats` as the script for the `download_statistics` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(
'ver1', 'table1', 'download_statistics',
'TestScripts.Test.DownloadTableStats'
)
```

Following is the C# signature for the call `DownloadTableStats`.

```
public void DownloadTableStats(  
    string user,  
    string table,  
    int warnings,  
    int errors,  
    int deletedRows,  
    int fetchedRows,  
    int downloadRows,  
    int bytes )
```

end_connection connection event

| | |
|-----------------------|--|
| Function | Processes any statements just before the MobiLink synchronization server closes a connection with the consolidated database server, either in preparation to shut down or when a connection is removed from the connection pool. This script is normally used to complete any actions started by the begin_connection script and free any resources acquired by it. |
| Parameters | None. |
| Default action | None. |
| Description | You can use the end_connection script to perform an action of your choice just prior to closing of a connection between the MobiLink synchronization server and the consolidated database server. |
| See also | "begin_connection connection event" on page 452 |
| Java example | The following stored procedure call registers a Java method called endConnection as the script for the end_connection event when synchronizing the script version <i>ver1</i> . This syntax is for Adaptive Server Anywhere consolidated databases. |

```
call ml_add_java_connection_script( 'ver1',  
    'end_connection',  
    'ExamplePackage.ExampleClass.endConnection' )
```

Following is the sample Java method endConnection. It prints a message to the MobiLink output log. (This might be useful at development time but would slow down a production server.)

```
public String endConnection()  
{ java.lang.System.out.println( "ending connection" );  
  return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called EndConnection as the script for the end_connection connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script(  
    'ver1',  
    'end_connection',  
    'TestScripts.Test.EndConnection'  
)
```

Following is the C# signature for the call EndConnection.

```
public void EndConnection()
```

end_download connection event

Function Processes any statements just after the MobiLink synchronization server concludes preparation of the download data.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|--------------|
| 1 | last_download | TIMESTAMP |
| 2 | ml_username | VARCHAR(128) |

Default action None.

Description The MobiLink synchronization server executes this script after all rows have been downloaded and, if expecting a download acknowledgement, confirmation of receipt has been received. Download information is processed in a single transaction. The execution of this script is the last non statistical action in this transaction.

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 user has never synchronized successfully, this value is set to 1900-01-01.

See also "begin_download connection event" on page 454

SQL example The following example shows one possible use of an end_download connection script.

```
DELETE FROM ULEmpCust ec WHERE ? IS NOT NULL AND
ec.emp_id = ? AND action = 'D'
```

Java example The following stored procedure call registers a Java method called endDownloadConnection as the script for the end_download connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',
'end_download',
'ExamplePackage.ExampleClass.endDownloadConnection' )
```

Following is the sample Java method `endDownloadConnection`. It uses the current MobiLink connection (saved earlier) to perform an update before the download ends.

```
public String endDownloadConnection( Timestamp ts,
String user )
throws java.sql.SQLException
{   String del_sql =      "DELETE FROM ULEmpCust ec " +
    "WHERE ec.emp_id = '" + user + "' " +
    "AND action = 'D' ";
    execUpdate( _syncConn, del_sql );
    return( null );
}
```

.NET example

The following stored procedure call registers a .NET method called `EndDownload` as the script for the `end_download` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script(
    'ver1',
    'end_download',
    'TestScripts.Test.EndDownload' )
```

Following is the C# signature for the call `EndDownload`.

```
public void EndDownload(
    DateTime timestamp,
    string user )
```

end_download table event

Function Processes statements related to a specific table just after the MobiLink synchronization server concludes preparing the stream of downloaded inserts, updates, and deletes.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|--------------|
| 1 | last_download | TIMESTAMP |
| 2 | ml_username | VARCHAR(128) |
| 3 | table | VARCHAR(128) |

Default action None.

Description The MobiLink synchronization server executes this script after all rows have been downloaded and confirmation of receipt has been received. The download information is prepared in a separate transaction. The execution of this script is the last table-specific, non-statistical action in this transaction.

You can have one end_download script for each table in the remote database.

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 user has never synchronized successfully, this value is set to 1900-01-01.

See also "begin_download table event" on page 456
"end_download connection event" on page 487

Java example The following stored procedure call registers a Java method called endDownloadTable as the script for the end_download table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script ( 'ver1', 'table1',
    'end_download',
    'ExamplePackage.ExampleClass.endDownloadTable' )
```

Following is the sample Java method endDownloadTable. It resets the current table member variable.

```
public String endDownloadTable( Timestamp ts,
String user, String table )
{
    _curTable = null;
    return( null );
}
```

.NET example

The following stored procedure call registers a .NET method called EndTableDownload as the script for the end_download table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(
    'ver1', 'table1', 'end_download',
    'TestScripts.Test.EndTableDownload'
)
```

Following is the C# signature for the call EndTableDownload.

```
public void EndTableDownload
    DateTime timestamp,
    string user,
    string table )
```


end_download_deletes table event

- Function** Processes statements related to a specific table just after preparing a list of rows to be deleted from the specified table in the remote database.
- Parameters** In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.
- Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|--------------|
| 1 | last_download | TIMESTAMP |
| 2 | ml_username | VARCHAR(128) |
| 3 | table | VARCHAR(128) |

Default action None.

Description This script is executed immediately after preparing a list of rows to be deleted from the named table in the remote database.

You can have one end_download_deletes script for each table in the remote database.

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 user has never synchronized successfully, this value is set to 1900-01-01.

See also "begin_download_deletes table event" on page 458
 "end_download connection event" on page 487
 "begin_download_rows table event" on page 460
 "end_download_rows table event" on page 493

SQL example You may want to mark a row as deleted on the remote database in this event, using a WHERE clause on the UPDATE that matches the WHERE clause used for your download_delete_cursor.

```
Call ml_add_table_script('version1', 'Leads',
  'end_download_deletes', 'UPDATE Leads SET OnRemote = 0
  WHERE LastModified > ? AND Owner = ? AND DeleteFlag=1');
```

Java example

The following stored procedure call registers a Java method called `endDownloadDeletes` as the script for the `end_download_deletes` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',  
    'end_download_deletes',  
    'ExamplePackage.ExampleClass.endDownloadDeletes' )
```

Following is the sample Java method `endDownloadDeletes`. It returns the `end_download_deletes` SQL statement. MobiLink will execute this statement.

```
public String endDownloadDeletes( Timestamp ts,  
    String user, String table )  
{ return( "UPDATE Leads SET OnRemote = 0 WHERE  
    LastModified > ? AND Owner = ? AND DeleteFlag=1" ); }
```

.NET example

The following stored procedure call registers a .NET method called `EndDownloadDeletes` as the script for the `end_download_deletes` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1', 'end_download_deletes',  
    'TestScripts.Test.EndDownloadDeletes'  
)
```

Following is the C# signature for the call `EndDownloadDeletes`.

```
public void EndDownloadDeletes( DateTime timestamp,  
    string user, string table )
```

end_download_rows table event

- Function** Processes statements related to a specific table just after preparing a list of rows to be inserted or updated in the specified table in the remote database.
- Parameters** In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.
- Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|--------------|
| 1 | last_download | TIMESTAMP |
| 2 | ml_username | VARCHAR(128) |
| 3 | table | VARCHAR(128) |

Default action None.

Description This script is executed immediately after preparing the stream of rows to be inserted or updated in the named table in the remote database.

You can have one end_download_rows script for each table in the remote database.

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 user has never synchronized successfully, this value is set to 1900-01-01.

See also "begin_download_rows table event" on page 460
 "end_download connection event" on page 487
 "end_download_deletes table event" on page 491
 "begin_download_deletes table event" on page 458

SQL example You may want to mark a row as successfully downloaded to the remote database in this event, using a WHERE clause on the UPDATE that matches the WHERE clause used for your download_cursor.

```
call ml_add_table_script('version1', 'Leads',
  'end_download_rows', 'UPDATE Leads SET OnRemote = 1
  WHERE LastModified > ? AND Owner = ? AND
  DownloadFlag=1');
```

Java example

The following stored procedure call registers a Java method called `endDownloadRows` as the script for the `end_download_rows` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',
    'end_download_rows',
    'ExamplePackage.ExampleClass.endDownloadRows' )
```

Following is the sample Java method `endDownloadRows`. It prints a message to the MobiLink output log. (This might be useful at development time but would slow down a production server.)

```
public String endDownloadRows( Timestamp ts,
    String user, String table )
{
    java.lang.System.out.println( "Done downloading
        inserts and updates for table " + table );
    return( null ); }
}
```

.NET example

The following stored procedure call registers a .NET method called `EndDownloadRows` as the script for the `end_download_rows` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(
    'ver1', 'table1', 'end_download_rows',
    'TestScripts.Test.EndDownloadRows'
)
```

Following is the C# signature for the call `EndDownloadRows`.

```
public void EndDownloadRows(
    DateTime timestamp,
    string user,
    string table )
```

end_synchronization connection event

Function Processes any statements at the time an application disconnects from the MobiLink synchronization server upon completion of the synchronization process.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |

Default action None.

Description The MobiLink synchronization server executes this script after synchronization is complete and, if expecting a download acknowledgement, the MobiLink client has returned confirmation of receipt of the download stream.

This script is executed within a separate transaction after the download transaction. It is useful for maintaining statistics.

See also "begin_synchronization connection event" on page 462
 "begin_synchronization table event" on page 464
 "end_synchronization table event" on page 497

Java example The following stored procedure call registers a Java method called endSynchronizationConnection as the script for the end_synchronization event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1', 'end_synchronization',
'ExamplePackage.ExampleClass.endSynchronizationConnection' )
```

Following is the content of the sample Java method endSynchronizationConnection. It uses the JDBC connection to execute an update.

```
public String endSynchronizationConnection(
String user )
throws java.sql.SQLException
{
    execUpdate( _syncConn, "UPDATE sync_count set cnt =
count + 1 where user_id = '" + user + "' " );
    return( null );
}
```

.NET example

The following stored procedure call registers a .NET method called EndSync as the script for the end_synchronization connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script(  
    'ver1',  
    'end_synchronization',  
    'TestScripts.Test.EndSync'  
)
```

Following is the C# signature for the call EndSync.

```
public void EndSync( string user )
```

end_synchronization table event

- Function** Processes statements related to a specific table at the time an application disconnects from the MobiLink synchronization server upon completion of the synchronization process.
- Parameters** In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.
- Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | table | VARCHAR(128) |

- Default action** None.
- Description** The MobiLink synchronization server executes this script after an application has synchronized and is about to disconnect from the MobiLink synchronization server, and before the connection level script of the same name.
- You can have one end_synchronization script for each table in the remote database.

- See also** "begin_synchronization table event" on page 464
 "end_synchronization connection event" on page 495
 "end_synchronization table event" on page 497

- Java example** The following stored procedure call registers a Java method called endSynchronizationTable as the script for the end_synchronization table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',
    'end_synchronization',
    'ExamplePackage.ExampleClass.endSynchronizationTable' )
```

Following is the sample Java method endSynchronizationTable. It takes no action. MobiLink interprets NULL as no script.

```
public String endSynchronizationTable( String user,
    String table )
{ return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called EndTableSync as the script for the end_synchronization table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1', 'end_synchronization',  
    'TestScripts.Test.EndTableSync'  
)
```

Following is the C# signature for the call EndTableSync.

```
public void EndTableSync( string user, string table )
```


end_upload connection event

Function Processes any statements just after the MobiLink synchronization server concludes processing uploaded inserts, updates, and deletes.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |

Default action None.

Description The MobiLink synchronization server executes this script as the last step in the processing of uploaded information. Upload information is processed in a single transaction. The execution of this script is the last action in this transaction before statistical scripts.

See also "begin_upload connection event" on page 466
"end_upload table event" on page 502

SQL example The following statements define a stored procedure and an end_upload script suited to the CustDB sample application from an Oracle installation.

```
CREATE OR REPLACE PROCEDURE ULCustomerIDPool_maintain(
  SyncUserID IN integer )
AS
  pool_count INTEGER;
  pool_max    INTEGER;
BEGIN
  -- Determine how many ids to add to the pool
  SELECT COUNT(*)
    INTO pool_count
    FROM ULCustomerIDPool
    WHERE pool_emp_id = SyncUserID;
  -- Determine the current Customer id max
  SELECT MAX(pool_cust_id)
    INTO pool_max
    FROM ULCustomerIDPool;
  -- Top up the pool with new ids
  WHILE pool_count < 20 LOOP
    pool_max := pool_max + 1;
    INSERT INTO ULCustomerIDPool ( pool_cust_id,
pool_emp_id ) VALUES ( pool_max, SyncUserID );
    pool_count := pool_count + 1;
  END LOOP;
END;
ml_add_table_script( 'custdb', 'ULCustomerIDPool',
'end_upload', -
ULCustomerIDPool_maintain( ? );)
```

Java example

The following stored procedure call registers a Java method called `endUploadConnection` as the script for the `end_upload` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',
'end_upload',
'ExamplePackage.ExampleClass.endUploadConnection' )
```

Following is the sample Java method `endUploadConnection`. It calls a method to perform operations on the database.

```
public String endUploadConnection( String user )
{ // clean up new and old tables
  Iterator two_iter = _tables_with_ops.iterator();
  while( two_iter.hasNext() )
  { TableInfo cur_table = (TableInfo)two_iter.next();
    dumpTableOps( _sync_conn, cur_table ); }
  _tables_with_ops.clear(); }
```

.NET example

The following stored procedure call registers a .NET method called `EndUpload` as the script for the `end_upload` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script( 'ver1',  
    'end_upload',  
    'TestScripts.Test.EndUpload'  
)
```

Following is the C# signature for the call EndUpload.

```
public void EndUpload( string user )
```

end_upload table event

Function Processes statements related to a specific table just after the MobiLink synchronization server concludes processing the stream of uploaded inserts, updates, and deletions.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | table | VARCHAR(128) |

Default action None.

Description The MobiLink synchronization server executes this script as the last step in the processing of uploaded information. Upload information is processed in a separate transaction. The execution of this script is the last table-specific action in this transaction.

You can have one end_upload script for each table in the remote database.

See also "begin_upload table event" on page 468
"end_upload connection event" on page 499

SQL example The event can be used to clean up anything that may still exist in the database after the upload processing is done for a particular table.

```
Call ml_add_table_script ('version1', 'Leads',  
    'end_upload', 'DELETE FROM T_Leads');
```

Java example The following stored procedure call registers a Java method called endUploadTable as the script for the end_upload table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',  
    'end_upload',  
    'ExamplePackage.ExampleClass.endUploadTable' )
```

Following is the sample Java method endUploadTable. It generates a delete for a table with a name related to the passing-in table name.

```
public String endUploadTable( String user,
String table)
{ return( "DELETE from '" + table + "_temp'" ); }
```

.NET example

The following stored procedure call registers a .NET method called EndTableUpload as the script for the end_upload table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(
    'ver1', 'table1', 'end_upload',
    'TestScripts.Test.EndTableUpload'
)
```

Following is the C# signature for the call EndTableUpload.

```
public void EndTableUpload( string user, string table )
```

The following C# example moves rows inserted into a temporary table into the table passed into the script.

```
public void EndUpload( string user, string table )
{
    DBCommand stmt = curConn.CreateCommand();

    // move the uploaded rows to the destination table
    stmt.CommandText = "INSERT INTO "
        + table
        + " SELECT * FROM dnet_ul_temp";
    stmt.ExecuteNonQuery();
    stmt.Close();
}
```

end_upload_deletes table event

Function Processes statements related to a specific table just after applying deletes uploaded from the specified table in the remote database.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | table | VARCHAR(128) |

Default action None.

Description This script is run immediately after applying the changes that result from rows deleted in the remote table named in the second parameter.

You can have one end_upload_deletes script for each table in the remote database.

See also "begin_upload_deletes table event" on page 470

SQL example You can use this event to process rows deleted during the upload stream on an intermediate table. You can compare the rows in the base table with rows in the intermediate table and decide what to do with the deleted row.

```
Call ml_add_table_script('version1', 'Leads',
    'end_uploads_deletes', 'call EndUploadDeletesLeads()');
Create procedure EndUploadDeletesLeads ( )
Begin
    FOR names AS curs CURSOR FOR
        SELECT LeadID FROM Leads WHERE LeadID NOT IN (SELECT
        LeadID FROM T_Leads);
    DO
        CALL decide_what_to_do( LeadID );
    END FOR;
end
```

Java example The following stored procedure call registers a Java method called endUploadDeletes as the script for the end_upload_deletes table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',  
    'end_upload_deletes',  
    'ExamplePackage.ExampleClass.endUploadDeletes' )
```

Following is the sample Java method `endUploadDeletes`. It calls a Java method that manipulates the database.

```
public String endUploadDeletes( String user,  
    String table )  
    throws java.sql.SQLException  
{ processUploadedDeletes( _syncConn, table );  
    return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called `EndUploadDeletes` as the script for the `end_upload_deletes` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1', 'end_upload_deletes',  
    'TestScripts.Test.EndUploadDeletes'  
    )
```

Following is the C# signature for the call `EndUploadDeletes`.

```
public void EndUploadDeletes( string user, string table  
    )
```

end_upload_rows table event

Function Processes statements related to a specific table just after applying uploaded inserts and updates from the specified table in the remote database.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | table | VARCHAR(128) |

Default action None.

Description Uploaded information can require inserting or updating rows in the consolidated database. This script is run immediately after applying the changes that result from modifications to the remote table named in the second parameter.

You can have one end_upload_rows script for each table in the remote database.

See also "begin_upload_rows table event" on page 472

SQL example You use this event to process rows deleted during the upload stream on an intermediate table. You can compare the rows in the base table with the rows in the intermediate table and decide what to do with the deleted row.

```
Call ml_add_table_script('version1', 'Leads',
'end_uploads_deletes', 'call EndUploadDeletesLeads()');
Create procedure EndUploadDeletesLeads ( )
Begin
  FOR names AS curs CURSOR FOR
    SELECT LeadID FROM Leads WHERE LeadID NOT IN (select
LeadID from T_Leads);
  DO
    CALL decide_what_to_do( LeadID );
  END FOR;
end
```


Java example

The following stored procedure call registers a Java method called `endUploadRows` as the script for the `end_upload_rows` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',
                              'end_upload_rows',
                              'ExamplePackage.ExampleClass.endUploadRows' )
```

Following is the sample Java method `endUploadRows`. It calls a Java method that manipulates the database.

```
public String endUploadRows(    String user,
                              String table )
    throws java.sql.SQLException
{    processUploadedRows( _syncConn, table );
    return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called `EndUploadRows` as the script for the `end_upload_rows` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script( 'ver1', 'table1',
                              'end_upload_rows',
                              'TestScripts.Test.EndUploadRows'
                              )
```

Following is the C# signature for the call `EndUploadRows`.

```
public void EndUploadRows(
    string user,
    string table )
```

example_upload_cursor table event

Function Provides an event that the MobiLink synchronization server does not use during processing of the upload stream to handle rows inserted into the remote database. The event is not called.

| Parameters | Item | Parameter |
|------------|------|-----------|
| | 1 | column 1 |
| | 2 | column 2 |
| | ... | ... |

Description The statement based example_upload_cursor script performs direct inserts of column values identical to those specified in the example_upload_cursor statement. The example_upload_cursor event is not called by MobiLink.

SQL example The script is not called. If it were called, it would insert the values into a table named *Customer* in the consolidated database. The final column of the table identifies the Customer as active. The final column does not appear in the remote database.

```
SELECT cust_id, name, rep_id
FROM customer
WHERE cust_id=?
```

example_upload_delete table event

Function Processes the upload stream to handle rows deleted from the remote database. The script is not called by MobiLink.

Parameters

| Item | Parameter |
|------|-----------|
| 1 | column 1 |
| 2 | column 2 |
| ... | ... |

Description The statement based example_upload_delete script handles rows that are deleted in the remote database. The action taken at the consolidated database can be a DELETE statement, but need not be.

See also "upload_delete table event" on page 545

SQL example This example marks customers that are deleted from the remote database as inactive.

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

example_upload_insert table event

Function Provides an event that the MobiLink synchronization server uses during processing of the upload stream to handle rows inserted into the remote database.

| Parameters | Item | Parameter |
|------------|------|-----------|
| | 1 | column 1 |
| | 2 | column 2 |
| | ... | ... |

Description The statement based example_upload_insert script performs direct inserts of column values identical to those specified in the upload_insert statement.

The example_upload_insert event is not called.

See also "upload_insert table event" on page 549

SQL example The script is not called. But if called, it would insert the values into a table named *Customer* in the consolidated database. The final column of the table identifies the Customer as active. The final column does not appear in the remote database.

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

example_upload_update table event

Function An example event for the upload stream to handle rows updated at the remote database. The example script is not called by MobiLink but is identical in form to the upload_update event.

| Parameters | Clause | Parameters |
|------------|--------|---------------------------------------|
| | SET | column 1 column 2 ... |
| | WHERE | primary key 1 primary key 2 ... |

Description You create an example_upload_update event script by using the option -za in the dbmlsync command line.

See also "upload_update table event" on page 560

SQL example This example handles updates made to the *Customer* table in the remote database. The script updates the values in a table named *Customer* in the consolidated database. Note: The script is never called and is only an example script.

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

handle_error connection event

Function Executed whenever the MobiLink synchronization server encounters a SQL error.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|---|
| 1 | action_code | INTEGER. This is an INOUT parameter. |
| 2 | error_code | INTEGER |
| 3 | error_message | TEXT |
| 4 | ml_username | VARCHAR(128) |
| 5 | table | VARCHAR(128). If the script is not a table script, the table name is NULL. |

Default action When no handle_error script is defined or this script causes an error, the default action code is 3000: rollback the current transaction and cancel the current synchronization.

Description The MobiLink synchronization server sends in the current action_code. Initially, this is set to 3000 for each set of errors caused by a single SQL operation. Usually, there is only one error per SQL operation, but there may be more. This handle_error script is called once per error in the set. The action code passed into the first error is 3000. Subsequent calls are passed in the action code returned by the previous call. MobiLink will use the numerically highest value returned from multiple calls.

You can modify the action code in the script, and return a value instructing MobiLink how to proceed. The action code parameter takes one of the following values:

- ◆ **1000** Skip the current row and continue processing.

- ◆ **3000** Rollback the current transaction and cancel the current synchronization. This is the default action code, and is used when no `handle_error` script is defined or this script causes an error.
- ◆ **4000** Rollback the current transaction, cancel the synchronization, and shut down the MobiLink synchronization server.

SQL scripts for the `handle_error` event must be implemented as stored procedures.

The `action_code` parameter is an INOUT parameter: a parameter that provides a value to the procedure, and could be given a new value by the procedure.

The MobiLink synchronization server executes this script whenever it encounters an error during the synchronization process. The error codes and message allow you to identify the nature of the error. If the error happened as part of synchronization, the user name is supplied. Otherwise, this value is NULL.

If the error happened while manipulating a particular table, the table name is supplied. Otherwise, this value is NULL. The table name is the name of a table in the client application. This name may or may not have a direct counterpart in the consolidated database, depending upon the design of the synchronization system.

The action code tells the MobiLink synchronization server what to do next. Before it calls this script, the MobiLink synchronization server sets the action code to a default value, which depends upon the severity of the error. Your script may modify this value. Your script must return or set an action code.

You can return a value from the `handle_error` script in two ways.

- ◆ Pass the action parameter to an OUTPUT parameter of a procedure:

```
CALL my_handle_error( ?, ?, ?, ?, ? )
```

- ◆ Set the action code via a procedure or function return value:

```
? = CALL my_handle_error( ?, ?, ?, ? )
```

Most DBMSs use the RETURN statement to set the return value from a procedure or function.

The CustDB sample application contains error handlers for various database-management systems.

See also

"report_error connection event" on page 529

"report_odbc_error connection event" on page 531

"handle_odbc_error connection event" on page 515

SQL example

The following example works with an Adaptive Server Anywhere consolidated database. It allows your application to ignore redundant inserts.

```
CREATE PROCEDURE ULHandleError( INOUT action integer,
IN error_code integer, IN error_message varchar(1000),
IN user_name varchar(128), IN table_name varchar(128) )
BEGIN
  -- -196 is SQLE_INDEX_NOT_UNIQUE
  -- -194 is SQLE_INVALID_FOREIGN_KEY
  if error_code = -196 or error_code = -194 then
    -- ignore the error and keep going
    SET action = 1000;
  else
    -- abort the synchronization
    SET action = 3000;
  end if;
END
```

Java example

The following stored procedure call registers a Java method called `handleError` as the script for the `handle_error` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',
'handle_error',
'ExamplePackage.ExampleClass.handleError' )
```

Following is the sample Java method `handleError`. It processes an error based on the data that is passed in. It also determines the resulting error code.

```
public String handleError(
  ianywhere.ml.script.InOutInteger actionCode, int
  errorCode, String errorMessage, String user, String
  table )
{  int new_ac;
  if( user == null )
  {  new_ac = handleNonSyncError( errorCode,
    errorMessage ); }
  else if( table == null )
  {  new_ac = handleConnectionError( errorCode,
    errorMessage, user ); }
  else
  {  new_ac = handleTableError( errorCode,
    errorMessage, user, table ); }
  // keep the most serious action code
  if( actionCode.getValue() < new_ac )
  {  actionCode.setValue( new_ac ); }
  return( null ); }
```


handle_odbc_error connection event

Function Executed whenever the MobiLink synchronization server encounters an error triggered by the ODBC Driver Manager.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|--------------------------------------|
| 1 | action_code | INTEGER. This is an INOUT parameter. |
| 2 | ODBC_state | VARCHAR(5) |
| 3 | error_message | TEXT |
| 4 | ml_username | VARCHAR(128) |
| 5 | table | VARCHAR(128) |

Default action The MobiLink synchronization server selects a default action code. You can modify the action code in the script, and return a value instructing MobiLink how to proceed. The action code parameter takes one of the following values:

- ◆ **1000** Skip the current row and continue processing.
- ◆ **3000** Rollback the current transaction and cancel the current synchronization. This is the default action code, and is used when no handle_error script is defined or this script causes an error.
- ◆ **4000** Rollback the current transaction, cancel the synchronization, and shut down the MobiLink synchronization server.

Description The MobiLink synchronization server executes this script whenever it encounters an error flagged by the ODBC Driver Manager during the synchronization process. The error codes allow you to identify the nature of the error.

The action code tells the MobiLink synchronization server what to do next. Before it calls this script, the MobiLink synchronization server sets the action code to a default value, which depends upon the severity of the error. Your script may modify this value. Your script must return or set an action code.

The `handle_odbc_error` script is called after the `handle_error` and `report_error` scripts, and before the `report_odbc_error` script.

When only one, but not both, error-handling script is defined, the return value from that script decides error behavior. When both error-handling scripts are defined, the MobiLink synchronization server uses the numerically highest action code. If both `handle_error` and `handle_ODBC_error` are defined, MobiLink uses the numerically highest action code returned from all calls.

See also

"`handle_error` connection event" on page 512
"report_error connection event" on page 529
"report_odbc_error connection event" on page 531

Java example

The following stored procedure call registers a Java method called `handleODBCError` as the script for the `handle_odbc_error` event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',  
    'handle_odbc_error',  
    'ExamplePackage.ExampleClass.handleODBCError' )
```

Following is the sample Java method `handleODBCError`. It processes an error based on the data that is passed in. It also determines the resulting error code.

```
public String handleODBCError(  
    ianywhere.ml.script.InOutInteger actionCode,  
    String ODBCState, String errorMessage, String user,  
    String table )  
{    int new_ac;  
    if( user == null )  
    {    new_ac = handleNonSyncError( ODBCState,  
        errorMessage ); }  
    else if( table == null )  
    {    new_ac = handleConnectionError( ODBCState,  
        errorMessage, user ); }  
    else {    new_ac = handleTableError( ODBCState,  
        errorMessage, user, table ); }  
    // keep the most serious action code  
    if( actionCode.getValue() < new_ac )  
    {    actionCode.setValue( new_ac ); }  
    return( null ); }
```

modify_last_download_timestamp connection event

Function The script can be used to modify the last_download timestamp for the current synchronization.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|--------------------|--|
| 1 | download_timestamp | TIMESTAMP. This is an INOUT parameter. |
| 2 | ml_username | VARCHAR(128) |

Default action None.

Description Use this script when you want to modify the last_download time for the current synchronization. If this script is defined, the MobiLink synchronization server calls this script and uses the modified last_download timestamp as the last_download timestamp passed to the download scripts.

SQL scripts for the modify_last_download_timestamp event must be implemented as stored procedures.

The download_timestamp parameter is an INOUT parameter: a parameter that provides a value to the procedure, and could be given a new value by the procedure.

SQL example The following example downloads everything from one day ago, regardless of whether the databases were synchronized since then.

First, create a procedure for your Adaptive Server Anywhere consolidated database:

```
CREATE PROCEDURE ModifyLastDownloadTimestamp (
  inout last_download_time TIMESTAMP , in user_name
  VARCHAR(128) )
BEGIN
  SELECT dateadd(day, -1, last_download_time )
  INTO last_download_time
END
```

Second, install the script into your Adaptive Server Anywhere consolidated database:

```
call ml_add_connection_script( 'modify_ts_test',  
    'modify_last_download_timestamp',  
    'call ModifyLastDownloadTimestamp ( ?, ? )' )
```

Java example

The following stored procedure call registers a Java method called `modifyLastDownloadTimestamp` as the script for the `modify_last_download_timestamp` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',  
    'modify_last_download_timestamp',  
    'ExamplePackage.ExampleClass.modifyLastDownloadTimestamp' )
```

Following is the sample Java method `modifyLastDownloadTimestamp`. It prints the current and new timestamp and modifies the timestamp that is passed in.

```
public String modifyLastDownloadTimestamp( Timestamp  
last_download_time, String user_name )  
{ java.lang.System.out.println( "old date: " +  
    last_download_time.toString() );  
    last_download_time.setDate(  
        last_download_time.getDate() -1 );  
    java.lang.System.out.println( "new date: " +  
        last_download_time.toString() );  
    return( null ); }
```

modify_next_last_download_timestamp connection event

- Function** The script can be used to modify the last_download timestamp for the next synchronization.
- Parameters** In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.
- Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|--------------------|--|
| 1 | download_timestamp | TIMESTAMP. This is an INOUT parameter. |
| 2 | last_download | TIMESTAMP |
| 3 | ml_username | VARCHAR(128) |

Default action None.

Description Use this script when you want to modify the last_download time for the next synchronization. If this script is defined, the MobiLink synchronization server calls this script and sends the next last_download timestamp down to the remote, which will send it as part of the next synchronization.

SQL scripts for the modify_next_last_download_timestamp event must be implemented as stored procedures.

The download_timestamp parameter is an INOUT parameter: a parameter that provides a value to the procedure, and could be given a new value by the procedure.

The order of parameters changes if you use the -zd dbmlsrv8 option. When you use -zd, the order is download_timestamp, ml_username, last_download.

SQL example The following example shows one application of this script. First, create a procedure for your Adaptive Server Anywhere consolidated database:

```
CREATE PROCEDURE ModifyNextDownloadTimestamp (
    inout download_timestamp TIMESTAMP ,
    in last_download TIMESTAMP ,
    in user_name VARCHAR(128) )
BEGIN
    SELECT dateadd(hour, -1, download_timestamp )
    INTO download_timestamp
END
```

Second, install the script into your Adaptive Server Anywhere consolidated database:

```
call ml_add_connection_script(
    'modify_ts_test',
    'modify_next_last_download_timestamp',
    'call ModifyNextDownloadTimestamp ( ?, ?, ? )' )
```

Java example

The following stored procedure call registers a Java method called `modifyNextDownloadTimestamp` as the script for the `modify_next_last_download_timestamp` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',
    'modify_next_last_download_timestamp',
    'ExamplePackage.ExampleClass.modifyNextDownloadTimestamp' )
```

Following is the sample Java method `modifyNextDownloadTimestamp`. It sets the download timestamp back an hour.

```
public String modifyNextDownloadTimestamp(Timestamp
download_timestamp, Timestamp last_download,
String user_name )
{ download_timestamp.setHours(
    download_timestamp.getHours() -1 );
    return( null ); }
```

modify_user connection event

Function Provide the MobiLink user name.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

| Item | Parameter | Description |
|------|-------------|--|
| 1 | ml_username | VARCHAR(128). This is an INOUT parameter. |

Default action None

Description The MobiLink server provides the user name as a parameter when it calls scripts; the user name is sent by the MobiLink client. In some cases, you may want to have an alternate user name. This script allows you to modify the user name used in calling MobiLink scripts.

The ml_username parameter is an INOUT parameter: a parameter that provides a value to the script, and could be given a new value by the script. The parameter must be long enough to hold the user name.

SQL scripts for the modify_last_download_timestamp event must be implemented as stored procedures.

See also "authenticate_user connection event" on page 446
"authenticate_user_hashed connection event" on page 450

Java example The following stored procedure call registers a Java method called modifyUser as the script for the modify_user connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',
    'modify_user',
    'ExamplePackage.ExampleClass.modifyUser' )
```

Following is the sample Java method modifyUser. It gets the user ID from the database and then uses it to set the user name.

```
public void ModifyUser( InOutString io_user_name )
    throws SQLException
{
    Statement uid_select = curConn.createStatement();
    ResultSet uid_result = uid_select.executeQuery(
        "select rep_id from SalesRep where name = '" +
        io_user_name.getValue() + "' " );
    uid_result.next();
    io_user_name.setValue(
        java.lang.Integer.toString(uid_result.getInt( 1 ))
    );
    uid_result.close();
    uid_select.close();
    return; }
}
```

.NET example

The following stored procedure call registers a .NET method called ModUser as the script for the modify_user connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script( 'ver1',
    'modify_user',
    'TestScripts.Test.ModUser'
)
```

Following is the C# signature for the call ModUser.

```
public void ModUser( string user )
```


new_row_cursor cursor event

Function

Defines the insert cursor that the MobiLink synchronization server uses to insert the new values of rows that were updated in the remote database, but conflict with values presently in the consolidated database.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |

Description

When a row is updated on a remote database, the MobiLink client saves a copy of the original values. The client sends both old and new values to the MobiLink synchronization server. Also used to input an INSERT operation in forced conflict mode.

When the MobiLink synchronization server receives an updated row, it compares the original values with the present values in the consolidated database, using the upload_cursor. If the old uploaded values do not match the current value in the consolidated database, the row conflicts. Instead of updating the row, the MobiLink synchronization server inserts both old and new values into the consolidated database using the old_row_cursor and the new_row_cursor, respectively.

The MobiLink synchronization server uses a cursor to insert the new uploaded values from conflicting rows into the consolidated database. This script contains the SELECT statement used to define this cursor.

It is common practice to use temporary tables to store the old and new versions of conflicting rows. You can create these temporary tables in an earlier script.

You can have one new_row_cursor script for each table in the remote database.

Normally, the columns in the select list must match those in the client table in both order and type. However, the MobiLink synchronization server permits you to add one extra column. If you do so, the MobiLink synchronization server automatically inserts the user name into the first column, then proceeds to insert the new row values using the remaining columns, as usual.

Note

The script is ignored if any of the following scripts are defined for the same table: upload_insert, upload_update, upload_delete, upload_fetch, upload_new_row_insert, upload_old_row_insert.

See also

"Writing upload_cursor scripts" on page 68
"upload_cursor cursor event" on page 543
"old_row_cursor cursor event" on page 525
"Handling conflicts" on page 104
"resolve_conflict table event" on page 533

SQL example

The following SELECT statement defines a *new_row_cursor* script suited to the CustDB sample application.

```
SELECT order_id, cust_id, prod_id, emp_id, disc, quant,  
       notes, status FROM ULNewOrder FOR update
```

The primary key of the ULOrder table is *order_id*.

The following SELECT statement could instead be used for the same client table. This variation includes the permitted one extra row. The MobiLink synchronization server automatically stores the user name in the first column.

```
SELECT user_name, order_id, cust_id, prod_id, emp_id,  
       disc, quant, notes, status FROM ULNewOrder FOR update
```

Java example

This script must return valid SQL.

The following stored procedure call registers a Java method called newRowCursor as the script for the new_row_cursor event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script(  
    'ver1', 'table1', 'new_row_cursor',  
    'ExamplePackage.ExampleClass.newRowCursor' )
```

Following is the sample Java method newRowCursor. It dynamically generates a new row cursor statement by calling a Java method.

```
public String newRowCursor()  
{ return( getRowCursor ( _curTable ) ); }
```

old_row_cursor cursor event

Function Defines the cursor that the MobiLink synchronization server uses to insert the old values of rows that were updated in the remote database, but that conflict with values presently in the consolidated database. The event is also used to insert the values of deleted rows when in forced conflict mode.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |

Default action None.

Description When a row is updated on a remote database, the MobiLink client saves a copy of the original values. The client sends both old and new values to the MobiLink synchronization server.

When the MobiLink synchronization server receives an updated row, it compares the original values with the present values in the consolidated database, using the upload_cursor cursor event. If the old uploaded values do not match the current values in the consolidated database, the row conflicts. Instead of updating the row, the MobiLink synchronization server inserts both old and new values into the consolidated database using the old_row_cursor event and the new_row_cursor event.

It is common practice to use temporary tables to store the old and new versions of conflicting rows. In Adaptive Server Anywhere, you can create these tables in an earlier script. Some non-ASA consolidated databases support temporary tables, but they usually differ significantly from the temporary tables offered by ASA. Consult your DBMS documentation for details. An alternative to a temporary table is a base table with an extra column for the MobiLink user name. This effectively partitions the rows of the base table between concurrent synchronizations.

The MobiLink synchronization server uses a cursor to insert the old uploaded values from conflicting rows into the consolidated database. This script contains the SELECT statement used to define this cursor.

You can have one old_row_cursor script for each table in the remote database.

Normally, the columns in the SELECT list must match those in the client table in both order and type. However, the MobiLink synchronization server permits you to add one extra column. If you do so, the MobiLink synchronization server automatically inserts the user name into the first column, then proceeds to insert the old row values using the remaining columns, as usual.

See also

"Writing upload_cursor scripts" on page 68

"upload_cursor cursor event" on page 543

"new_row_cursor cursor event" on page 523

SQL example

The following SELECT statement defines an *old_row_cursor* script suited to the CustDB sample application for an Oracle installation. The primary key of the ULOrder table is order_id.

```
SELECT order_id, cust_id, prod_id, emp_id, disc, quant,  
       notes, status FROM ULOldOrder
```

The following SELECT statement could instead be used for the same client table. This variation includes the permitted one extra row. The MobiLink synchronization server automatically stores the user name in the first column.

```
SELECT user_name, order_id, cust_id, prod_id, emp_id,  
       disc, quant, notes, status FROM ULOldOrder
```

Java example

This script must return valid SQL.

The following stored procedure call registers a Java method called *oldRowCursor* as the script for the *old_row_cursor* event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',  
                              'old_row_cursor',  
                              'ExamplePackage.ExampleClass.oldRowCursor' )
```

Following is the sample Java method *oldRowCursor*. It dynamically generates an old row cursor statement by calling a Java method.

```
public String oldRowCursor()  
{ return( getRowCursor( _curTable ) ); }
```

prepare_for_download connection event

Function Processes any required operations between the upload and download transactions.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------|--------------|
| 1 | last_download | TIMESTAMP |
| 2 | ml_username | VARCHAR(128) |

Default action None.

Description The MobiLink synchronization server executes this script as a separate transaction, between the upload transaction and the start of the download transaction.

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 user has never synchronized successfully, this value is set to 1900-01-01.

See also "end_upload connection event" on page 499
"begin_download connection event" on page 454

SQL example You use this event to process rows deleted during the upload stream on the intermediate table. You can compare the rows in the base table with the rows in the intermediate table and decide what to do with the deleted row.

```
Call ml_add_table_script('version1', 'Leads',
'end_uploads_deletes', 'call EndUploadDeletesLeads()');

Create procedure EndUploadDeletesLeads ( )
Begin
    FOR names AS curs CURSOR FOR
        SELECT LeadID FROM Leads WHERE LeadID NOT IN (SELECT
        LeadID FROM T_Leads);
    DO
        CALL decide_what_to_do( LeadID );
    END FOR;
end
```

Java example

The following stored procedure call registers a Java method called `prepareForDownload` as the script for the `prepare_for_download` event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',  
    'prepare_for_download',  
    'ExamplePackage.ExampleClass.prepareForDownload' )
```

Following is the sample Java method `prepareForDownload`. It calls a Java method to modify some rows in the database.

```
public String prepareForDownload ( Timestamp ts,  
    String user )  
{    adjustUploadedRows( _syncConn, user );  
    return( null ); }
```

.NET example

The following stored procedure call registers a .NET method called `PrepareForDownload` as the script for the `prepare_for_download` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script( 'ver1',  
    'prepare_for_download',  
    'TestScripts.Test.PrepareForDownload'  
    )
```

Following is the C# signature for the call `PrepareForDownload`.

```
public void PrepareForDownload(  
    DateTime timestamp,  
    string user )
```

report_error connection event

Function Allows you to log errors and to record the actions selected by the `handle_error` script.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|----------------------------|---|
| 1 | <code>action_code</code> | INTEGER. This parameter is mandatory. |
| 2 | <code>error_code</code> | INTEGER. This parameter is optional if none of the following parameters are specified. |
| 3 | <code>error_message</code> | TEXT. This parameter is optional if none of the following parameters are specified. |
| 4 | <code>ml_username</code> | VARCHAR(128). This parameter is optional if none of the following parameters are specified. |
| 5 | <code>table</code> | VARCHAR(128). This parameter is optional. |

Default action None.

Description This script allows you to log errors and to record the actions selected by the `handle_error` script. This script is executed after the `handle_error` event, whether or not a `handle_error` script is defined. It is always executed in its own transaction, on a different database connection than the synchronization connection (the administrative/information connection).

The error code and error message allow you to identify the nature of the error. The action code value is returned by the last call to an error handling script for the SQL operation that caused the current error.

If the error happened as part of synchronization, the user name is supplied. Otherwise, this value is NULL.

If the error happened while manipulating a particular table, the table name is supplied. Otherwise, this value is NULL. The table name is the name of a table in the remote database. This name may or may not have a direct counterpart in the consolidated database, depending on the design of the synchronization system.

See also

"handle_error connection event" on page 512

"handle_odbc_error connection event" on page 515

"report_odbc_error connection event" on page 531

Java example

The following stored procedure call registers a Java method called `reportError` as the script for the `report_error` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',  
    'report_error',  
    'ExamplePackage.ExampleClass.reportError' )
```

Following is the sample Java method `reportError`. It logs the error to a table using the JDBC connection provided by MobiLink. It also sets the action code.

```
public String reportError(  
    ianywhere.ml.script.InOutInteger actionCode,  
    int errorCode, String errorMessage, String user,  
    String table )  
    throws java.sql.SQLException  
{ // insert error information in a table  
    JDBCLogError( _syncConn, errorCode, errorMessage,  
        user, table );  
    actionCode.setValue( getActionCode( errorCode ) );  
    return( null ); }
```


report_odbc_error connection event

Function Allows you to log errors and to record the actions selected by the `handle_odbc_error` script.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|----------------------------|---|
| 1 | <code>action_code</code> | INTEGER. This parameter is mandatory. |
| 2 | <code>ODBC_state</code> | VARCHAR(5). This parameter is optional if none of the following parameters are specified. |
| 3 | <code>error_message</code> | TEXT. This parameter is optional if none of the following parameters are specified. |
| 4 | <code>ml_username</code> | VARCHAR(128). This parameter is optional if none of the following parameters are specified. |
| 5 | <code>table</code> | VARCHAR(128). This parameter is optional. |

Default action None.

Description This script allows you to log errors and to record the actions selected by the `handle_odbc_error` script. This script is executed after the `handle_odbc_error` event, whether or not a `handle_odbc_error` script is defined. It is always executed in its own transaction, on a different database connection than the synchronization connection (the administrative/information connection).

The error code and error message allow you to identify the nature of the error. The action code value is returned by the last call to an error handling script for the SQL operation that caused the current error.

If the error happened as part of synchronization, the user name is supplied. Otherwise, this value is NULL.

If the error happened while manipulating a particular table, the table name is supplied. Otherwise, this value is NULL. The table name is the name of a table in the remote database. This name may or may not have a direct counterpart in the consolidated database, depending on the design of the synchronization system.

See also

"handle_error connection event" on page 512

"handle_odbc_error connection event" on page 515

"report_error connection event" on page 529

Java example

The following stored procedure call registers a Java method called reportODBCError as the script for the report_odbc_error event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',  
    'report_odbc_error',  
    'ExamplePackage.ExampleClass.reportODBCError' )
```

Following is the sample Java method reportODBCError. It logs the error to a table using the JDBC connection provided by MobiLink. It also sets the action code.

```
public String reportODBCError (  
    ianywhere.ml.script.InOutInteger actionCode,  
    String ODBCState, String errorMessage, String user,  
    String table )  
    throws java.sql.SQLException  
{    JDBCLogError( _syncConn, ODBCState, errorMessage,  
    user, table );  
    actionCode.setValue( getActionCode( ODBCState ) );  
    return( null ); }
```

resolve_conflict table event

Function

Defines a process for resolving a conflict in a specific table.

Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. You must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | table | VARCHAR(128) |

Default action

None.

Description

When a row is updated on a remote database, the MobiLink client saves a copy of the original values. The client sends both old and new values to the MobiLink synchronization server.

When the MobiLink synchronization server receives an updated row, it compares the original values with the present values in the consolidated database. The comparison is carried out using the `upload_fetch` script or, if using cursor-based uploads, the `upload_cursor` script.

If the old uploaded values do not match the current values in the consolidated database, the row conflicts. Instead of updating the row, the MobiLink synchronization server inserts both old and new values into the consolidated database. The old and new rows are handled using the `upload_old_row_insert` and `upload_new_row_insert` scripts, respectively. If you are using cursor-based uploads the rows are handled using `old_row_cursor` and `new_row_cursor`, respectively.

Once the values have been inserted, the MobiLink synchronization server executes the `resolve_conflict` script. It provides the opportunity to resolve the conflict. You can implement any scheme of your choosing.

This script is executed once per conflict.

Alternatively, instead of defining the `resolve_conflict` script, you can resolve conflicts in a set-oriented fashion by putting conflict-resolution logic either in your `end_upload_rows` script or in your `end_upload table` script.

You can have one `resolve_conflict` script for each table in the remote database.

See also

"upload_old_row_insert table event" on page 553
"upload_new_row_insert table event" on page 551
"upload_update table event" on page 560
"old_row_cursor cursor event" on page 525
"new_row_cursor cursor event" on page 523
"end_upload_rows table event" on page 506

SQL example

The following statement defines a *resolve_conflict* script suited to the CustDB sample application for an Oracle installation. It calls a stored procedure *ULResolveOrderConflict*.

```
exec ml_add_table_script( 'custdb', 'ULOrder',  
  'resolve_conflict','begin ULResolveOrderConflict();  
end; ' )  
CREATE OR REPLACE PROCEDURE ULResolveOrderConflict()  
AS  
  new_order_id integer;  
  new_status   varchar(20);  
  new_notes    varchar(50);  
BEGIN  
  -- approval overrides denial  
  SELECT order_id, status, notes  
    INTO new_order_id, new_status, new_notes  
    FROM ULNewOrder  
   WHERE syncuser_id = SyncUserID;  
  IF new_status = 'Approved' THEN  
    UPDATE ULOrder o  
      SET o.status = new_status, o.notes =  
new_notes  
    WHERE o.order_id = new_order_id;  
  END IF;  
  DELETE FROM ULOldOrder;  
  DELETE FROM ULNewOrder;  
END;
```

Java example

The following stored procedure call registers a Java method called *resolveConflict* as the script for the *resolve_conflict* table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',  
  'resolve_conflict',  
  'ExamplePackage.ExampleClass.resolveConflict' )
```

Following is the sample Java method *resolveConflict*. It calls a Java method that will use the JDBC connection provided by MobiLink. It also sets the action code.

```
public String resolveConflict( String user,  
  String table)  
{  resolveRows(_syncConn, user ); }
```

synchronization_statistics connection event

Function Tracks synchronization statistics.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|---------------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | warnings | INTEGER |
| 3 | errors | INTEGER |
| 4 | deadlocks | INTEGER |
| 5 | synchronized_tables | INTEGER |
| 6 | connection_retries | INTEGER |

Default action None.

Description The `synchronization_statistics` event allows you to gather, for any user and connection, various statistics about the current synchronization. The `synchronization_statistics` connection script is called just prior to the commit at the end of the end synchronization transaction.

See also "download_statistics connection event" on page 479
 "download_statistics table event" on page 482
 "upload_statistics connection event" on page 554
 "upload_statistics table event" on page 557
 "synchronization_statistics table event" on page 537
 "time_statistics connection event" on page 539
 "time_statistics table event" on page 541
 "MobiLink Monitor" on page 231

SQL example The following example comes from an Oracle installation.

```
INSERT INTO sync_con_audit (id, ml_user, warnings,
errors, deadlocks, synchronized_tables,
connection_retries) VALUES (s_audit.nextval,?,?,?,?,?,?)
```

Once statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

Java example

The following stored procedure call registers a Java method called `synchronizationStatisticsConnection` as the script for the `synchronization_statistics` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1', 'synchronization_statistics',  
  'ExamplePackage.ExampleClass.synchronizationStatisticsConnection' )
```

Following is the sample Java method `synchronizationStatisticsConnection`. It logs some of the statistics to the MobiLink output log. (This might be useful at development time but would slow down a production server.)

```
public String synchronizationStatisticsConnection(  
    String user, int warnings, int errors, int deadlocks,  
    int synchronizedTables, int connectionRetries )  
{  
    java.lang.System.out.println( "synch statistics  
    number of deadlocks: " + deadlocks ;  
    return( null ); }  
}
```

.NET example

The following stored procedure call registers a .NET method called `SyncStats` as the script for the `synchronization_statistics` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script(  
    'ver1',  
    'synchronization_statistics',  
    'TestScripts.Test.SyncStats'  
)
```

Following is the C# signature for the call `SyncStats`.

```
public void SyncStats( string user,  
    int warnings,  
    int errors,  
    int deadLocks,  
    int syncedTables,  
    int connRetries )
```

synchronization_statistics table event

Function Tracks synchronization statistics.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | table | VARCHAR(128) |
| 3 | warnings | INTEGER |
| 4 | errors | INTEGER |

Default action None.

Description The synchronization_statistics event allows you to gather, for any user and table, the number of warnings and errors that occurred during synchronization. The synchronization_statistics table script is called just prior to the commit at the end of the end synchronization transaction.

See also "download_statistics connection event" on page 479
 "download_statistics table event" on page 482
 "upload_statistics connection event" on page 554
 "upload_statistics table event" on page 557
 "synchronization_statistics connection event" on page 535
 "time_statistics connection event" on page 539
 "time_statistics table event" on page 541
 "MobiLink Monitor" on page 231

SQL example The following example comes from an Oracle installation.

```
INSERT INTO sync_tab_audit (id, ml_user, table,
  warnings, errors) VALUES (s_audit.nextval,?,?,?,?)
```

Once synchronization statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

Java example

The following stored procedure call registers a Java method called `synchronizationStatisticsTable` as the script for the `synchronization_statistics` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script( 'ver1', 'table1',  
'synchronization_statistics',  
'ExamplePackage.ExampleClass.synchronizationStatisticsTable' )
```

Following is the sample Java method `synchronizationStatisticsTable`. It logs some of the statistics to the MobiLink output log. (This might be useful at development time but would slow down a production server.)

```
public String synchronizationStatisticsTable(  
    String user, String table, int warnings, int errors )  
{  
    java.lang.System.out.println( "synch statistics for  
        table: " + table + " errors: " + errors );  
    return( null ); }  
}
```

.NET example

The following stored procedure call registers a .NET method called `SyncTableStats` as the script for the `synchronization_statistics` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1', 'synchronization_statistics',  
    'TestScripts.Test.SyncTableStats'  
)
```

Following is the C# signature for the call `SyncTableStats`.

```
public void SyncTableStats(  
    string user,  
    string table,  
    int warnings,  
    int errors )
```


time_statistics connection event

Function Tracks time statistics by user and event.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | event_name | VARCHAR(128) |
| 3 | num_calls | INTEGER |
| 4 | min_time | INTEGER |
| 5 | max_time | INTEGER |
| 6 | total_time | INTEGER |

Default action None.

Description The time_statistics event allows you to gather time statistics for any user during synchronization. The statistics are gathered only for those events for which there is a corresponding script. The script gathers aggregate data for occasions where a single event occurs multiple times. The script can be especially useful for time comparisons across users, events and tables.

See also "time_statistics table event" on page 541
 "download_statistics connection event" on page 479
 "download_statistics table event" on page 482
 "upload_statistics connection event" on page 554
 "upload_statistics table event" on page 557
 "synchronization_statistics connection event" on page 535
 "synchronization_statistics table event" on page 537
 "MobiLink Monitor" on page 231

SQL example The following example comes from an Oracle installation.

```
INSERT INTO time_statistics (id, ml_user, table,
event_name, num_calls, min_time, max_time, total_time)
VALUES (ts_id.nextval,?,?,?,?,?,?)
```

Java example

The following stored procedure call registers a Java method called `timeStatisticsConnection` as the script for the `time_statistics` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1',
    'time_statistics',
    'ExamplePackage.ExampleClass.timeStatisticsConnection' )
```

Following is the sample Java method `timeStatisticsConnection`. It prints statistics for the `prepare_for_download` event.

```
public void timeStatisticsConnection( String
ml_username, String table_name, String event_name,
int num_calls, int min_time, int max_time,
int total_time )
{ if( event_name.equals( "prepare_for_download" )
  { java.lang.System.out.println(
    "preapre_for_download num_calls: " + num_calls +
    "total_time: " + total_time ); } }
```

.NET example

The following stored procedure call registers a .NET method called `TimeStats` as the script for the `time_statistics` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script( 'ver1',
    'time_statistics',
    'TestScripts.Test.TimeStats'
)
```

Following is the C# signature for the call `TimeStats`.

```
public void TimeStats(
    string user,
    string eventName,
    int numCalls,
    int minTime,
    int maxTime,
    int totTime )
```

time_statistics table event

Function Tracks time statistics.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|-------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | table | VARCHAR(128) |
| 3 | event_name | VARCHAR(128) |
| 4 | num_calls | INTEGER |
| 5 | min_time | INTEGER |
| 6 | max_time | INTEGER |
| 7 | total_time | INTEGER |

Default action None.

Description The time_statistics table event allows you to gather time statistics for any user and table during synchronization. The statistics are gathered only for those events for which there is a corresponding script. The script gathers aggregate data for occasions where a single event occurs multiple times. The script can be especially useful for time comparisons across users, events and tables.

See also "time_statistics connection event" on page 539
 "download_statistics connection event" on page 479
 "download_statistics table event" on page 482
 "upload_statistics connection event" on page 554
 "upload_statistics table event" on page 557
 "synchronization_statistics connection event" on page 535
 "synchronization_statistics table event" on page 537
 "MobiLink Monitor" on page 231

SQL example The following example comes from an Oracle installation.

```
INSERT INTO time_statistics ( id, ml_user, table,
event_name, num_calls, min_time, max_time, total_time)
VALUES (ts_id.nextval,?,?,?,?,?,?)
```

Java example

The following stored procedure call registers a Java method called `timeStatisticsTable` as the script for the `time_statistics` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script(  
    'ver1', 'table1', 'time_statistics',  
    'ExamplePackage.ExampleClass.timeStatisticsTable' )
```

Following is the sample Java method `timeStatisticsTable`. It prints statistics for the `upload_old_row_insert` event.

```
public void timeStatisticsConnection( String  
ml_username, String table_name, String event_name,  
int num_calls, int min_time, int max_time,  
int total_time )  
{ if( event_name.equals( "upload_old_row_insert" )  
  { java.lang.System.out.println(  
    "upload_old_row_insert num_calls: " + num_calls +  
    "total_time: " + total_time ); } }
```

.NET example

The following stored procedure call registers a .NET method called `TimeTableStats` as the script for the `time_statistics` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1', 'time_statistics',  
    'TestScripts.Test.TimeTableStats'  
)
```

Following is the C# signature for the call `TimeTableStats`.

```
public void TimeTableStats(  
    string user,  
    string table,  
    string eventName,  
    int numCalls,  
    int minTime,  
    int maxTime,  
    int totTime )
```

upload_cursor cursor event

Function Defines a cursor that the MobiLink synchronization server uses to insert, update, or delete rows during processing of the upload stream.

Statement-based events recommended

For most purposes, it is recommended that you use the statement-based events `upload_delete`, `upload_insert`, and `upload_update` instead of the `upload_cursor` event to process the upload stream.

Parameters

| Item | Parameter |
|------|---------------|
| 1 | primary key 1 |
| 2 | primary key 2 |
| ... | ... |

Default action

None.

A default `upload_cursor` SQL script can be generated using the MobiLink synchronization server `-zac` option. Also, the UltraLite analyzer generates a `SELECT` statement based on your reference database that you can use to get started.

Description

The MobiLink synchronization server opens a cursor with which to insert, update, or delete rows in the consolidated database based on rows uploaded from a client application. This script should contain a suitable `SELECT` statement or call a stored procedure that contains a suitable `SELECT` statement.

The parameters are the values of each column included in the primary key of the corresponding client table. You must use these in a `WHERE` clause, so that the synchronization can identify a unique row based on these values. The type and order of the parameters is as defined in the `example_upload_cursor` script. This order is the same as that in the corresponding table definition in the remote database, which in turn may have been copied from your reference database.

You can have one `upload_cursor` script for each table in the remote database.

For Java and .NET applications, this script must return valid SQL.

See also

"Writing `upload_cursor` scripts" on page 68
 "`upload_delete` table event" on page 545
 "`upload_insert` table event" on page 549
 "`upload_update` table event" on page 560
 "`download_cursor` cursor event" on page 474

SQL example

The following SELECT statement defines the upload cursor in the CustDB sample application.

```
SELECT cust_id, cust_name
FROM ULCustomer
WHERE cust_id = ?
```

The primary key of the ULCustomer table in the CustDB sample application is the column cust_id. If the corresponding table in the consolidated database is, instead, named Customer, then change the above statement as follows.

```
SELECT cust_id, cust_name
FROM Customer
WHERE cust_id = ?
```

Java example

The following stored procedure call registers a Java method called uploadCursor as the script for the upload_cursor cursor event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script(
    'ver1', 'table1', 'upload_cursor',
    'ExamplePackage.ExampleClass.uploadCursor' )
```

Following is the sample Java method uploadCursor. It dynamically generates an upload cursor.

```
public String uploadCursor()
{ return( getUploadCursor( _curTable ) ); }
```

.NET example

The following C# example deletes the contents of a temporary table. It then returns SQL that causes rows to be uploaded into the temporary table.

```
public string UploadCursor()
{
    DBCommand stmt = curConn.CreateCommand();
    stmt.CommandText = "DELETE FROM dnet_ul_temp";
    stmt.ExecuteNonQuery();
    stmt.Close();

    return( "SELECT * FROM dnet_ul_temp WHERE pk = ?" );
}
```

upload_delete table event

Function Provides an event that the MobiLink synchronization server uses during processing of the upload stream to handle rows deleted from the remote database.

Parameters

| Item | Parameter |
|------|---------------|
| 1 | primary key 1 |
| 2 | primary key 2 |
| ... | ... |

Description The statement-based upload_delete script handles rows that are deleted in the remote database. The action taken at the consolidated database can be a DELETE statement, but need not be.

You can have one upload_delete script for each table in the remote database.

For Java and .NET applications, this script must return valid SQL.

See also "upload_insert table event" on page 549
"upload_update table event" on page 560

SQL example This example is taken from the Contact sample and can be found in *Samples\MobiLink\Contact\build_consol.sql*. It marks customers that are deleted from the remote database as inactive.

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

Java example The following stored procedure call registers a Java method called uploadDeleteTable as the script for the upload_delete table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script(
    'ver1', 'table1', 'upload_delete',
    'ExamplePackage.ExampleClass.uploadDeleteTable' )
```

Following is the sample Java method uploadDeleteTable. It dynamically generates an UPLOAD statement.

```
public string uploadDeleteTable()
{ return( genUD(_curTable) ); }
```

.NET example

The following stored procedure call registers a .NET method called UploadDelete as the script for the upload_delete table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1', 'upload_delete',  
    'TestScripts.Test.UploadDelete'  
)
```

Following is the C# signature for the call UploadDelete.

```
public string UploadDelete( object pk1 )
```


upload_fetch table event

Function Provides an event that the MobiLink synchronization server uses to identify update conflicts during statement-based processing of the upload stream.

Parameters

| Item | Parameter |
|------|---------------|
| 1 | primary key 1 |
| 2 | primary key 2 |
| ... | ... |

Description

The statement-based `upload_fetch` script fetches rows from a synchronized table for the purposes of conflict detection. It is a companion to the `upload_update` event.

The columns of the result set must match the number of columns being uploaded from the remote database for this table. If the values returned do not match the before image in the uploaded row, a conflict is identified.

You can have one `upload_fetch` script for each table in the remote database.

See also

"resolve_conflict table event" on page 533

"upload_delete table event" on page 545

"upload_insert table event" on page 549

"upload_update table event" on page 560

SQL example

The following SQL script is taken from the Contact sample and can be found in *Samples\MobiLink\Contact\build_consol.sql*. It is used to identify conflicts that occur when rows updated in the remote database *Product* table are uploaded. This script selects rows from a table also named *Product*, but depending on your consolidated and remote database schema, the two table names may not match.

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

Java example

This script must return valid SQL.

The following stored procedure call registers a Java method called `uploadFetchTable` as the script for the `upload_fetch` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script(
    'ver1', 'table1', 'upload_fetch',
    'ExamplePackage.ExampleClass.uploadFetchTable' )
```

Following is the sample Java method `uploadFetchTable`. It dynamically generates an `UPLOAD` statement.

```
public string uploadFetchTable()  
{   return( genUF(_curTable) ); }
```

upload_insert table event

Function Provides an event that the MobiLink synchronization server uses during processing of the upload stream to handle rows inserted into the remote database.

Parameters

| Item | Parameter |
|------|-----------|
| 1 | column 1 |
| 2 | column 2 |
| ... | ... |

Description The statement based upload_insert script performs direct inserts of column values.

You can have one upload_insert script for each table in the remote database.

For Java and .NET applications, this script must return valid SQL.

See also "upload_delete table event" on page 545
 "upload_update table event" on page 560
 "upload_fetch table event" on page 547

SQL example This example is taken from the Contact sample and can be found in *Samples\MobiLink\Contact\build_consol.sql*. It handles inserts made on the *Customer* table in the remote database. The script inserts the values into a table named *Customer* in the consolidated database. The final column of the table identifies the Customer as active. The final column does not appear in the remote database.

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

Java example The following stored procedure call registers a Java method called uploadInsertTable as the script for the upload_insert table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script(
    'ver1', 'table1', 'upload_insert',
    'ExamplePackage.ExampleClass.uploadInsertTable' )
```

Following is the sample Java method uploadInsertTable. It dynamically generates an UPLOAD statement.

```
public string uploadInsertTable()
{ return("insert into" + _curTable + getCols(_curTable)
  + "values" + getQM(_curTable)); }
```

.NET example

The following stored procedure call registers a .NET method called UploadInsert as the script for the upload_insert table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1', 'upload_insert',  
    'TestScripts.Test.UploadInsert'  
)
```

Following is the C# signature for the call UploadInsert.

```
public string UploadInsert( string user )
```

upload_new_row_insert table event

Function

Conflict resolution scripts for statement-based uploads commonly require access to the old and new values of rows updated in the remote database. This event allows you to handle the new values of rows updated rows in the remote database during conflict resolution.

Parameters

| Item | Parameter |
|------|-----------|
| 1 | column 1 |
| 2 | column 2 |
| ... | ... |

Description

You can use this event to assist in developing conflict resolution procedures for statement-based updates. The event parameters hold the values for the row in the remote database before the update was carried out. It is also used to insert `INSERTED` rows in statement-based, forced-conflict mode.

A typical action for this event is to hold the row in a temporary table for use by a `resolve_conflict` script.

You can have one `upload_new_row_insert` script for each table in the remote database.

For Java and .NET applications, this script must return valid SQL.

See also

"`resolve_conflict` table event" on page 533

"`upload_update` table event" on page 560

SQL example

This example is taken from the `Contact` sample and can be found in `Samples\MobiLink>Contact\build_consol.sql`. It handles updates made on the `product` table in the remote database. The script inserts the new value of the row into a global temporary table named `product_conflict`. The final column of the table identifies the row as a new row.

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

Java example

The following stored procedure call registers a Java method called `uploadNewRowInsertTable` as the script for the `upload_new_row_insert` table event when synchronizing the script version `ver1`. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script(
    'ver1', 'table1', 'upload_new_row_insert',
    'ExamplePackage.ExampleClass.uploadNewRowInsertTable'
)
```

Following is the sample Java method `uploadNewRowInsertTable`. It dynamically generates an `UPLOAD` statement.

```
public string uploadNewRowInsertTable()  
{ return("insert into" + _curTable + "_new" +  
  getCols(_curTable) + "values" + getQM(_curTable)); }
```

upload_old_row_insert table event

Function Conflict resolution scripts for statement-based uploads commonly require access to the old and new values of rows updated in the remote database. This event allows you to handle the new values of rows updated rows in the remote database during conflict resolution.

Parameters

| Item | Parameter |
|------|-----------|
| 1 | column 1 |
| 2 | column 2 |
| ... | ... |

Description The statement based `upload_old_row_insert` script performs direct insert of column values as specified in the `upload_old_row_insert` statement. You can have one `upload_old_row_insert` script for each table in the remote database.

For Java and .NET applications, this script must return valid SQL.

See also "resolve_conflict table event" on page 533
 "upload_update table event" on page 560

SQL example This example is taken from the Contact sample and can be found in *Samples\MobiLink\Contact\build_consol.sql*. It handles updates made on the *product* table in the remote database. The script inserts the old value of the row into a global temporary table named *product_conflict*. The final column of the table identifies the row as an old row.

```
insert into DBA.product_conflict( id, name, size,
quantity, unit_price, row_type )
values( ?, ?, ?, ?, ?, 'O' )
```

Java example

The following stored procedure call registers a Java method called `uploadOldRowInsertTable` as the script for the `upload_old_row_insert` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script(
'ver1', 'table1', 'upload_old_row_insert',
'ExamplePackage.ExampleClass.uploadNewRowInsertTable'
)
```

Following is the sample Java method `uploadOldRowInsertTable`. It dynamically generates an `UPLOAD` statement.

```
public string uploadOldRowInsertTable()
{ old" + getCols(_curTable) +
  "values" + getQM(_curTable)); }
```

upload_statistics connection event

Function Tracks synchronization statistics for upload operations.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|--------------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | warnings | INTEGER |
| 3 | errors | INTEGER |
| 4 | inserted_rows | INTEGER |
| 5 | deleted_rows | INTEGER |
| 6 | updated_rows | INTEGER |
| 7 | conflicted_inserts | INTEGER |
| 8 | conflicted_deletes | INTEGER |
| 9 | conflicted_updates | INTEGER |
| 10 | ignored_inserts | INTEGER |
| 11 | ignored_deletes | INTEGER |
| 12 | ignored_updates | INTEGER |
| 13 | bytes | INTEGER |
| 14 | deadlocks | INTEGER |

Default action None.

Description The upload_statistics event allows you to gather, for any user, statistics on upload happenings. The upload_statistics connection script is called just prior to the commit at the end of the upload transaction.

See also "download_statistics connection event" on page 479
"download_statistics table event" on page 482
"upload_statistics table event" on page 557
"synchronization_statistics connection event" on page 535
"synchronization_statistics table event" on page 537
"time_statistics connection event" on page 539

"time_statistics table event" on page 541

"MobiLink Monitor" on page 231

SQL example

The following example comes from an Oracle installation.

```
INSERT INTO upload_summary_audit (
  id, ml_user, warnings, errors, inserted_rows,
  deleted_rows, updated_rows, conflicted_inserts,
  conflicted_deletes, conflicted_updates,
  bytes, ignored_inserts, ignored_deletes,
  ignored_updates, bytes, deadlocks)
VALUES (usa_audit.nextval,?,?,?,?,?,?,?,?,?,?,?,?,?)
```

Once statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

Java example

The following stored procedure call registers a Java method called `uploadStatisticsConnection` as the script for the `upload_statistics` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_connection_script( 'ver1', 'upload_statistics',
'ExamplePackage.ExampleClass.uploadStatisticsConnection' )
```

Following is the sample Java method `uploadStatisticsConnection`. It logs some statistics to the MobiLink output log. (This might be useful at development time but would slow down a production server.)

```
public String uploadStatisticsConnection(
  String user,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictedInserts,
  int conflictedDeletes,
  int conflictedUpdates,
  int ignoredInserts,
  int ignoredDeletes,
  int ignoredUpdates,
  int bytes,
  int deadlocks
)
{
  java.lang.System.out.println( "updated rows: " +
    updatedRows );
}
```

.NET example

The following stored procedure call registers a .NET method called `UploadStats` as the script for the `upload_statistics` connection event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_connection_script( 'ver1',  
    'upload_statistics',  
    'TestScripts.Test.UploadStats'  
)
```

Following is the C# signature for the call UploadStats.

```
public void UploadStats(  
    string user,  
    int warnings,  
    int errors,  
    int insertedRows,  
    int deletedRows,  
    int updatedRows,  
    int conflictInserts,  
    int conflictDeletes,  
    int conflictUpdates,  
    int ignoredInserts,  
    int ignoredDeletes,  
    int ignoredUpdates,  
    int bytes,  
    int deadlocks )
```

upload_statistics table event

Function Tracks synchronization statistics for upload operations for a specific table.

Parameters In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, you should use the appropriate corresponding data type. See "SQL-Java data types" on page 171 and "SQL-.NET data types" on page 195.

Event parameters are optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you want to use parameter 2.

| Item | Parameter | Description |
|------|--------------------|--------------|
| 1 | ml_username | VARCHAR(128) |
| 2 | table | VARCHAR(128) |
| 3 | warnings | INTEGER |
| 4 | errors | INTEGER |
| 5 | inserted_rows | INTEGER |
| 6 | deleted_rows | INTEGER |
| 7 | updated_rows | INTEGER |
| 8 | conflicted_inserts | INTEGER |
| 9 | conflicted_deletes | INTEGER |
| 10 | conflicted_updates | INTEGER |
| 11 | ignored_inserts | INTEGER |
| 12 | ignored_deletes | INTEGER |
| 13 | ignored_updates | INTEGER |
| 14 | bytes | INTEGER |
| 15 | deadlocks | INTEGER |

Default action None.

Description The upload_statistics event allows you to gather, for any user, vital statistics on synchronization happenings as they apply to any table. The upload_statistics table script is called just prior to the commit at the end of the upload transaction.

See also "download_statistics connection event" on page 479
 "upload_statistics connection event" on page 554
 "upload_statistics table event" on page 557

"synchronization_statistics connection event" on page 535

"synchronization_statistics table event" on page 537

"time_statistics connection event" on page 539

"time_statistics table event" on page 541

"MobiLink Monitor" on page 231

SQL Example

The following example comes from an Oracle installation.

```
INSERT INTO upload_tables_audit (
  id, user_name, table, warnings, errors,
  inserted_rows, deleted_rows, updated_rows,
  conflicted_inserts, conflicted_deletes,
  conflicted_updates, ignored_inserts, ignored_deletes,
  ignored_updates, bytes, deadlocks)
VALUES ( ut_audit.nextval,
  ?,?,?,,?,?,,?,?,,?,?,,?,?)
```

Once statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

Java example

The following stored procedure call registers a Java method called `uploadStatisticsTable` as the script for the `upload_statistics` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script(
  'ver1', 'table1', 'upload_statistics',
  'ExamplePackage.ExampleClass.uploadStatisticsTable' )
```

Following is the sample Java method `uploadStatisticsTable`. It logs some statistics to the MobiLink output log. (This might be useful at development time but would slow down a production server.)

```
public String uploadStatisticsTable(
  String user,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictedInserts,
  int conflictedDeletes,
  int conflictedUpdates,
  int ignoredInserts,
  int ignoredDeletes,
  int ignoredUpdates,
  int bytes,
  int deadlocks
)
{
  java.lang.System.out.println( "updated rows: " +
    updatedRows );
}
```

.NET example

The following stored procedure call registers a .NET method called UploadTableStats as the script for the upload_statistics table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1', 'upload_statistics',  
    'TestScripts.Test.UploadTableStats'  
)
```

Following is the C# signature for the call UploadTableStats.

```
public void UploadTableStats(  
    string user,  
    string table,  
    int warnings,  
    int errors,  
    int insertedRows,  
    int deletedRows,  
    int updatedRows,  
    int conflictInserts,  
    int conflictDeletes,  
    int conflictUpdates,  
    int ignoredInserts,  
    int ignoredDeletes,  
    int ignoredUpdates,  
    int bytes,  
    int deadlocks )
```

upload_update table event

Function Provides an event that the MobiLink synchronization server uses during processing of the upload stream to handle rows updated at the remote database.

| Parameters | Clause | Parameters |
|------------|--------|---------------------------------------|
| | SET | column 11 column 22 ... |
| | WHERE | primary key 1 primary key 2 ... |

Description The statement-based upload_update script performs direct update of column values as specified in the upload_update statement.

The WHERE clause must include all of the primary key columns that are being synchronized. The SET clause must contain all of the non-primary key columns that are being synchronized.

You use as many non-primary key columns in your SET clause as exist in the table, and MobiLink will send the correct number of column values. Similarly, in the WHERE clause, you can have any number of primary keys, but all must be specified here, and MobiLink will send the correct values. MobiLink sends these column values and primary key values in the order the columns or primary keys appear in a MobiLink report of your schema. You can use the -vh option to determine the column ordering for this table schema.

You can have one upload_update script for each table in the remote database. For Java and .NET applications, this script must return valid SQL.

See also "upload_delete table event" on page 545
"upload_fetch table event" on page 547
"upload_insert table event" on page 549

SQL example This example is taken from the Contact sample and can be found in *Samples\MobiLink\Contact\build_consol.sql*. It handles updates made to the *Customer* table in the remote database. The script updates the values in a table named *Customer* in the consolidated database.

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

Java example

The following stored procedure call registers a Java method called `uploadUpdateTable` as the script for the `upload_update` table event when synchronizing the script version *ver1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_java_table_script(  
    'ver1', 'table1', 'upload_update',  
    'ExamplePackage.ExampleClass.uploadUpdateTable' )
```

Following is the sample Java method `uploadUpdateTable`. It dynamically generates an `UPLOAD` statement.

```
public string uploadUpdateTable()  
{ return( genUU(_curTable) ); }
```

.NET example

The following stored procedure call registers a .NET method called `UploadUpdate` as the script for the `upload_update` table event when synchronizing the script version *ver1* and the table *table1*. This syntax is for Adaptive Server Anywhere consolidated databases.

```
call ml_add_dnet_table_script(  
    'ver1', 'table1', 'upload_update',  
    'TestScripts.Test.UploadUpdate'  
)
```

Following is the C# signature for the call `UploadUpdate`.

```
public string UploadUpdate()
```


CHAPTER 21

MobiLink SQL Statements

About this chapter

This chapter provides reference material about the SQL statements required to synchronize Adaptive Server Anywhere clients.


Contents

| Topic | Page |
|--|------|
| ALTER PUBLICATION statement | 565 |
| ALTER SYNCHRONIZATION DEFINITION statement (deprecated) | 566 |
| ALTER SYNCHRONIZATION SITE statement (deprecated) | 567 |
| ALTER SYNCHRONIZATION SUBSCRIPTION statement | 568 |
| ALTER SYNCHRONIZATION TEMPLATE statement (deprecated) | 569 |
| ALTER SYNCHRONIZATION USER statement | 570 |
| CREATE PUBLICATION statement | 571 |
| CREATE SYNCHRONIZATION DEFINITION statement (deprecated) | 572 |
| CREATE SYNCHRONIZATION SITE statement (deprecated) | 573 |
| CREATE SYNCHRONIZATION SUBSCRIPTION statement | 574 |
| CREATE SYNCHRONIZATION TEMPLATE statement (deprecated) | 575 |
| CREATE SYNCHRONIZATION USER statement | 576 |
| DROP PUBLICATION statement | 577 |
| DROP SYNCHRONIZATION DEFINITION statement (deprecated) | 578 |
| DROP SYNCHRONIZATION SITE statement (deprecated) | 579 |
| DROP SYNCHRONIZATION SUBSCRIPTION statement | 580 |
| DROP SYNCHRONIZATION TEMPLATE statement (deprecated) | 581 |
| DROP SYNCHRONIZATION USER statement [MobiLink] | 582 |
| START SYNCHRONIZATION DELETE statement | 583 |
| STOP SYNCHRONIZATION DELETE statement | 584 |

ALTER PUBLICATION statement

Function

Use this statement to alter a publication. A publication identifies synchronized data in a Adaptive Server Anywhere remote database.

 For complete documentation of this statement, see "ALTER PUBLICATION statement" on page 216 of the book *ASA SQL Reference Manual*.

Syntax

ALTER PUBLICATION [*owner.*] *publication-name* *alterpub-clause*, ...

alterpub-clause:

ADD TABLE *article-description*
 | **MODIFY TABLE** *article-description*
 | { **DELETE** | **DROP** } **TABLE** [*owner.*] *table-name*
 | **RENAME** *publication-name*

owner, *publication-name*, *table-name* : *identifier*

article-description :

table-name [(*column-name*, ...)]
 [**WHERE** *search-condition*]
 [**SUBSCRIBE BY** *expression*]

Permissions

Must have DBA authority, or be owner of the publication. Requires exclusive access to all tables referred to in the statement.

Side effects

Automatic commit.

ALTER SYNCHRONIZATION DEFINITION statement (deprecated)

Function

Use this statement to alter a synchronization definition. This command is deprecated. Please use synchronization publications and subscriptions instead.

🔗 For complete documentation of this statement, see "ALTER SYNCHRONIZATION DEFINITION statement" on page 222 of the book *ASA SQL Reference Manual*.

Syntax

ALTER SYNCHRONIZATION DEFINITION *sync-def-name*

```
[ SITE sync-site-name, ]  
[ TYPE sync-type, ]  
[ ADDRESS network-parameters, ]  
[ ADD OPTION parameter=value, ... ]  
[ MODIFY OPTION parameter=value, ... ]  
[ DELETE OPTION parameter, ...  
  | DELETE ALL OPTION, ]  
[ RENAME new-sync-def-name, ]  
[ ADD TABLE article-description, ... ]  
[ MODIFY TABLE article-description, ... ]  
[ DELETE TABLE table-name, ... ]
```

network-parameters : *string*

article-description :
 table-name [(*column-name*, ...)]
 [**WHERE** *search-condition*]

value : *string* | *integer*

Permissions

Must have DBA authority. Requires exclusive access to all tables referred to in the statement.

Side effects

Automatic commit.

ALTER SYNCHRONIZATION SITE statement (deprecated)

Function

Use this statement to alter a site within a MobiLink reference database, to be used when extracting Adaptive Server Anywhere remote databases with the *mlxtract* utility. This command is deprecated. Please use synchronization publications and subscriptions instead.

🔗 For complete documentation of this statement, see "ALTER SYNCHRONIZATION SITE statement [MobiLink]" on page 225 of the book *ASA SQL Reference Manual*.

Syntax

```
ALTER SYNCHRONIZATION SITE sync-site-name
[ RENAME new-sync-site-name, ]
[ TYPE sync-type, ]
[ ADDRESS network-parameters, ]
[ ADD OPTION parameter=value, ... ]
[ MODIFY OPTION parameter=value, ... ]
[ DELETE OPTION parameter, ...] DELETE ALL OPTION ]
```

network-parameters : *string*

value : *string* | *integer*

Permissions

Must have DBA authority.

Side effects

Automatic commit.

ALTER SYNCHRONIZATION SUBSCRIPTION statement

Description

Use this statement in an Adaptive Server Anywhere remote database to alter the properties of a subscription of a MobiLink user to a publication.

For complete documentation of this statement, see "ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" on page 227 of the book *ASA SQL Reference Manual*.

Syntax

ALTER SYNCHRONIZATION SUBSCRIPTION

```
TO publication-name
[ FOR ml_username, ... ]
[ TYPE sync-type ]
[ ADDRESS network-parameters ]
[ ADD OPTION option=value, ... ]
[ MODIFY OPTION option=value, ... ]
[ DELETE { ALL OPTION | OPTION option=value, ... } ]
```

ml_username: identifier

network-parameters: string

sync-type: http | https | tcpip | ActiveSync

value: string | integer

Side effects

Automatic commit.

Permissions

Must have DBA authority. Requires exclusive access to all tables referred to in the publication.

ALTER SYNCHRONIZATION TEMPLATE statement (deprecated)

Function

Use this statement to alter a template within a MobiLink reference database, to be used when extracting Adaptive Server Anywhere remote databases with the *mlxtract* utility. This command is deprecated. Please use synchronization publications and subscriptions instead.

🔗 For complete documentation of this statement, see "ALTER SYNCHRONIZATION TEMPLATE statement [MobiLink]" on page 229 of the book *ASA SQL Reference Manual*.

Syntax

ALTER SYNCHRONIZATION TEMPLATE *sync-template-name*,
 [**TYPE** *sync-type*,]
 [**ADDRESS** *network-parameters*,]
 [**ADD OPTION** *parameter=value*, ...]
 [**MODIFY OPTION** *parameter=value*, ...]
 [**DELETE OPTION** *parameter*, ...] **DELETE ALL OPTION**,]
 [**RENAME** *new-sync-def-name*,]
 [**ADD TABLE** *article-description*, ...]
 [**MODIFY TABLE** *article-description*, ...]
 [**DELETE TABLE** *table-name*, ...]

article-description:

table-name [(*column-name*, ...)]
 [**WHERE** *search-condition*]

value:

string | *integer*

Permissions

Must have DBA authority. Requires exclusive access to all tables referred to in the statement.

Side effects

Automatic commit.

ALTER SYNCHRONIZATION USER statement

Description

Use this statement in an Adaptive Server Anywhere remote database to alter the properties of a MobiLink user.

🔗 For complete documentation of this statement, see "ALTER SYNCHRONIZATION USER statement [MobiLink]" on page 231 of the book *ASA SQL Reference Manual*.

Syntax

```
ALTER SYNCHRONIZATION USER ml_username
[ TYPE sync-type ]
[ ADDRESS network-parameters ]
[ ADD OPTION option=value, ... ]
[ MODIFY OPTION option=value, ... ]
[ DELETE { ALL OPTION | OPTION option } ]
```

ml_username: *identifier*

network-parameters: *string*

sync-type: **http** | **https** | **tcpip** | **ActiveSync**

value: *string* | *integer*

Permissions

Must have DBA authority. Requires exclusive access to all tables referred to in the publication.


Side effects

Automatic commit.

CREATE PUBLICATION statement

Function

Use this statement to create a publication. In MobiLink a publication identifies synchronized data in a Adaptive Server Anywhere remote database.

 For complete documentation of this statement, see "CREATE PUBLICATION statement" on page 314 of the book *ASA SQL Reference Manual*.

Syntax

CREATE PUBLICATION [*owner*.]*publication-name*
(**TABLE** *article-description*, ...)

owner, publication-name : *identifier*

article-description:

table-name [(*column-name*, ...)]

[**WHERE** *search-condition*]

[**SUBSCRIBE BY** *expression*]

Permissions

Must have DBA authority. Requires exclusive access to all tables referred to in the statement.


Side effects

Automatic commit.

CREATE SYNCHRONIZATION DEFINITION statement (deprecated)

Function

Use this statement to specify how to register with the MobiLink synchronization server and to identify the contents that are to be uploaded from the remote database to the consolidated database. This command is deprecated. Please use synchronization publications and subscriptions instead.

 For complete documentation of this statement, see "CREATE SYNCHRONIZATION DEFINITION statement [MobiLink]" on page 326 of the book *ASA SQL Reference Manual*.

Syntax

CREATE SYNCHRONIZATION DEFINITION *sync-def-name*

SITE *ml_username*
[**TYPE** *sync-type*]
ADDRESS *network-parameters*
[**OPTION** *parameter=value*, ...]
(**TABLE** *article-description*, ...)

ml_username: identifier

network-parameters: string

sync-type: **http** | **https** | **tcpip**

value: string | integer

article-description:

table-name [(*column-name*, ...)]
[**WHERE** *search-condition*]

Permissions

Must have DBA authority. Requires exclusive access to all tables named in the article description.

Side effects

Automatic commit.

CREATE SYNCHRONIZATION SITE statement (deprecated)

Function

Use this statement to create a site within a MobiLink reference database, to be used when extracting Adaptive Server Anywhere remote databases with the *mlxtract* utility. This command is deprecated. Please use synchronization publications and subscriptions instead.

🔗 For complete documentation of this statement, see "CREATE SYNCHRONIZATION SITE statement [MobiLink]" on page 328 of the book *ASA SQL Reference Manual*.

Syntax

```
CREATE SYNCHRONIZATION SITE ml_username
USING sync-template-name
[ TYPE sync-type ]
[ ADDRESS network-parameters ]
[ OPTION option=value, ... ]
```

ml_username, *sync-template-name*: *identifier*

network-parameters: *string*

sync-type: **http** | **https** | **tcpip**

value: *string* | *integer*

Permissions

Must have DBA authority.

Side effects

Automatic commit.

CREATE SYNCHRONIZATION SUBSCRIPTION statement

Function Use this statement in an Adaptive Server Anywhere remote database to subscribe a MobiLink user to a publication.

For complete documentation of this statement, see "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" on page 331 of the book *ASA SQL Reference Manual*.

Syntax

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO publication-name  
  [ FOR ml_username, ... ]  
  [ TYPE sync-type ]  
  [ ADDRESS network-parameters ]  
  [ OPTION option=value, ... ]
```

ml_username: identifier

network-parameters: string

sync-type: **http** | **https** | **tcpip** | **ActiveSync**

value: string | integer

Permissions Must have DBA authority. Requires exclusive access to all tables referred to in the publication.

Side effects Automatic commit.

CREATE SYNCHRONIZATION TEMPLATE statement (deprecated)

Function

Use this statement to create a template within a MobiLink reference database, to be used when extracting Adaptive Server Anywhere remote databases with the *mlxtract* utility. This command is deprecated. Please use synchronization publications and subscriptions instead.

🔗 For complete documentation of this statement, see "CREATE SYNCHRONIZATION TEMPLATE statement [MobiLink]" on page 333 of the book *ASA SQL Reference Manual*.

Syntax

CREATE SYNCHRONIZATION TEMPLATE *sync-template-name*

[**TYPE** *sync-type*]
ADDRESS *network-parameters*
 [**OPTION** *option=value*]
 (**TABLE** *article-description*, ...)

network-parameters : *string*

article-description :

table-name [(*column-name*, ...)]
 [**WHERE** *search-condition*]

value : *string* | *integer*

Permissions

Must have DBA authority. Requires exclusive access to all tables referred to in the statement.

Side effects

Automatic commit.

CREATE SYNCHRONIZATION USER statement

Description

Use this statement in an Adaptive Server Anywhere remote database to create a synchronization user.

🔗 For complete documentation of this statement, see "CREATE SYNCHRONIZATION USER statement [MobiLink]" on page 335 of the book *ASA SQL Reference Manual*.

Syntax

```
CREATE SYNCHRONIZATION USER ml_username  
  [ TYPE sync-type ]  
  [ ADDRESS network-parameters ]  
  [ OPTION option=value, ... ]
```

ml_username: *identifier*

network-parameters: *string*

sync-type: **http** | **https** | **tcpip** | **ActiveSync**

value:
 string | *integer*

Permissions


Must have DBA authority.

Side effects

Automatic commit.

DROP PUBLICATION statement

Function Use this statement to drop a publication. In MobiLink a publication identifies synchronized data in a Adaptive Server Anywhere remote database.

 For complete documentation of this statement, see "DROP PUBLICATION statement" on page 402 of the book *ASA SQL Reference Manual*.

Syntax **DROP PUBLICATION** [*owner*.]*publication-name*
owner, publication-name : identifier

Permissions Must have DBA authority.

Side effects Automatic commit. All subscriptions to the publication are dropped.

DROP SYNCHRONIZATION DEFINITION statement (deprecated)

Function

Use this statement to drop a synchronization definition. This command is deprecated. Definitions and sites have been replaced with synchronization publications and subscriptions.

🔗 For complete documentation of this statement, see "DROP SYNCHRONIZATION DEFINITION statement [MobiLink]" on page 408 of the book *ASA SQL Reference Manual*.

Syntax

DROP SYNCHRONIZATION DEFINITION *sync-def-name*

Permissions

Must have DBA authority.

Side effects

Automatic commit.

DROP SYNCHRONIZATION SITE statement (deprecated)

Function Use this statement to drop a specific synchronization site. This command is deprecated. Definitions and sites have been replaced with synchronization publications and subscriptions.

🔗 For complete documentation of this statement, see "DROP SYNCHRONIZATION SITE statement [MobiLink]" on page 409 of the book *ASA SQL Reference Manual*.

Syntax **DROP SYNCHRONIZATION SITE** *sync-site-name*

Permissions Must have DBA authority.

Side effects Automatic commit.

DROP SYNCHRONIZATION SUBSCRIPTION statement

Function Use this statement to drop a synchronization subscription within a MobiLink remote database or a MobiLink reference database.

🔗 For complete documentation of this statement, see "DROP SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" on page 410 of the book *ASA SQL Reference Manual*.

Syntax **DROP SYNCHRONIZATION SUBSCRIPTION**
TO *publication-name*
[FOR *ml_username***, ...]**

ml_username: *identifier*

Side Effects Automatic commit.

Permissions Must have DBA authority. Requires exclusive access to all tables referred to in the publication.

DROP SYNCHRONIZATION TEMPLATE statement (deprecated)

| | |
|---------------------|---|
| Function | Use this statement to drop a synchronization template. This command is deprecated. Please use synchronization publications and subscriptions instead. |
| Syntax | DROP SYNCHRONIZATION TEMPLATE <i>sync-template-name</i> |
| Permissions | Must have DBA authority. |
| Side effects | Automatic commit. Dropping a synchronization template implicitly drops all sites using that template. |

DROP SYNCHRONIZATION USER statement [MobiLink]

Description

Use this statement to drop a synchronization user from a MobiLink remote database.

🔗 For complete documentation of this statement, see "DROP SYNCHRONIZATION USER statement [MobiLink]" on page 412 of the book *ASA SQL Reference Manual*.

Syntax

DROP SYNCHRONIZATION USER *ml_username*, ...

ml_username: *identifier*

Permissions


Must have DBA authority. Requires exclusive access to all tables referred to in the publication.

Side Effects

All subscriptions associated with the user are also deleted.

START SYNCHRONIZATION DELETE statement

Function Use this statement to restart logging of deletes for MobiLink synchronization.

 For complete documentation of this statement, see "START SYNCHRONIZATION DELETE statement [MobiLink]" on page 556 of the book *ASA SQL Reference Manual*.


Syntax **START SYNCHRONIZATION DELETE**

Permissions Must have DBA authority.

Side Effects None.

STOP SYNCHRONIZATION DELETE statement

Function Use this statement to temporarily stop logging of deletes for MobiLink synchronization.

 For complete documentation of this statement, see "STOP SYNCHRONIZATION DELETE statement [MobiLink]" on page 563 of the book *ASA SQL Reference Manual*.

Syntax **STOP SYNCHRONIZATION DELETE**

Permissions Must have DBA authority.

Side Effects None.

CHAPTER 22

Stored Procedures

About this chapter

This chapter provides information about the MobiLink pre-defined stored procedures. There are two types of stored procedure:

- ◆ **Server stored procedures** facilitate management of synchronization scripts using SQL statements. They are stored in the consolidated database.
- ◆ **Client stored procedures** perform specific tasks during synchronization on an Adaptive Server Anywhere client. These are called client event-hook procedures.

Contents

| Topic | Page |
|--|------|
| Stored procedures to add or delete scripts | 586 |
| Client event-hook procedures | 592 |

Stored procedures to add or delete scripts

You must add synchronization scripts to system tables in the consolidated database before you can use them. The following stored procedures add synchronization scripts to the consolidated database. They can also be used to delete scripts.

Notes

- ◆ When you add a script using a stored procedure, the script is a string. Any strings within the script need to be escaped. For Adaptive Server Anywhere, each quotation mark (') needs to be doubled so as not to terminate the string.
- ◆ You cannot use stored procedures to add scripts longer than 255 bytes to Adaptive Server Enterprise 11.5 or earlier. Instead, use Sybase Central or direct insertion to define longer scripts.
- ◆ 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_`.

ml_add_connection_script

Function Use this stored procedure to add or delete SQL connection scripts in the consolidated database.

Parameters

| Item | Parameter | Description |
|------|-----------|--|
| 1 | version | CHAR(128) |
| 2 | event | CHAR(128) |
| 3 | script | For Adaptive Server Anywhere and MS SQL Server, this is TEXT. For ASE, this is VARCHAR(16384). For ASE prior to 12.5, this is VARCHAR(255). For DB2, this is VARCHAR(4000). For Oracle, this is VARCHAR. |

Description To delete a connection script, set the script parameter to NULL.

When you add a script, the script is inserted into the *ml_script* table and the appropriate references are defined to associate the script with the event and script version that you specify. If the version name is new, it is automatically inserted into the *ml_version* table.

See also "Adding and deleting scripts in your consolidated database" on page 63

"ml_add_table_script" on page 587
 "ml_add_dnet_connection_script" on page 588
 "ml_add_dnet_table_script" on page 589
 "ml_add_java_connection_script" on page 589
 "ml_add_java_table_script" on page 590

Example

The following statement adds a connection script associated with the begin_synchronization event to an Adaptive Server Anywhere consolidated database. The script itself is the single statement that sets the @EmployeeID variable.

```
call ml_add_connection_script( 'custdb',
    'begin_synchronization',
    'set @EmployeeID = ?' )
```

ml_add_table_script**Function**

Use this stored procedure to add or delete SQL table scripts in the consolidated database.

Parameters

| Item | Parameter | Description |
|------|------------|--|
| 1 | version | VARCHAR(128) |
| 2 | table_name | VARCHAR(128) |
| 3 | event | VARCHAR(128) |
| 4 | script | For Adaptive Server Anywhere and MS SQL Server, this is TEXT. For ASE, this is VARCHAR(16384). For ASE prior to 12.5, this is VARCHAR(255). For DB2, this is VARCHAR(4000). For Oracle, this is VARCHAR. |

Description

To delete a table script, set the script parameter to NULL.

When you add a script, the script is inserted into the *ml_script* table and the appropriate references are defined to associate the script with the table, event and script version that you specify. If the version name is new, it is automatically inserted into the *ml_version* table.

See also

"Adding and deleting scripts in your consolidated database" on page 63
 "ml_add_connection_script" on page 586
 "ml_add_dnet_connection_script" on page 588
 "ml_add_dnet_table_script" on page 589
 "ml_add_java_connection_script" on page 589
 "ml_add_java_table_script" on page 590

Example The following command adds a cursor script associated with the upload_cursor event on the *ULCustomer* table.

```
call ml_add_table_script( 'custdb',
    'ULCustomer',
    'upload_cursor',
    'SELECT cust_id, cust_name
      FROM ULCustomer WHERE cust_id = ?' )
```

ml_add_dnet_connection_script

Function Use this stored procedure to add or delete .NET connection scripts in the consolidated database.

Parameters

| Item | Parameter | Description |
|------|-----------|--|
| 1 | version | CHAR(128) |
| 2 | event | CHAR(128) |
| 3 | script | For Adaptive Server Anywhere and MS SQL Server, this is TEXT. For ASE, this is VARCHAR(16384). For ASE prior to 12.5, this is VARCHAR(255). For DB2, this is VARCHAR(4000). For Oracle, this is VARCHAR. |

Description To delete a connection script, set the script parameter to NULL.

The *script* value is a public method in a class in the MobiLink synchronization server classpath (for example, MyClass.MyMethod).

When you add a script, the method is associated with the event and script version that you specify. If the version name is new, it is automatically inserted into the *ml_version* table.

See also "Adding and deleting scripts in your consolidated database" on page 63
"ml_add_dnet_table_script" on page 589
"ml_add_connection_script" on page 586
"ml_add_table_script" on page 587
"ml_add_java_table_script" on page 590
"Methods" on page 172

Example The following example assigns the beginDownloadConnection method of the ExampleClass class to the begin_download event.

```
call ml_add_dnet_connection_script( 'ver1',
    'begin_download',
    'ExamplePackage.ExampleClass.beginDownloadConnection' )
```

ml_add_dnet_table_script

Function Use this stored procedure to add or delete .NET table scripts in the consolidated database.

Parameters

| Item | Parameter | Description |
|------|-----------|--|
| 1 | version | VARCHAR(128) |
| 2 | table | VARCHAR(128) |
| 3 | event | VARCHAR(128) |
| 4 | script | For Adaptive Server Anywhere and MS SQL Server, this is TEXT. For ASE, this is VARCHAR(16384). For ASE prior to 12.5, this is VARCHAR(255). For DB2, this is VARCHAR(4000). For Oracle, this is VARCHAR. |

Description To delete a connection script, set the script parameter to NULL.

The *script* value is a public method in a class in the MobiLink synchronization server classpath (for example, MyClass.MyMethod).

When you add a script, the method is associated with the table, event, and script version that you specify. If the version name is new, it is automatically inserted into the *ml_version* table.

See also "Adding and deleting scripts in your consolidated database" on page 63
 "ml_add_dnet_connection_script" on page 588
 "ml_add_connection_script" on page 586
 "ml_add_table_script" on page 587
 "ml_add_java_connection_script" on page 589
 "Methods" on page 172

Example The following example assigns the empDownloadCursor method of the EgClass class to the download_cursor event for the table emp.

```
call ml_add_dnet_table_script( 'ver1', 'emp',
                             'download_cursor', EgPackage.EgClass.empDownloadCursor )
```

ml_add_java_connection_script

Function Use this stored procedure to add or delete Java connection scripts in the consolidated database.

| Parameters | Item | Parameter | Description |
|------------|------|-----------|--|
| | 1 | version | CHAR(128) |
| | 2 | event | CHAR(128) |
| | 3 | script | For Adaptive Server Anywhere and MS SQL Server, this is TEXT. For ASE, this is VARCHAR(16384). For ASE prior to 12.5, this is VARCHAR(255). For DB2, this is VARCHAR(4000). For Oracle, this is VARCHAR. |

Description

To delete a connection script, set the script parameter to NULL.

The *script* value is a public method in a class in the MobiLink synchronization server classpath (for example, MyClass.MyMethod).

When you add a script, the method is associated with the event and script version that you specify. If the version name is new, it is automatically inserted into the *ml_version* table.

See also

"Adding and deleting scripts in your consolidated database" on page 63
"ml_add_connection_script" on page 586
"ml_add_table_script" on page 587
"ml_add_dnet_connection_script" on page 588
"ml_add_dnet_table_script" on page 589
"ml_add_java_table_script" on page 590
"Methods" on page 172

Example

The following example is taken from the *Samples\MobiLink\JavaAuthentication* sample. It assigns the endConnection method of the CustEmpScripts class to the end_connection event.

```
call ml_add_java_connection_script( 'ver1',  
    'end_connection',  
    'CustEmpScripts.endConnection' )
```

ml_add_java_table_script

Function

Use this stored procedure to add or delete Java table scripts in the consolidated database.

Parameters

| Item | Parameter | Description |
|------|-----------|--|
| 1 | version | VARCHAR(128) |
| 2 | table | VARCHAR(128) |
| 3 | event | VARCHAR(128) |
| 4 | script | For Adaptive Server Anywhere and MS SQL Server, this is TEXT. For ASE, this is VARCHAR(16384). For ASE prior to 12.5, this is VARCHAR(255). For DB2, this is VARCHAR(4000). For Oracle, this is VARCHAR. |

Description

To delete a connection script, set the script parameter to NULL.

The *script* value is a public method in a class in the MobiLink synchronization server classpath (for example, MyClass.MyMethod).

When you add a script, the method is associated with the table, event, and script version that you specify. If the version name is new, it is automatically inserted into the *ml_version* table.

See also

"Adding and deleting scripts in your consolidated database" on page 63

"ml_add_connection_script" on page 586

"ml_add_table_script" on page 587

"ml_add_dnet_connection_script" on page 588

"ml_add_dnet_table_script" on page 589

"ml_add_java_connection_script" on page 589

"Methods" on page 172

Example

The following example is taken from the *Samples\MobiLink\JavaAuthentication* sample. It assigns the empDownloadCursor method of the CustEmpScripts class to the download_cursor event for the table emp.

```
call ml_add_java_table_script( 'ver1', 'emp',
    'download_cursor', 'CustEmpScripts.empDownloadCursor' )
```

Client event-hook procedures

The following stored procedures provide the interface for customizing synchronization at Adaptive Server Anywhere clients.

Tip

If a `*_begin` hook executes successfully, the corresponding `*_end` hook is called regardless of any error that occurs afterwards. If the `*_begin` hook is not defined, but you have defined an `*_end` hook, then the `*_end` hook is called unless an error occurs prior the point in time where the `*_begin` hook would normally be called.

☞ For more information about using client event hooks, see "Customizing the client synchronization process" on page 157.

The `dbmsync` utility 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.

Caution

Do not perform any COMMIT or ROLLBACK operations in event-hook procedures. The procedures are executed on the same connection as the synchronization, and a COMMIT or ROLLBACK may interfere with synchronization.

The `#hook_dict` table

The `#hook_dict` table is created in the remote database immediately before a hook is called using the following CREATE statement:

```
CREATE TABLE #hook_dict(  
  name VARCHAR(128) NOT NULL UNIQUE,  
  value VARCHAR(255) NOT NULL)
```

`dbmsync` uses the `#hook_dict` table to pass values to hook functions; hook functions use the `#hook_dict` table to pass values back to `dbmsync`.

For example, if your `dbmsync` command line is like the one below:

```
dbmsync -c 'dsn=MyDsn' -n pub1, pub2 -u MyUser
```

When the `sp_hook_dbmsync_abort` hook is called the `#hook_dict` table will contain the following rows:

| Name | Value |
|------------------------------|--------|
| publication_0 | pub1 |
| publication_1 | pub2 |
| MobiLink user | MyUser |
| Abort synchronization | false |

Your abort hook can retrieve values from the *#hook_dict* table and use them to customize behavior. For example, to retrieve the MobiLink user you would use a *SELECT* statement like this:

```
SELECT value
FROM #hook_dict
WHERE name = 'MobiLink user'
```

In/out parameters can be updated by your hook to modify the behavior of *dbmsync*. For example, your hook could instruct *dbmsync* to abort synchronization by updating the *abort* synchronization row of the table using a statement like this:

```
UPDATE #hook_dict
SET value='true'
WHERE name='abort synchronization'
```

The description of each hook lists the rows in the *#hook_dict* table. Each is a member of a matched pair of cell values in the table. Given a single row, you can use a *SELECT* statement to retrieve a corresponding matched value.

sp_hook_dbmsync_abort

Function

Use this stored procedure to cancel the synchronization process.

Rows in #hook_dict table

| Name | Values | Description |
|--|------------------------|--|
| abort synchronization (in out) | True False | If you set the abort synchronization row of the <i>#hook_dict</i> table to true , then <i>dbmsync</i> terminates immediately after the event. |
| publication_n (in) | Any publication name | The publications being synchronized. There is one <i>publication_n</i> entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

| | |
|-------------|--|
| Description | <p>If a procedure of this name exists, it is called at the beginning of the sequence of events, and then again after each synchronization delay.</p> <p>Actions of this procedure are committed immediately after execution.</p> |
| See also | "Synchronization event hook sequence" on page 157 |
| Example | <p>The following procedure prevents synchronization during a scheduled maintenance hour between 19:00 and 20:00 each day.</p> <pre>create procedure sp_hook_dbmlsync_abort() begin declare down_time_start time; declare is_down_time varchar(128); set down_time_start='19:00'; if abs(datediff(hour,down_time_start,now(*))) < 1 then set is_down_time='true'; else set is_down_time='false'; end if; UPDATE #hook_dict SET value = is_down_time WHERE name = 'abort synchronization' end</pre> |

sp_hook_dbmlsync_begin

| | |
|----------|--|
| Function | Use this stored procedure to add custom actions at the beginning of the synchronization process. |
|----------|--|

Rows in #hook_dict table

| Name | Values | Description |
|----------------------|------------------------|--|
| publication_n (in) | Any publication name | The publications being synchronized. There is one publication_n entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

| | |
|-------------|--|
| Description | <p>If a procedure of this name exists, it is called at the beginning of the synchronization process.</p> <p>Actions of this procedure are committed immediately after execution.</p> |
| See also | "Synchronization event hook sequence" on page 157 |

sp_hook_dbmlsync_delay

Function

Use this stored procedure to control when synchronization takes place.

Rows in #hook_dict table

| Name | Values | Description |
|--|------------------------|--|
| delay duration (in out) | Any number of seconds | If the procedure sets the delay duration value to zero, then <i>dbmlsync</i> synchronization proceeds. A non-zero delay_duration value specifies the number of seconds before the delay hook is called again. |
| maximum accumulated delay (in out) | | The <i>maximum accumulated delay</i> specifies the maximum number of seconds delay you wish to create before each synchronization using the delay hook. <i>Dbmlsync</i> keeps track of the total delay created by all calls to the delay hook since the last synchronization. If no synchronization has occurred since <i>dbmlsync</i> started running, the total delay is calculated from the time <i>dbmlsync</i> started up. When the total delay exceeds the value of Maximum Accumulated delay, synchronization begins without any further calls to the delay hook. |
| publication_n (in) | Any publication name | The publications being synchronized. There is one <i>publication_n</i> entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description

If a procedure of this name exists, it is called before **sp_hook_dbmlsync_begin** at the beginning of the synchronization process.

Actions of this procedure are committed immediately after execution.

See also

"Synchronization event hook sequence" on page 157

Example

The following procedure delays synchronization during a scheduled maintenance hour between 19:00 and 20:00 each day.

```
create procedure sp_hook_dbmlsync_delay()
begin
  declare down_time_start time;
  declare is_down_time varchar(128);
  set down_time_start='19:00';
  if abs( datediff( minute,down_time_start,now(*) ) ) <
  60 then
    set is_down_time='10';
  else
    set is_down_time='0';
  end if;
  UPDATE #hook_dict
  SET value = is_down_time
  WHERE name = 'delay duration'
end
```

sp_hook_dbmlsync_download_begin

Function Use this stored procedure to add custom actions at the beginning of the download stage of the synchronization process.

Rows in #hook_dict table

| Name | Values | Description |
|-------------------------|------------------------|--|
| publication_n (in) | Any publication name | The publications being synchronized. There is one publication_n entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description If a procedure of this name exists, it is called at the beginning of the download stage of the synchronization process.

Actions of this procedure are committed or rolled back when the download stream is committed or rolled back.

See also "Synchronization event hook sequence" on page 157

sp_hook_dbmlsync_download_com_error

Function Use this stored procedure to add custom actions when communications errors occur while reading the download stream sent by the MobiLink synchronization server.

Rows in #hook_dict table

| Name | Values | Description |
|--------------------------------|------------------------|---|
| table name (in) | A table name | The table to which operations were being applied when the error occurred. The value is an empty string if <i>dbmsync</i> is unable to identify the table. |
| publication_n (in) | Any publication name | The publications being synchronized. There is one <i>publication_n</i> entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description

If a procedure of this name exists, it is invoked when a communication error is detected during the download phase of synchronization. The download is then terminated.

This procedure executes on a separate connection, so that failures can be logged. Otherwise, the action of logging would be rolled back along with the synchronization actions. If *dbmsync* cannot establish a separate connection, the procedure is not called.

Actions of this procedure are committed immediately after execution.

See also

"Synchronization event hook sequence" on page 157

sp_hook_dbmsync_download_end**Function**

Use this stored procedure to add custom actions at the end of the download stage of the synchronization process.

Rows in #hook_dict table

| Name | Values | Description |
|--------------------------------|------------------------|---|
| publication_n (in) | Any publication name | The publications being synchronized. There is one <i>publication_n</i> entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description

If a procedure of this name exists, it is called at the end of the download stage of the synchronization process.

Actions of this procedure are committed or rolled back when the download stream is committed or rolled back.

See also "Synchronization event hook sequence" on page 157

sp_hook_dbmsync_download_fatal_sql_error

Function Take action when a synchronization download is about to be rolled back because of a database error.

Rows in #hook_dict table

| Name | Values | Description |
|---------------------------------|------------------------|---|
| table name (in) | A table name | The table to which operations were being applied when the error occurred. The value is an empty string if <i>dbmsync</i> is unable to identify the table. |
| SQL error code (in) | SQL error code | Identifies the SQL error code returned by the database when the operation failed. |
| publication_n (in) | Any publication name | The publications being synchronized. There is one <i>publication_n</i> entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description If a procedure of this name exists, it is called immediately before a synchronization download is rolled back because of a database error. This occurs whenever an SQL error is encountered that cannot be ignored, or when the *sp_hook_dbmsync_download_SQL_error* hook has already been called and has chosen not to ignore the error.

This procedure executes on a separate connection, so that failures can be logged. Otherwise, the action of logging would be rolled back along with the synchronization actions. If *dbmsync* cannot establish a separate connection, the procedure is not called.

Actions of this procedure are committed immediately after execution.

See also "Synchronization event hook sequence" on page 157
"sp_hook_dbmsync_download_sql_error" on page 601

sp_hook_dbmlsync_download_log_ri_violation

Function

Logs referential integrity violations in the download process.

Rows in #hook_dict table

| Name | Values | Description |
|------------------------------------|------------------------|--|
| publication_n (in) | Any publication name | The publications being synchronized. There is one publication_n entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |
| Foreign key table (in) | A table name | The table containing the foreign key column for which the hook is being called. |
| Primary key table (in) | A table name | The table referenced by the foreign key for which the hook is being called. |
| Role name (in) | A role name | The role name of the foreign key for which the hook is being called. |

Description

A download RI violation occurs when rows in the download stream violate foreign key relationships on the remote database. This hook allows you to log RI violations as they occur so that you can investigate their cause later.

After the download is complete, but before it is committed, dbmlsync checks for RI violations. If it finds any, it identifies a foreign key that has an RI violation and calls `sp_hook_dbmlsync_download_log_ri_violation` (if it is implemented). It then calls `sp_hook_dbmlsync_download_ri_conflict` (if it is implemented). If there is still a conflict, dbmlsync deletes the rows. This process is repeated for remaining foreign keys that have RI violations.

This hook is called only when there are RI violations involving tables that are currently being synchronized. If there are RI violations involving tables that are not being synchronized, this hook is not called and the synchronization fails.

This hook is called on a separate connection from the one that dbmlsync uses for the download. The connection used by the hook has an isolation level of 0 so that the hook can see the rows that have been applied from the download stream that are not yet committed. The actions of the hook are committed immediately after it completes so that changes made by this hook will be preserved regardless of whether the download stream is committed or rolled back.

Do not attempt to use this hook to correct RI violation problems. It should be used for logging only. Use `sp_hook_dbmsync_download_ri_violation` to resolve RI violations.

See also "sp_hook_dbmsync_download_ri_violation" on page 600

sp_hook_dbmsync_download_ri_violation

Function Allows you to resolve referential integrity violations in the download process.

Rows in #hook_dict table

| Name | Values | Description |
|------------------------------------|------------------------|--|
| publication_n (in) | Any publication name | The publications being synchronized. There is one publication_n entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |
| Foreign key table (in) | A table name | The table containing the foreign key column for which the hook is being called. |
| Primary key table (in) | A table name | The table referenced by the foreign key for which the hook is being called. |
| Role name (in) | A role name | The role name of the foreign key for which the hook is being called. |

Description A download RI violation occurs when rows in the download stream violate foreign key relationships on the remote database. This hook allows you to attempt to resolve RI violations before dbmsync deletes the rows that are causing the conflict.

After the download is complete, but before it is committed, dbmsync checks for RI violations. If it finds any, it identifies a foreign key that has an RI violation and calls `sp_hook_dbmsync_download_log_ri_violation` (if it is implemented). It then calls `sp_hook_dbmsync_download_ri_conflict` (if it is implemented). If there is still a conflict, dbmsync deletes the rows. This process is repeated for remaining foreign keys that have RI violations.

This hook is called only when there are RI violations involving tables that are currently being synchronized. If there are RI violations involving tables that are not being synchronized, this hook is not called and the synchronization fails.

This hook is called on the same connection that *dbmlsync* uses for the download. This hook should not contain any explicit or implicit commits, because they may lead to inconsistent data in the database. The actions of this hook are committed or rolled back when the download stream is committed or rolled back.

Unlike other hook actions, the operations performed during this hook are not uploaded during the next synchronization.

See also

"*sp_hook_dbmlsync_download_log_ri_violation*" on page 599

sp_hook_dbmlsync_download_sql_error

Function

Handle database errors reading the download stream sent by the MobiLink synchronization server.

Rows in #hook_dict table

| Name | Values | Description |
|---------------------------------|------------------------|---|
| table name (in) | A table name | The table to which operations were being applied when the error occurred. The value is an empty string if <i>dbmlsync</i> is unable to identify the table. |
| continue (in out) | True False | Indicates whether the error should be ignored and synchronization should continue. This parameter should be set to true to ignore the error and continue or false to call the <i>sp_hook_dbmlsync_download_fatal_sql_error</i> hook and stop synchronization. When true is returned in this field the operation that caused the SQL error is lost. |
| SQL error code (in) | SQL error code | Identifies the SQL error code returned by the database when the operation failed. |
| publication_n (in) | Any publication name | The publications being synchronized. There is one <i>publication_n</i> entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description If a procedure of this name exists, it is invoked when a database error is detected during the download phase of synchronization. The procedure is only invoked for errors where it is possible to ignore the error and continue with synchronization. For fatal errors, the *sp_hook_dbmlsync_download_fatal_SQL_error* procedure is called.

Actions of this procedure are committed or rolled back when the download stream is committed or rolled back.

See also "Synchronization event hook sequence" on page 157
"sp_hook_dbmlsync_download_fatal_sql_error" on page 598

sp_hook_dbmlsync_download_table_begin

Function Use this stored procedure to add custom actions immediately before each table is downloaded.

| Rows in #hook_dict table | Name | Values | Description |
|--------------------------|-------------------------|------------------------|--|
| | table name (in) | A table name | The table to which operations are about to be applied. |
| | publication_n (in) | Any publication name | The publications being synchronized. There is one publication_n entry for each publication being uploaded. |
| | MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description If a procedure of this name exists, it is called for each table immediately before downloaded operations are applied to that table. Actions of this procedure are committed or rolled back when the download stream is committed or rolled back.

See also "Synchronization event hook sequence" on page 157

sp_hook_dbmlsync_download_table_end

Function Use this stored procedure to add custom actions immediately after each table is downloaded.

Rows in #hook_dict table

| Name | Values | Description |
|--------------------------------|------------------------|--|
| table name (in) | A table name | The table to which operations have just been applied. |
| delete count (in) | Number of rows | The number of rows in this table deleted by the download stream. |
| upsert count (in) | Number of rows | The number of rows in this table updated or inserted by the download stream. |
| publication_n (in) | Any publication name | The publications being synchronized. There is one publication_n entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description

If a procedure of this name exists, it is called immediately after all operations in the download stream for a table have been applied.

Actions of this procedure are committed or rolled back when the download stream is committed or rolled back.

See also

"Synchronization event hook sequence" on page 157

sp_hook_dbmlsync_end**Function**

Use this stored procedure to add custom actions immediately before synchronization is complete.

| Rows in #hook_dict table | Name | Values | Description |
|--------------------------|---------------------------------|------------------------|---|
| | restart (in out) | true false | If set to true then, instead of shutting down, <i>dbmlsync</i> begins a new synchronization subject to the same scheduling parameters that applied to the synchronization it just completed. If the field is false (the default) then <i>dbmlsync</i> shuts down or restarts according to its command line arguments. |
| | exit code (in) | Any number | If set to anything other than zero (the default), this represents a synchronization error. |
| | publication_ <i>n</i> (in) | Any publication name | The publications being synchronized. There is one publication_ <i>n</i> entry for each publication being uploaded. |
| | MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description If a procedure of this name exists, it is called as the last event during synchronization.

Actions of this procedure are committed immediately after execution.

See also "Customizing the client synchronization process" on page 157
"Synchronization event hook sequence" on page 157

sp_hook_dbmlsync_logscan_begin

Function Use this stored procedure to add custom actions immediately before the transaction log is scanned for upload.

Rows in #hook_dict table

| Name | Values | Description |
|--|------------------------|--|
| starting log offset_n (in) | A number | The log offset value where scanning is to begin. There is one value for each publication being uploaded. |
| log scan retry (in) | True False | If this is the first time the transaction log has been scanned for this synchronization, the value is false; otherwise it is true. The log is scanned twice when the MobiLink synchronization server and <i>dbmlsync</i> have different information about where the scanning should begin. |
| publication_n (in) | Any publication name | The publications being synchronized. There is one <i>publication_n</i> entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description

If a procedure of this name exists, it is called immediately before *dbmlsync* scans the transaction log to assemble the upload stream.

Actions of this procedure are committed immediately after execution.

See also

"Synchronization event hook sequence" on page 157

sp_hook_dbmlsync_logscan_end**Function**

Use this stored procedure to add custom actions immediately after the transaction log is scanned for upload.

Rows in #hook_dict table

| Name | Values | Description |
|--|------------------------|--|
| ending log offset (in) | A number | The log offset value where scanning ended. |
| starting log offset_n (in) | A number | The log offset value where scanning began. |
| log scan retry (in) | True False | If this is the first time the transaction log has been scanned for this synchronization, the value is false; otherwise it is true. The log is scanned twice when the MobiLink synchronization server and <i>dbmlsync</i> have different information about where the scanning should begin. |
| publication_n (in) | Any publication name | The publications being synchronized. There is one publication_n entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description

If a procedure of this name exists, it is called immediately after *dbmlsync* has scanned the transaction log to assemble the upload stream.

Actions of this procedure are committed immediately after execution.

See also

"Synchronization event hook sequence" on page 157

sp_hook_dbmlsync_upload_begin

Function

Use this stored procedure to add custom actions immediately before the transmission of the upload.

Rows in #hook_dict table

| Name | Values | Description |
|--------------------------------|------------------------|--|
| publication_n (in) | Any publication name | The publications being synchronized. There is one publication_n entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

| | |
|--------------------|---|
| Description | <p>If a procedure of this name exists, it is called immediately before <i>dbmlsync</i> sends the upload stream.</p> <p>Actions of this procedure are committed immediately after execution.</p> |
| See also | "Synchronization event hook sequence" on page 157 |

sp_hook_dbmlsync_upload_end

| | |
|-----------------|---|
| Function | Use this stored procedure to add custom actions after <i>dbmlsync</i> has verified receipt of the upload stream by the MobiLink synchronization server. |
|-----------------|---|

Rows in #hook_dict table

| Name | Values | Description |
|--------------------------------|---|---|
| failure cause (in) | See range of values in Description, below | The cause of failure of an upload. For more information, see Description. |
| upload status (in) | retry committed failed | <p>Specifies the status returned by the MobiLink synchronization server when <i>dbmlsync</i> attempts to verify receipt of the upload stream.</p> <p>retry The MobiLink synchronization server and <i>dbmlsync</i> had different values for the log offset that the upload stream should start from. The upload stream was not committed by the MobiLink synchronization server. The <i>dbmlsync</i> utility will attempt to send another upload stream starting from a new log offset.</p> <p>committed The upload stream was received by the MobiLink synchronization server, and committed.</p> <p>failed The MobiLink synchronization server did not commit the upload stream.</p> |
| publication_n (in) | Any publication name | The publications being synchronized. There is one <i>publication_n</i> entry for each publication being uploaded. |
| MobiLink user (in) | Any MobiLink user name | The MobiLink user for which you are synchronizing. |

Description

If a procedure of this name exists, it is called immediately after *dbmlsync* has sent the upload stream and received confirmation of it from the MobiLink synchronization server.

Actions of this procedure are committed immediately after execution.

The range of possible parameter values for the *failure cause* row in the *#hook_dict* table includes:

- ◆ **UPLD_ERR_COMMUNICATIONS_FAILURE** A communication error occurred.
- ◆ **UPLD_ERR_LOG_OFFSET_MISMATCH** The upload failed because of conflict between log offset stored on the remote and consolidated databases.
- ◆ **UPLD_ERR_GENERAL_FAILURE** The upload failed for an unknown reason.
- ◆ **UPLD_ERR_INVALID_USERID_OR_PASSWORD** The userid or password was incorrect.
- ◆ **UPLD_ERR_USERID_OR_PASSWORD_EXPIRED** The userid or password expired.
- ◆ **UPLD_ERR_USERID_ALREADY_IN_USE** The userid was already in use.
- ◆ **UPLD_ERR_DOWNLOAD_NOT_AVAILABLE** The upload was committed on the consolidated but an error occurred that prevented MobiLink from generating a download stream.
- ◆ **UPLD_ERR_PROTOCOL_MISMATCH** *Dbmlsync* received unexpected data from the MobiLink synchronization server.
- ◆ **UPLD_ERR_SQLCODE_n** Here, *n* is an integer. A SQL error occurred in the consolidated database. The integer specified is the SQLCODE for the error encountered.

See also

"Synchronization event hook sequence" on page 157

C H A P T E R 2 3

Utilities

About this chapter This chapter describes the MobiLink utility programs that are required to build and synchronize UltraLite applications.

 For information about the MobiLink synchronization server, see "MobiLink Synchronization Server Options" on page 379.

 For information about other Adaptive Server Anywhere utilities, see "Database Administration Utilities" on page 435 of the book *ASA Database Administration Guide*.

Contents

| Topic | Page |
|---|------|
| ActiveSync provider installation utility | 610 |
| MobiLink stop utility | 613 |
| MobiLink client database extraction utility | 614 |
| MobiLink user authentication utility | 618 |
| Certificate reader utility | 620 |
| Certificate generation utility | 621 |

ActiveSync provider installation utility

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

Syntax

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

| Options | Description |
|---------|--|
| -d | Disable the application on creation. |
| -k path | Specify the location of the desktop provider <i>dbasdesk.dll</i> . |
| -n | Register the application but do not copy it to the device. |
| -u | Uninstall the MobiLink ActiveSync provider. |
| -v path | Specify the location of the device provider <i>dbasdev.dll</i> . |

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

Description

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

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

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

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

For complete instructions on using the ActiveSync provider installation utility, see "Installing the MobiLink provider for ActiveSync" on page 301 of the book *UltraLite User's Guide*, and "Registering applications for use with ActiveSync" on page 302 of the book *UltraLite User's Guide*.

Options

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

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

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

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

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

Arguments

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

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

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

class The registered Windows class name for the application.

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

Examples

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

```
dbasinst
```

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

```
dbasinst -u
```

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

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

See also

"Using ActiveSync synchronization" on page 143

"Installing the MobiLink provider for ActiveSync" on page 301 of the book
UltraLite User's Guide

"Registering applications for use with ActiveSync" on page 302 of the book
UltraLite User's Guide

"Adding ActiveSync synchronization to your application" on page 305 of the
book *UltraLite User's Guide*

"ActiveSync parameters" on page 399 of the book *UltraLite User's Guide*

MobiLink stop utility

Stops the MobiLink synchronization server on the local machine.

Syntax

dbmlstop [*options*] [*server-name*]

| Option | Description |
|-----------------------|--|
| -f | Forced shutdown. Use if a hard shutdown does not work. |
| -h | Hard shutdown. MobiLink stops all synchronizations and exits. Some remotes may report an error. |
| -q | Quiet mode. Suppresses the banner. |
| -t <i>time</i> | Soft shutdown, with a hard shutdown done after the specified time. <i>time</i> is a number followed by D, H, M, or S (for days, hours, minutes and seconds). For example, -t 10m specifies that the server should be shut down in 10 minutes or when current synchronizations complete, whichever is sooner. D, H, M, and S are not case sensitive. |
| -w | Waits for the server to shut down before continuing. |

Parameters

Server-name If the MobiLink synchronization server is started using the **-zs** option, it must be shut down specifying the server name.

For more information, see "**-zs** option" on page 405

Description

By default (if none of **-w**, **-f**, **-h** or **-t** are specified), **dbmlstop** does a soft shutdown. This means that it stops accepting new connections and exits when the current synchronizations are complete.

MobiLink client database extraction utility

Creates an Adaptive Server Anywhere client database using another Adaptive Server Anywhere database (called the reference database) as a template.

Syntax

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

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

| Option | Description |
|------------------|---|
| -xp | Exclude stored procedures. |
| -xv | Exclude views. |
| -y | Overwrite command file without confirmation. |
| <i>directory</i> | The directory to which the files are written. This option is not needed if you use <code>-an</code> or <code>-ac</code> . |
| <i>site-name</i> | Specify which client database to generate. |

Description

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

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

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

Options

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

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

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

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

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

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

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

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

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

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

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

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

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

Extract both schema definition and triggers (-it) By default, only the schema and synchronization definition are extracted. Triggers are not extracted. This option provides causes triggers present in the reference database to be extracted also.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

See also

"Extracting remote databases" on page 149

MobiLink user authentication utility

Registers MobiLink users at the consolidated database. For Adaptive Server Anywhere remotes, the users must have previously been created at the remote databases with the CREATE SYNCHRONIZATION USER statement.

Syntax

```
dbmluser [ options ] -c "connection-string"
        { -f file | -u user [ -p password ] }
```

| Option | Description |
|------------------------|--|
| -c "keyword=value;..." | Supply database connection parameters. The connection string must give the utility permission to connect to the consolidated database using an ODBC data source. This parameter is required. |
| -d | Deletes the user name specified by -f or -u. |
| -dl | Display messages in the window or on the command line and also in the log file, if specified. |
| -f filename | Read the user names and passwords from the specified file. The file should be a text file containing one user name and password pair on each line, separated by white space. You must specify either -f or -u. |
| -o filename | Log output messages to file. |
| -os size | Limit size of output file. The size is the maximum file size for logging output messages, specified in bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively. By default, there is no size limit. The minimum size limit is 10 kb. |

| Option | Description |
|--------------------------------|---|
| -ot <i>filename</i> | Truncate the log file and then append output messages to it. The default is to send output to the screen. |
| -p <i>password</i> | Password to associate with the user. This option can only be used with -u. |
| -pc <i>collation-id</i> | Supply database collation ID for character set translation of the user name and password. This should be one of the Adaptive Server Anywhere collation labels such as those listed in "Initialization utility options" on page 467 of the book <i>ASA Database Administration Guide</i> . For machines using single-byte character sets the default is 1252LATIN1 . For machines using multi-byte character sets, the default is 932JPN . |
| -q | Run in minimized window. |
| -u <i>ml_username</i> | Specify user name to add (or delete, if used with -d). Only one user can be specified on a single command line. You must specify either -f or -u. This option is used with -p if passwords are being used. You must specify either -f or -u. |

Description

Given a user/password pair, the dbmluser utility first attempts to add the user. If the user has already been added to the consolidated database, it attempts to update the password for that user.

There are alternative ways to register user names in the consolidated database:

- ◆ Use Sybase Central.
- ◆ Specify the -zu+ command line option with dbmlsrv8. In this case, any existing MobiLink users that have not been added to the consolidated database are added when they first synchronize.

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, set the name with CREATE SYNCHRONIZATION USER and synchronize with that user name.
- ◆ For UltraLite remotes, you can either use the user_name field of the ul_synch_info structure; or in Java, use the SetUserName() method of the ULSynchInfo class before synchronizing.

See also

"Authenticating MobiLink Users" on page 251

Certificate reader utility

Use the *readcert* utility to display values within a certificate and validate the chain of certificates.

Syntax

readcert *certificate-name*

Description

The certificate you specify can be elliptic-curve or RSA.

When synchronization occurs through an ECC_TLS or RSA_TLS synchronization stream, the MobiLink synchronization server sends its certificate to the client, as well as the certificate of the entity that signed it, and so on up to a self-signed root. The client checks that the chain is valid and that it trusts the root certificate in the chain.

This utility scans an X509 authentication certificate and displays the field values. It then checks that the chain of certificates is valid. A validation error is reported if any of the certificates in the chain have expired, are in the wrong order, or are missing.

See also

"Transport-Layer Security" on page 283

Certificate generation utility

Use the gencert utility to create a new elliptic-curve or RSA certificate, or to sign a pre-generated certificate request.

For more information about security of MobiLink synchronization, see "Transport-Layer Security" on page 283.

Syntax

gencert [-c | -s] [-r] [-q]

| Option | Description |
|------------------------|---|
| -c | Generate a certificate authority certificate. |
| -q <i>request-file</i> | Sign a pre-generated certificate request. |
| -r | Generate a self-signed root certificate. |
| -s | Generate a server identity certificate. |

Description

This utility creates a new X509 certificate. When first started, it prompts whether you want to generate an elliptic-curve or RSA certificate.

If you are generating an elliptic-curve certificate, gencert generates an elliptic-curve key pair. If you are generating an RSA certificate, it prompts for a key size between 512 and 2028, and then creates a certificate using RSA.

The gencert utility then requests values for the distinguished fields. These fields include the country, state or province, locality, organization, organizational unit, and common name, the serial number, and an expiry date. It then requests the file name of a certificate that is to sign the new certificate.

If no certificate name is supplied, the new certificate becomes a root certificate. If a certificate file name is supplied, gencert reads and validates the certificate chain and requests the name of the file that contains the signer's private key. It then requests the password for that private key.

Then the utility requests the password that is to protect the new private key.

This utility writes three different types of files. One file contains only the new certificate. Another contains only the encrypted private key, and a third file contains both the certificate and the encrypted private key.

Often, not all three files are needed. For example, if the certificate is to be a certificate authority, used to sign other certificates, the file that contains only the certificates is distributed as a trusted root certificate to clients. The file containing the encrypted private key is stored securely. In this case, security is improved by storing the private key and the certificate separately, so the third file is not generated.

If, instead, the certificate is to identify a server, the encrypted private key should be stored with the certificate, so the utility writes only the file that contains both pieces of information.

If the signing certificate is not a root certificate, but is instead part of a chain, *gencert* reads and validates the entire chain before issuing the new certificate.

When generating a server identity certificate, the entire chain is always saved. Otherwise, saving the entire chain is optional.

When signing a pre-generated certificate request, *gencert* only prompts for a serial number, expiry date, the certificate and private key of the signer, and an output file for the signed certificate.

Gencert can sign any request that is generated by the *Certicom reqtool* utility or any other third party application that generates certificate requests in the appropriate format, such as the Microsoft IIS Web server or the Netscape iPlanet Web server. Following is an example of a certificate that is in the appropriate format:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBqjCCARMCQAwwAIElMAkGA1UEBhMCVVMxDTALBgNVBAGTBHRlc3QxDTALBgNV
BACTBHRlc3QxDTALBgNVBAoTBHRlc3QxDTALBgNVBAsTBHRlc3QxHzAdBgNVBAMT
Fm12YW5kZkZwLXBjLnN5YmFzZS5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJ
AoGBAKaD6a15MDIGYNGO1ctjAeF16VSVgwvPlg1z1OEMILjyAW51zDMJo1KFazxc
PtGs0AlKbJH/1EHUeJ4kp7zGuyV40ipEwgRB9NSxzza6mSKewsulR735CY8X07Z/
agfaJNGRiYEC39/SD3+bcN7NkDn250xJ6FPHYxbfcf/1EUTNAGMBAAGgADANBgkq
hkiG9w0BAQQFAAOBgQAvgNKRtSVLEUFIQUvf/abo959UBf+ZDoZzUCx1vnkUjBrA
G/zVdu2A3rqazsr17ihP0nRWnr+iFj+vK5ZP2Lg6jiFAzBxC/3w3fWYYJ6ImvodX
coYD3EuoXxWcKfiRq6AAB8S1Jcdjntz8HuLmXmWm2tNXVUIcXuEZ00ErANOPXQ==
-----END NEW CERTIFICATE REQUEST-----
```

Options

- c** Generate a certificate authority certificate. A certificate authority can be used to sign other certificates. By default, generated certificates cannot be used as certificate authorities.
- q** Sign a pre-generated certificate request. You can specify either an elliptic-curve or an RSA certificate to be signed.
- r** Generate a root certificate. A root certificate is signed only by itself. The default is to prompt for the name of a file that contains the certificate that is to sign the new, generated certificate.

-s Generate a server identity certificate, used to identify a MobiLink synchronization server, rather than a client. A server identity certificate cannot be a certificate authority, so this option is incompatible with the **-c** option. The default is to generate a client certificate.

See also

"Transport-Layer Security" on page 283

Example

The following example signs a certificate request called *certreq.txt*.

```
c:\>gencert -s -q certreq.txt
Certificate Generation Tool
Serial Number: 01
Certificate valid for how many years: 10
Enter file path of signer's certificate: rsaroot.crt
Enter file path of signer's private key: rsaroot.key
Enter password for signer's private key: test
Enter file path to save certificate: testcert.crt
Save entire chain (y/n): y
```


C H A P T E R 2 4

Data Type Conversions

About this chapter This chapter provides information about the conversion of data types must take place when a MobiLink synchronization server communicates with a consolidated database that was not made with Adaptive Server Anywhere. The following tables identify these mappings.

If you are writing synchronization scripts in .NET languages or in Java, you may need to know how to map SQL data types to Java and .NET data types. For more information, see "SQL-.NET data types" on page 195 and "SQL-Java data types" on page 171.

Contents

| Topic | Page |
|-----------------------------------|------|
| Sybase Adaptive Server Enterprise | 626 |
| IBM DB2 | 627 |
| Oracle | 629 |
| Microsoft SQL Server | 630 |

Note
Only supported data types are presented in this chapter.

Sybase Adaptive Server Enterprise

The following table identifies how MobiLink data types are mapped to Adaptive Server Enterprise data types.

| MobiLink data type | Sybase ASE data type |
|------------------------------|-----------------------------|
| bit | bit |
| tinyint | tinyint |
| smallint | smallint |
| int | int |
| integer | integer |
| decimal [defaults p=30, s=6] | numeric(30,6) |
| numeric [defaults p=30 s=6] | numeric(30,6) |
| float | real |
| real | real |
| double | float |
| smallmoney | numeric(10,4) |
| money | numeric(19,4) |
| date | datetime |
| time | datetime |
| timestamp | datetime |
| smalldatetime | datetime |
| datetime | datetime |
| char(n) | varchar(n) |
| character(n) | varchar(n) |
| varchar(n) | varchar(n) |
| character varying(n) | varchar(n) |
| long varchar | text |
| text | text |
| binary(n) | binary(n) |
| long binary | image |
| image | image |
| bigint | numeric(20,0) |

IBM DB2

The following table identifies how MobiLink data types are mapped to IBM DB2 data types.

| MobiLink data type | IBM DB2 data type |
|-------------------------------|--------------------------------------|
| bit | smallint |
| tinyint | smallint |
| smallint | smallint |
| int | int |
| integer | int |
| bigint | decimal(20,0) |
| char(1–254) | varchar(n) |
| char(255–4000) | varchar(n) |
| char(4001–32767) | long varchar |
| character(1–254) | varchar(n) |
| character(255–4000) | varchar(n) |
| character(4001–32767) | long varchar |
| varchar(1–4000) | varchar(n) |
| varchar(4001–32767) | long varchar |
| character varying(1–4000) | varchar(n) |
| character varying(4001–32767) | long varchar or CLOB(n) |
| long varchar | long varchar or CLOB(n) |
| text | long varchar |
| binary(1–4000) | varchar for bit data or BLOB(n) |
| binary(4001–32767) | long varchar for bit data or BLOB(n) |
| long binary | long varchar for bit data or BLOB(n) |
| image | long varchar for bit data or BLOB(n) |
| decimal [defaults p=30, s=6] | decimal(30,6) |
| numeric [defaults p=30 s=6] | decimal(30,6) |
| real | real |
| float | float |
| double | float |

| MobiLink data type | IBM DB2 data type |
|--------------------|-------------------|
| smallmoney | decimal(10,4) |
| money | decimal(19,4) |
| date | date |
| time | time |
| smalldatetime | timestamp |
| datetime | timestamp |
| timestamp | timestamp |

Oracle

The following table identifies how MobiLink data types are mapped to Oracle data types.

| MobiLink data type | Oracle data type |
|----------------------|--|
| bit | number(1,0) |
| tinyint | number(3,0) |
| smallint | number(5,0) |
| int | number(11,0) |
| bigint | number(20,0) |
| decimal(prec, scale) | number(prec, scale) |
| numeric(prec, scale) | number(prec, scale) |
| float | float |
| real | real |
| smallmoney | numeric(13,4) |
| money | number(19,4) |
| date | date |
| time | date |
| timestamp | date |
| smalldatetime | date |
| datetime | date |
| char(n) | if (n > 255) long else varchar(n), or CLOB(n) |
| varchar(n) | if (n > 2000) long else varchar(n), or CLOB(n) |
| long varchar | long or CLOB(n) |
| binary(n) | if (n > 255) long raw else raw(n), or BLOB(n) |
| varbinary(n) | if (n > 255) long raw else raw(n), or BLOB(n) |
| long binary | long raw |

For Oracle LONG data types to synchronize properly, you must check the Oracle **Force Retrieval of Long Columns** ODBC option in the ODBC data source configuration dialog.

Microsoft SQL Server

The following table identifies how MobiLink data types are mapped to Microsoft SQL Server data types.

| MobiLink data type | Microsoft SQL Server data type |
|------------------------------|---|
| bit | bit |
| tinyint | tinyint |
| smallint | smallint |
| int | int |
| bigint | numeric(20,0) |
| decimal [defaults p=30, s=6] | decimal(prec, scale) |
| numeric [defaults p=30 s=6] | numeric(prec, scale) |
| float | if (prec) float(prec) else float |
| real | real |
| smallmoney | smallmoney |
| money | money |
| date | datetime |
| time | datetime |
| timestamp | datetime |
| smalldatetime | datetime |
| datetime | datetime |
| char(n) | if (length > 255) text else varchar(length) |
| character(n) | varchar(n) |
| varchar(n) | if (length > 255) text else varchar(length) |
| long varchar | text |
| binary(n) | if (length > 255) image else binary(length) |
| long binary | image |
| double | float |

MobiLink Communication Error Messages

About this chapter This chapter lists MobiLink client/server communication errors, as well as their probable causes.

 The error messages are written to the MobiLink synchronization server message log and the MobiLink Adaptive Server Anywhere client message log. The error codes are returned to UltraLite clients in the **ss_error_code** member of the **stream_error** parameter.

Contents

| Topic | Page |
|---|------|
| Communication error messages sorted by code | 632 |
| Communication error messages sorted by message | 636 |
| Communication error messages sorted by constant | 640 |
| Communication error descriptions | 644 |

Communication error messages sorted by code

| Error code | Error message |
|------------|--|
| 0 | "No error or unknown error" on page 652 |
| 1 | "Invalid parameter %1!s!" on page 653 |
| 2 | "Parameter value %1!s!' is not an unsigned integer" on page 654 |
| 3 | "Parameter value %1!s!' is not an unsigned integer value or range. A range has the form NNN-NNN" on page 654 |
| 4 | "Parameter value %1!s!' is not a valid boolean value. The value must be 0 or 1" on page 653 |
| 5 | "Parameter value %1!s!' is not a valid hexadecimal value" on page 653 |
| 6 | "Unable to allocate %1!s! bytes" on page 652 |
| 7 | "Unable to parse the parameter string %1!s!" on page 655 |
| 8 | "Unable to read %1!s! bytes" on page 655 |
| 9 | "Unable to write %1!s! bytes" on page 680 |
| 10 | "An end write failed" on page 646 |
| 11 | "An end read failed" on page 645 |
| 12 | "Feature not implemented" on page 652 |
| 13 | "The operation would cause blocking" on page 680 |
| 14 | "Unable to generate a random number" on page 646 |
| 15 | "Unable to initialize the random number generator" on page 651 |
| 16 | "Unable to seed the random number generator" on page 670 |
| 17 | "Unable to create a random number object" on page 645 |
| 18 | "An error occurred during shutdown" on page 671 |
| 19 | "Unable to dequeue from the connection queue" on page 645 |
| 20 | "Invalid root certificate" on page 662 |
| 21 | "Unrecognized organization %1!s!" on page 658 |
| 22 | "Invalid certificate chain length (%1!s!)" on page 657 |
| 23 | "Certificate error (4023)" on page 661 |
| 24 | "Server certificate not trusted" on page 661 |
| 25 | "Unable to duplicate security context" on page 663 |

| Error code | Error message |
|------------|--|
| 26 | "Unable to attach the network layer to the security layer" on page 666 |
| 27 | "Internal error 4027" on page 667 |
| 28 | "Internal error 4028" on page 657 |
| 29 | "Internal error 4029" on page 657 |
| 30 | "Internal error 4030" on page 663 |
| 31 | "Internal error 4031" on page 666 |
| 32 | "Internal error 4032" on page 666 |
| 33 | "Unable to open certificate file %1!s!" on page 660 |
| 34 | "Unable to read certificates" on page 665 |
| 35 | "Unable to read the private key" on page 665 |
| 36 | "Unable to set the private key" on page 667 |
| 37 | "Unable to fetch a certificate expiry date" on page 660 |
| 38 | "Unable to copy a certificate" on page 663 |
| 39 | "Unable to add a certificate to a certificate chain" on page 656 |
| 40 | "Unable to find the trusted certificate file %1!s!" on page 669 |
| 41 | "Error reading from the trusted certificate file %1!s!" on page 670 |
| 42 | "No trusted certificates found" on page 659 |
| 43 | "Unable to allocate a certificate" on page 662 |
| 44 | "Unable to import a certificate" on page 664 |
| 45 | "Internal initialization error 4045" on page 668 |
| 46 | "Internal initialization error 4046" on page 668 |
| 47 | "Unable to set the protocol side (%1!s!)" on page 667 |
| 48 | "Unable to add a trusted certificate" on page 656 |
| 49 | "Unable to create a private key object" on page 662 |
| 50 | "A certificate has expired" on page 660 |
| 51 | "Unrecognized organization unit %1!s!" on page 659 |
| 52 | "Unrecognized common name %1!s!" on page 658 |
| 53 | "Handshake error" on page 664 |
| 54 | "Unsupported HTTP version: %1!s!" on page 651 |
| 55 | "Internal initialization error 4055" on page 668 |

| Error code | Error message |
|------------|---|
| 56 | "Internal initialization error 4056" on page 669 |
| 57 | "The host name '%!s!' could not be found" on page 676 |
| 58 | "Unable to get host by address" on page 674 |
| 59 | "Unable to determine localhost" on page 677 |
| 60 | "Unable to create a TCP/IP socket" on page 673 |
| 61 | "Unable to create a UDP socket" on page 674 |
| 62 | "Unable to bind a socket to port '%!s!'" on page 671 |
| 63 | "Unable to cleanup the socket layer" on page 672 |
| 64 | "Unable to close a socket" on page 672 |
| 65 | "Unable to connect a socket" on page 672 |
| 66 | "Unable to get a socket's local name" on page 675 |
| 67 | "Unable to get socket option number '%!s!'" on page 675 |
| 68 | "Unable to set socket option number '%!s!'" on page 678 |
| 69 | "Unable to listen on a socket. The backlog is '%!s!'" on page 676 |
| 70 | "Unable to shutdown a socket" on page 679 |
| 71 | "Unable to select a socket status" on page 678 |
| 72 | "Unable to initialize the sockets layer" on page 679 |
| 73 | "Invalid port number '%!s!'. The value must be between zero and 65535" on page 677 |
| 74 | "Unable to load the network interface library" on page 651 |
| 75 | "ActiveSync synchronization cannot be initiated by an application" on page 644 |
| 76 | "ActiveSync provider has not been installed" on page 644 |
| 77 | "The content type '%!s!' is unknown" on page 648 |
| 78 | "Client id is not available for us in HTTP header" on page 648 |
| 79 | "The HTTP buffer size specified is out of the valid range" on page 647 |
| 80 | "Extra data found in the HTTP body. '%!s!'" on page 649 |
| 81 | "Failed to read encoded CR LF" on page 648 |
| 82 | "Failed to read CR LF" on page 649 |
| 83 | "Timed out while waiting for the next HTTP request in this synchronization" on page 650 |

| Error code | Error message |
|-------------------|--|
| 84 | "Failed to read encoded chunk length" on page 647 |
| 85 | "An unexpected character was read while parsing the chunk length. %1!s!" on page 647 |
| 86 | "An error status was returned: %1!s!" on page 646 |
| 87 | "Unknown transfer encoding: %1!s!" on page 650 |
| 88 | "Unable to parse cookie: %1!s!" on page 650 |
| 89 | "Expected data from remote but current request is not a POST" on page 649 |

Communication error messages sorted by message

| Error code | Error message |
|------------|--|
| 50 | "A certificate has expired" on page 660 |
| 76 | "ActiveSync provider has not been installed" on page 644 |
| 75 | "ActiveSync synchronization cannot be initiated by an application" on page 644 |
| 11 | "An end read failed" on page 645 |
| 10 | "An end write failed" on page 646 |
| 18 | "An error occurred during shutdown" on page 671 |
| 86 | "An error status was returned: %1!s!" on page 646 |
| 85 | "An unexpected character was read while parsing the chunk length. %1!s!" on page 647 |
| 23 | "Certificate error (4023)" on page 661 |
| 78 | "Client id is not available for us in HTTP header" on page 648 |
| 41 | "Error reading from the trusted certificate file %1!s!" on page 670 |
| 89 | "Expected data from remote but current request is not a POST" on page 649 |
| 80 | "Extra data found in the HTTP body. %1!s!" on page 649 |
| 82 | "Failed to read CR LF" on page 649 |
| 81 | "Failed to read encoded CR LF" on page 648 |
| 84 | "Failed to read encoded chunk length" on page 647 |
| 12 | "Feature not implemented" on page 652 |
| 53 | "Handshake error" on page 664 |
| 27 | "Internal error 4027" on page 667 |
| 28 | "Internal error 4028" on page 657 |
| 29 | "Internal error 4029" on page 657 |
| 30 | "Internal error 4030" on page 663 |
| 31 | "Internal error 4031" on page 666 |
| 32 | "Internal error 4032" on page 666 |
| 45 | "Internal initialization error 4045" on page 668 |

| Error code | Error message |
|------------|--|
| 46 | "Internal initialization error 4046" on page 668 |
| 55 | "Internal initialization error 4055" on page 668 |
| 56 | "Internal initialization error 4056" on page 669 |
| 22 | "Invalid certificate chain length (%1!s!)" on page 657 |
| 1 | "Invalid parameter '%1!s!'" on page 653 |
| 73 | "Invalid port number %1!s!. The value must be between zero and 65535" on page 677 |
| 20 | "Invalid root certificate" on page 662 |
| 0 | "No error or unknown error" on page 652 |
| 42 | "No trusted certificates found" on page 659 |
| 4 | "Parameter value '%1!s!'" is not a valid boolean value. The value must be 0 or 1" on page 653 |
| 5 | "Parameter value '%1!s!'" is not a valid hexadecimal value" on page 653 |
| 2 | "Parameter value '%1!s!'" is not an unsigned integer" on page 654 |
| 3 | "Parameter value '%1!s!'" is not an unsigned integer value or range. A range has the form NNN-NNN" on page 654 |
| 24 | "Server certificate not trusted" on page 661 |
| 79 | "The HTTP buffer size specified is out of the valid range" on page 647 |
| 77 | "The content type '%1!s!'" is unknown" on page 648 |
| 57 | "The host name '%1!s!'" could not be found" on page 676 |
| 13 | "The operation would cause blocking" on page 680 |
| 83 | "Timed out while waiting for the next HTTP request in this synchronization" on page 650 |
| 39 | "Unable to add a certificate to a certificate chain" on page 656 |
| 48 | "Unable to add a trusted certificate" on page 656 |
| 6 | "Unable to allocate %1!s! bytes" on page 652 |
| 43 | "Unable to allocate a certificate" on page 662 |
| 26 | "Unable to attach the network layer to the security layer" on page 666 |
| 62 | "Unable to bind a socket to port %1!s!" on page 671 |
| 63 | "Unable to cleanup the socket layer" on page 672 |

| Error code | Error message |
|------------|--|
| 64 | "Unable to close a socket" on page 672 |
| 65 | "Unable to connect a socket" on page 672 |
| 38 | "Unable to copy a certificate" on page 663 |
| 60 | "Unable to create a TCP/IP socket" on page 673 |
| 61 | "Unable to create a UDP socket" on page 674 |
| 49 | "Unable to create a private key object" on page 662 |
| 17 | "Unable to create a random number object" on page 645 |
| 19 | "Unable to dequeue from the connection queue" on page 645 |
| 59 | "Unable to determine localhost" on page 677 |
| 25 | "Unable to duplicate security context" on page 663 |
| 37 | "Unable to fetch a certificate expiry date" on page 660 |
| 40 | "Unable to find the trusted certificate file %1!s!" on page 669 |
| 14 | "Unable to generate a random number" on page 646 |
| 66 | "Unable to get a socket's local name" on page 675 |
| 58 | "Unable to get host by address" on page 674 |
| 67 | "Unable to get socket option number %1!s!" on page 675 |
| 44 | "Unable to import a certificate" on page 664 |
| 15 | "Unable to initialize the random number generator" on page 651 |
| 72 | "Unable to initialize the sockets layer" on page 679 |
| 69 | "Unable to listen on a socket. The backlog is %1!s!" on page 676 |
| 74 | "Unable to load the network interface library" on page 651 |
| 33 | "Unable to open certificate file %1!s!" on page 660 |
| 88 | "Unable to parse cookie: %1!s!" on page 650 |
| 7 | "Unable to parse the parameter string %1!s!" on page 655 |
| 8 | "Unable to read %1!s! bytes" on page 655 |
| 34 | "Unable to read certificates" on page 665 |
| 35 | "Unable to read the private key" on page 665 |
| 16 | "Unable to seed the random number generator" on page 670 |
| 71 | "Unable to select a socket status" on page 678 |
| 68 | "Unable to set socket option number %1!s!" on page 678 |
| 36 | "Unable to set the private key" on page 667 |

| Error code | Error message |
|-------------------|---|
| 47 | "Unable to set the protocol side (%1!s!)" on page 667 |
| 70 | "Unable to shutdown a socket" on page 679 |
| 9 | "Unable to write %1!s! bytes" on page 680 |
| 87 | "Unknown transfer encoding: %1!s!" on page 650 |
| 52 | "Unrecognized common name %1!s!" on page 658 |
| 21 | "Unrecognized organization %1!s!" on page 658 |
| 51 | "Unrecognized organization unit %1!s!" on page 659 |
| 54 | "Unsupported HTTP version: %1!s!" on page 651 |

Communication error messages sorted by constant

| Constant | Error message |
|---------------------------------|---|
| ACTSYNC NOT INSTALLED | "ActiveSync provider has not been installed" on page 644 |
| ACTSYNC NO PORT | "ActiveSync synchronization cannot be initiated by a application" on page 644 |
| CREATE RANDOM OBJECT | "Unable to create a random number object" on page 645 |
| DEQUEUEING CONNECTION | "Unable to dequeue from the connection queue" on page 645 |
| END READ | "An end read failed" on page 645 |
| END WRITE | "An end write failed" on page 646 |
| GENERATE RANDOM | "Unable to generate a random number" on page 646 |
| HTTP BAD STATUS CODE | "An error status was returned: %1!s!" on page 646 |
| HTTP BUFFER SIZE OUT OF RANGE | "The HTTP buffer size specified is out of the valid range" on page 647 |
| HTTP CHUNK LEN BAD CHARACTER | "An unexpected character was read while parsing the chunk length. %1!s!" on page 647 |
| HTTP CHUNK LEN ENCODED MISSING | "Failed to read encoded chunk length" on page 647 |
| HTTP CLIENT ID NOT SET | "Client id is not available for us in HTTP header" on page 648 |
| HTTP CONTENT TYPE NOT SPECIFIED | "The content type %1!s!' is unknown" on page 648 |
| HTTP CRLF ENCODED MISSING | "Failed to read encoded CR LF" on page 648 |
| HTTP CRLF MISSING | "Failed to read CR LF" on page 649 |
| HTTP EXPECTED POST | "Expected data from remote but current request is not POST" on page 649 |
| HTTP EXTRA DATA END READ | "Extra data found in the HTTP body. %1!s!" on page 650 |
| HTTP NO CONTD CONNECTION | "Timed out while waiting for the next HTTP request or this synchronization" on page 650 |
| HTTP UNABLE TO PARSE COOKIE | "Unable to parse cookie: %1!s!" on page 650 |
| HTTP UNKNOWN TRANSFER ENCODING | "Unknown transfer encoding: %1!s!" on page 650 |
| HTTP VERSION | "Unsupported HTTP version: %1!s!" on page 651 |
| INIT RANDOM | "Unable to initialize the random number generator" on page 651 |

| Constant | Error message |
|-----------------------------------|--|
| LOAD NETWORK LIBRARY | "Unable to load the network interface library" on page 651 |
| MEMORY ALLOCATION | "Unable to allocate %1!s! bytes" on page 652 |
| NONE | "No error or unknown error" on page 652 |
| NOT IMPLEMENTED | "Feature not implemented" on page 652 |
| PARAMETER | "Invalid parameter %1!s!" on page 653 |
| PARAMETER NOT BOOLEAN | "Parameter value %1!s!' is not a valid boolean value must be 0 or 1" on page 653 |
| PARAMETER NOT HEX | "Parameter value %1!s!' is not a valid hexadecimal value" on page 653 |
| PARAMETER NOT UINT32 | "Parameter value %1!s!' is not an unsigned integer" on page 654 |
| PARAMETER NOT UINT32 RANGE | "Parameter value %1!s!' is not an unsigned integer in the specified range. A range has the form NNN-NNN" on page 654 |
| PARSE | "Unable to parse the parameter string %1!s!" on page 655 |
| READ | "Unable to read %1!s! bytes" on page 655 |
| SECURE ADD CERTIFICATE | "Unable to add a certificate to a certificate chain" on page 656 |
| SECURE ADD TRUSTED CERTIFICATE | "Unable to add a trusted certificate" on page 657 |
| SECURE CERTIFICATE CHAIN FUNC | "Internal error 4028" on page 657 |
| SECURE CERTIFICATE CHAIN LENGTH | "Invalid certificate chain length (%1!s!)" on page 657 |
| SECURE CERTIFICATE CHAIN REF | "Internal error 4029" on page 657 |
| SECURE CERTIFICATE COMMON NAME | "Unrecognized common name %1!s!" on page 658 |
| SECURE CERTIFICATE COMPANY NAME | "Unrecognized organization %1!s!" on page 658 |
| SECURE CERTIFICATE COMPANY UNIT | "Unrecognized organization unit %1!s!" on page 658 |
| SECURE CERTIFICATE COUNT | "No trusted certificates found" on page 659 |
| SECURE CERTIFICATE EXPIRED | "A certificate has expired" on page 660 |
| SECURE CERTIFICATE EXPIRY DATE | "Unable to fetch a certificate expiry date" on page 660 |
| SECURE CERTIFICATE FILE NOT FOUND | "Unable to open certificate file %1!s!" on page 661 |
| SECURE CERTIFICATE NOT TRUSTED | "Server certificate not trusted" on page 661 |
| SECURE CERTIFICATE REF | "Certificate error (4023)" on page 661 |
| SECURE CERTIFICATE ROOT | "Invalid root certificate" on page 662 |

| Constant | Error message |
|---|--|
| SECURE CREATE CERTIFICATE | "Unable to allocate a certificate" on page 662 |
| SECURE CREATE PRIVATE KEY OBJECT | "Unable to create a private key object" on page 662 |
| SECURE DUPLICATE CONTEXT | "Unable to duplicate security context" on page 663 |
| SECURE ENABLE NON BLOCKING | "Internal error 4030" on page 663 |
| SECURE EXPORT CERTIFICATE | "Unable to copy a certificate" on page 663 |
| SECURE HANDSHAKE | "Handshake error" on page 664 |
| SECURE IMPORT CERTIFICATE | "Unable to import a certificate" on page 664 |
| SECURE READ CERTIFICATE | "Unable to read certificates" on page 665 |
| SECURE READ PRIVATE KEY | "Unable to read the private key" on page 665 |
| SECURE SET CHAIN NUMBER | "Internal error 4032" on page 666 |
| SECURE SET CIPHER SUITES | "Internal error 4031" on page 666 |
| SECURE SET IO | "Unable to attach the network layer to the security layer" on page 666 |
| SECURE SET IO SEMANTICS | "Internal error 4027" on page 667 |
| SECURE SET PRIVATE KEY | "Unable to set the private key" on page 667 |
| SECURE SET PROTOCOL SIDE | "Unable to set the protocol side (%1!s!)" on page 667 |
| SECURE SET RANDOM FUNC | "Internal initialization error 4046" on page 668 |
| SECURE SET RANDOM REF | "Internal initialization error 4045" on page 668 |
| SECURE SET READ FUNC | "Internal initialization error 4055" on page 668 |
| SECURE SET WRITE FUNC | "Internal initialization error 4056" on page 669 |
| SECURE TRUSTED CERTIFICATE FILE NOT FOUND | "Unable to find the trusted certificate file '%1!s!'" on page 669 |
| SECURE TRUSTED CERTIFICATE READ | "Error reading from the trusted certificate file '%1!s!'" on page 670 |
| SEED RANDOM | "Unable to seed the random number generator" on page 670 |
| SHUTTING DOWN | "An error occurred during shutdown" on page 671 |
| SOCKET BIND | "Unable to bind a socket to port %1!s!" on page 671 |
| SOCKET CLEANUP | "Unable to cleanup the socket layer" on page 672 |
| SOCKET CLOSE | "Unable to close a socket" on page 672 |
| SOCKET CONNECT | "Unable to connect a socket" on page 672 |
| SOCKET CREATE TCPIP | "Unable to create a TCP/IP socket" on page 673 |
| SOCKET CREATE UDP | "Unable to create a UDP socket" on page 674 |

| Constant | Error message |
|---------------------------------|--|
| SOCKET GET HOST BY ADDR | "Unable to get host by address" on page 674 |
| SOCKET GET NAME | "Unable to get a socket's local name" on page 674 |
| SOCKET GET OPTION | "Unable to get socket option number %1s!" on page 675 |
| SOCKET HOST NAME NOT FOUND | "The host name '%1s!' could not be found" on page 675 |
| SOCKET LISTEN | "Unable to listen on a socket. The backlog is %1s!" on page 676 |
| SOCKET LOCALHOST NAME NOT FOUND | "Unable to determine localhost" on page 677 |
| SOCKET PORT OUT OF RANGE | "Invalid port number %1s!. The value must be between zero and 65535" on page 677 |
| SOCKET SELECT | "Unable to select a socket status" on page 678 |
| SOCKET SET OPTION | "Unable to set socket option number %1s!" on page 678 |
| SOCKET SHUTDOWN | "Unable to shutdown a socket" on page 679 |
| SOCKET STARTUP | "Unable to initialize the sockets layer" on page 679 |
| WOULD BLOCK | "The operation would cause blocking" on page 680 |
| WRITE | "Unable to write %1s! bytes" on page 680 |

Communication error descriptions

This section provides a full listing of error messages and descriptions.

Errors with an ODBC state marked "handled by ODBC driver" are not returned to ODBC applications, as the ODBC driver carries out the required actions.

ActiveSync provider has not been installed

| Item | Value |
|------------|--|
| Error code | 76 |
| Constant | ACTSYNC_NOT_INSTALLED (Java) STREAM_ERROR_ACTSYNC_NOT_INSTALLED (C/C++) ulStreamErrorActsycNotInstalled (Visual Basic) |

Probable cause The ActiveSync provider has not been installed. Run dbasinst to install it (see documentation for details).

ActiveSync synchronization cannot be initiated by an application

| Item | Value |
|------------|--|
| Error code | 75 |
| Constant | ACTSYNC_NO_PORT (Java) STREAM_ERROR_ACTSYNC_NO_PORT (C/C++) ulStreamErrorActsycNoPort (Visual Basic) |

Probable cause ActiveSync synchronization can only be initiated by ActiveSync itself, either by placing the device in its cradle or by selecting "Synchronize" from the ActiveSync Manager. To initiate a synchronization from an application, use the TCP/IP socket synchronization stream.

Unable to create a random number object

| Item | Value |
|------------|--|
| Error code | 17 |
| Constant | CREATE_RANDOM_OBJECT (Java) STREAM_ERROR_CREATE_RANDOM_OBJECT (C/C++) ulStreamErrorCreateRandomObject (Visual Basic) |

Probable cause The secure network layer could not create a random-number-generating object. Free up system resources, reconnect and retry the operation.

Unable to dequeue from the connection queue

| Item | Value |
|------------|---|
| Error code | 19 |
| Constant | DEQUEUEING_CONNECTION (Java) STREAM_ERROR_DEQUEUEING_CONNECTION (C/C++) ulStreamErrorDequeuingConnection (Visual Basic) |

Probable cause The MobiLink synchronization server encountered an error while attempting to get a queued connection (synchronization) request. Free up system resources. If the problem persists, restart the MobiLink synchronization server.

An end read failed

| Item | Value |
|------------|---|
| Error code | 11 |
| Constant | END_READ (Java) STREAM_ERROR_END_READ (C/C++) ulStreamErrorEndRead (Visual Basic) |

Probable cause Unable to finish a sequence of reads from the network.
See also: WRITE

An end write failed

| Item | Value |
|------------|--|
| Error code | 10 |
| Constant | END_WRITE (Java) STREAM_ERROR_END_WRITE (C/C++) ulStreamErrorEndWrite (Visual Basic) |

Probable cause Unable to finish a sequence of writes to the network.

See also: WRITE

Unable to generate a random number

| Item | Value |
|------------|--|
| Error code | 14 |
| Constant | GENERATE_RANDOM (Java) STREAM_ERROR_GENERATE_RANDOM (C/C++) ulStreamErrorGenerateRandom (Visual Basic) |

Probable cause The secure network layer requires a random number but was unable to generate one. Free up system resources, reconnect and retry the operation.

An error status was returned: '%1!s!'

| Item | Value |
|-------------|---|
| Error code | 86 |
| Constant | HTTP_BAD_STATUS_CODE (Java) STREAM_ERROR_HTTP_BAD_STATUS_CODE (C/C++) ulStreamErrorHttpBadStatusCode (Visual Basic) |
| Parameter 1 | The status line read. |

Probable cause Examine the status line to determine the cause of the failure.

The HTTP buffer size specified is out of the valid range

| Item | Value |
|------------|--|
| Error code | 79 |
| Constant | HTTP_BUFFER_SIZE_OUT_OF_RANGE (Java) STREAM_ERROR_HTTP_BUFFER_SIZE_OUT_OF_RANGE (C/C++) ulStreamErrorHttpBufferSizeOutOfRange (Visual Basic) |

Probable cause Fix the HTTP buffer size. A valid buffer size is positive and not overly large for the host platform.

An unexpected character was read while parsing the chunk length. %1!s!

| Item | Value |
|-------------|--|
| Error code | 85 |
| Constant | HTTP_CHUNK_LEN_BAD_CHARACTER (Java) STREAM_ERROR_HTTP_CHUNK_LEN_BAD_CHARACTER (C/C++) ulStreamErrorHttpChunkLenBadCharacter (Visual Basic) |
| Parameter 1 | The unexpected character. |

Probable cause Try using a fixed length HTTP body.

Failed to read encoded chunk length

| Item | Value |
|------------|--|
| Error code | 84 |
| Constant | HTTP_CHUNK_LEN_ENCODED_MISSING (Java) STREAM_ERROR_HTTP_CHUNK_LEN_ENCODED_MISSING (C/C++) ulStreamErrorHttpChunkLenEncodedMissing (Visual Basic) |

Probable cause Try using a fixed length HTTP body.

Client id is not available for us in HTTP header

| Item | Value |
|------------|--|
| Error code | 78 |
| Constant | HTTP_CLIENT_ID_NOT_SET (Java) STREAM_ERROR_HTTP_CLIENT_ID_NOT_SET (C/C++) ulStreamErrorHttpClientIdNotSet (Visual Basic) |

Probable cause The client id was not passed into the HTTP client code. Contact technical support for a fix.

The content type '%1!s!' is unknown

| Item | Value |
|-------------|---|
| Error code | 77 |
| Constant | HTTP_CONTENT_TYPE_NOT_SPECIFIED (Java) STREAM_ERROR_HTTP_CONTENT_TYPE_NOT_SPECIFIED (C/C++) ulStreamErrorHttpContentTypeNotSpecified (Visual Basic) |
| Parameter 1 | The context type. |

Probable cause An unknown content type was specified. Refer to the documentation and change the content type to one of the supported types.

Failed to read encoded CR LF

| Item | Value |
|------------|--|
| Error code | 81 |
| Constant | HTTP_CRLF_ENCODED_MISSING (Java) STREAM_ERROR_HTTP_CRLF_ENCODED_MISSING (C/C++) ulStreamErrorHttpCrlfEncodedMissing (Visual Basic) |

Probable cause The proxy you are using may not be comapatable with MobiLink. Please check your configuration.

Failed to read CR LF

| Item | Value |
|------------|---|
| Error code | 82 |
| Constant | HTTP_CRLF_MISSING (Java) STREAM_ERROR_HTTP_CRLF_MISSING (C/C++) ulStreamErrorHttpCrlfMissing (Visual Basic) |

Probable cause The proxy you are using may not be compatible with MobiLink. Please check your configuration.

Expected data from remote but current request is not a POST

| Item | Value |
|------------|--|
| Error code | 89 |
| Constant | HTTP_EXPECTED_POST (Java) STREAM_ERROR_HTTP_EXPECTED_POST (C/C++) ulStreamErrorHttpExpectedPost (Visual Basic) |

Probable cause The proxy you are using may not be compatible with MobiLink. Please check your configuration.

Extra data found in the HTTP body. %1!s!

| Item | Value |
|-------------|--|
| Error code | 80 |
| Constant | HTTP_EXTRA_DATA_END_READ (Java) STREAM_ERROR_HTTP_EXTRA_DATA_END_READ (C/C++) ulStreamErrorHttpExtraDataEndRead (Visual Basic) |
| Parameter 1 | First few characters in the extra data. |

Probable cause Extra data has been introduced into the HTTP body. This may have been added by a proxy agent, try eliminating the proxy.

Timed out while waiting for the next HTTP request in this synchronization

| Item | Value |
|------------|---|
| Error code | 83 |
| Constant | HTTP_NO_CONTD_CONNECTION (Java) STREAM_ERROR_HTTP_NO_CONTD_CONNECTION (C/C++) ulStreamErrorHttpNoContdConnection (Visual Basic) |

Probable cause The server timed out while waiting for the next HTTP request from the remote site. Determine why this request failed to reach the server or try a persistent connection.

Unable to parse cookie: '%1!s!'

| Item | Value |
|-------------|---|
| Error code | 88 |
| Constant | HTTP_UNABLE_TO_PARSE_COOKIE (Java) STREAM_ERROR_HTTP_UNABLE_TO_PARSE_COOKIE (C/C++) ulStreamErrorHttpUnableToParseCookie (Visual Basic) |
| Parameter 1 | The set cookie header. |

Probable cause Determine where the set cookie header is being corrupted.

Unknown transfer encoding: '%1!s!'

| Item | Value |
|-------------|---|
| Error code | 87 |
| Constant | HTTP_UNKNOWN_TRANSFER_ENCODING (Java) STREAM_ERROR_HTTP_UNKNOWN_TRANSFER_ENCODING (C/C++) ulStreamErrorHttpUnknownTransferEncoding (Visual Basic) |
| Parameter 1 | The unknown encoding. |

Probable cause Determine how the unknown transfer encoding is getting generated.

Unsupported HTTP version: %1!s!

| Item | Value |
|-------------|---|
| Error code | 54 |
| Constant | HTTP_VERSION (Java) STREAM_ERROR_HTTP_VERSION (C/C++) ulStreamErrorHttpVersion (Visual Basic) |
| Parameter 1 | The requested HTTP version. |

Probable cause

The requested HTTP version is unsupported. Consult the documentation and specify a supported HTTP version. At the time of publication the supported HTTP versions are 1.0 and 1.1.

Unable to initialize the random number generator

| Item | Value |
|------------|--|
| Error code | 15 |
| Constant | INIT_RANDOM (Java) STREAM_ERROR_INIT_RANDOM (C/C++) ulStreamErrorInitRandom (Visual Basic) |

Probable cause

The secure network layer could not initialize its random number generator. Free up system resources, reconnect and retry the operation.

Unable to load the network interface library

| Item | Value |
|------------|--|
| Error code | 74 |
| Constant | LOAD_NETWORK_LIBRARY (Java) STREAM_ERROR_LOAD_NETWORK_LIBRARY (C/C++) ulStreamErrorLoadNetworkLibrary (Visual Basic) |

Probable cause

The network interface library could not be found and/or loaded.

- 1) The sockets layer is properly installed. The correct network interface library (or DLL or shared object) must be present and accessible.
- 2) There are enough system resources available. Free up system resources if they are running low.

Unable to allocate %1!s! bytes

| Item | Value |
|-------------|--|
| Error code | 6 |
| Constant | MEMORY_ALLOCATION (Java) STREAM_ERROR_MEMORY_ALLOCATION (C/C++) ulStreamErrorMemoryAllocation (Visual Basic) |
| Parameter 1 | The number of bytes that was requested. |

Probable cause

The network layer was unable to allocate the given number of bytes of storage. Free up system memory and retry the operation. The technique used to free up system memory depends on the operating system and how it is configured. The simplest technique is to reduce the number of active processes. Consult your operating system documentation for details.

No error or unknown error

| Item | Value |
|------------|--|
| Error code | 0 |
| Constant | NONE (Java) STREAM_ERROR_NONE (C/C++) ulStreamErrorNone (Visual Basic) |

Probable cause

This code indicates there was either no network error, or an unknown network error occurred.

Feature not implemented

| Item | Value |
|------------|--|
| Error code | 12 |
| Constant | NOT_IMPLEMENTED (Java) STREAM_ERROR_NOT_IMPLEMENTED (C/C++) ulStreamErrorNotImplemented (Visual Basic) |

Probable cause

An unimplemented internal feature was requested. Please contact technical support.

Invalid parameter '%1!s!'

| Item | Value |
|-------------|---|
| Error code | 1 |
| Constant | PARAMETER (Java) STREAM_ERROR_PARAMETER (C/C++) ulStreamErrorParameter (Visual Basic) |
| Parameter 1 | The invalid parameter value. |

Probable cause

Network parameters are of the form "name=value;[name2=value2[;...]]". This code indicates an invalid parameter value. Consult the documentation for the corresponding parameter name, and correct the parameter value.

Parameter value '%1!s!' is not a valid boolean value. The value must be 0 or 1

| Item | Value |
|-------------|---|
| Error code | 4 |
| Constant | PARAMETER_NOT_BOOLEAN (Java) STREAM_ERROR_PARAMETER_NOT_BOOLEAN (C/C++) ulStreamErrorParameterNotBoolean (Visual Basic) |
| Parameter 1 | The invalid parameter value. |

Probable cause

Network parameters are of the form "name=value;[name2=value2[;...]]". The parameter value is not a boolean value. Locate the offending the parameter specification and change the value of the parameter to either 0 (for off or false) or 1 (for on or true).

Parameter value '%1!s!' is not a valid hexadecimal value

| Item | Value |
|-------------|---|
| Error code | 5 |
| Constant | PARAMETER_NOT_HEX (Java) STREAM_ERROR_PARAMETER_NOT_HEX (C/C++) ulStreamErrorParameterNotHex (Visual Basic) |
| Parameter 1 | The invalid parameter value. |

Probable cause Network parameters are of the form "name=value;[name2=value2[;...]]". The parameter value is not a hexadecimal (base 16) value. Locate the offending the parameter specification and change the value of the parameter to a hexadecimal value.

Parameter value '%1!s!' is not an unsigned integer

| Item | Value |
|-------------|--|
| Error code | 2 |
| Constant | PARAMETER_NOT_UINT32 (Java) STREAM_ERROR_PARAMETER_NOT_UINT32 (C/C++) ulStreamErrorParameterNotUInt32 (Visual Basic) |
| Parameter 1 | The invalid parameter value. |

Probable cause Network parameters are of the form "name=value;[name2=value2[;...]]". The parameter value is not an unsigned integer. Locate the offending parameter specification and change the value of the parameter to an unsigned integer.

**Parameter value '%1!s!' is not an unsigned integer value or range.
A range has the form NNN-NNN**

| Item | Value |
|-------------|---|
| Error code | 3 |
| Constant | PARAMETER_NOT_UINT32_RANGE (Java) STREAM_ERROR_PARAMETER_NOT_UINT32_RANGE (C/C++) ulStreamErrorParameterNotUInt32Range (Visual Basic) |
| Parameter 1 | The invalid parameter value. |

Probable cause Network parameters are of the form "name=value;[name2=value2[;...]]". The parameter value is not an unsigned integer value or range. Locate the offending the parameter specification and change the value of the parameter to an unsigned integer or an unsigned range. An unsigned range has the form: NNN-NNN.

Unable to parse the parameter string '%1!s!'

| Item | Value |
|-------------|---|
| Error code | 7 |
| Constant | PARSE (Java) STREAM_ERROR_PARSE (C/C++) ulStreamErrorParse (Visual Basic) |
| Parameter 1 | The parameter string that could not be parsed. |

Probable cause

Network parameters are of the form "name=value;[name2=value2[;...]]". Optionally, the entire list of parameters may be enclosed in parentheses. The given string does not follow this convention. Inspect the string, fix any formatting problems, and retry the operation.

Unable to read %1!s! bytes

| Item | Value |
|-------------|--|
| Error code | 8 |
| Constant | READ (Java) STREAM_ERROR_READ (C/C++) ulStreamErrorRead (Visual Basic) |
| Parameter 1 | The number of bytes that could not be read. |

Probable cause

Unable to read the given number of bytes from the network layer. Note that reads may occur as part of any larger network operation. For example, some network layers have sub-layers that perform several reads and writes as part of a basic operation in the upper layer.

The cause of a read error is usually one of the following:

1) The network had a problem that caused the read to fail.

Reconnect and retry the operation.

2) The connection timed out.

Reconnect and retry the operation.

3) The other side of the connection cleanly terminated the connection.

Consult the client and/or server logs for errors that indicate why the connection has been dropped.

Consult the output-log errors and fix the cause, then retry the operation.

4) The process at the other side of the connection was aborted.

Consult the client and/or server output logs for errors that indicate why the process was aborted.

If the process was shut down by other than normal means, there may not be any errors in its output log.

Reconnect and retry the operation.

5) The system is low on resources, and cannot perform the read.

Free up system resources, reconnect and retry the operation. If subsequent retry attempts fail, consult your network administrator.

Unable to add a certificate to a certificate chain

| Item | Value |
|------------|--|
| Error code | 39 |
| Constant | SECURE_ADD_CERTIFICATE (Java) STREAM_ERROR_SECURE_ADD_CERTIFICATE (C/C++) ulStreamErrorSecureAddCertificate (Visual Basic) |

Probable cause The secure network layer was unable to add a certificate to a certificate chain. Free up system resources and retry the operation.

Unable to add a trusted certificate

| Item | Value |
|------------|---|
| Error code | 48 |
| Constant | SECURE_ADD_TRUSTED_CERTIFICATE (Java) STREAM_ERROR_SECURE_ADD_TRUSTED_CERTIFICATE (C/C++) ulStreamErrorSecureAddTrustedCertificate (Visual Basic) |

Probable cause The secure network layer was unable to add a trusted certificate to a certificate chain. The most likely cause is a shortage of system resources. Free up system resources and retry the operation.

Internal error 4028

| Item | Value |
|------------|--|
| Error code | 28 |
| Constant | SECURE_CERTIFICATE_CHAIN_FUNC (Java) STREAM_ERROR_SECURE_CERTIFICATE_CHAIN_FUNC (C/C++) ulStreamErrorSecureCertificateChainFunc (Visual Basic) |

Probable cause An internal error has occurred in the network layer. Please contact technical support.

Invalid certificate chain length (%1s!)

| Item | Value |
|-------------|--|
| Error code | 22 |
| Constant | SECURE_CERTIFICATE_CHAIN_LENGTH (Java) STREAM_ERROR_SECURE_CERTIFICATE_CHAIN_LENGTH (C/C++) ulStreamErrorSecureCertificateChainLength (Visual Basic) |
| Parameter 1 | The certificate chain length. |

Probable cause The certificate chain has the wrong length. This is an internal error that should never occur.

Internal error 4029

| Item | Value |
|------------|---|
| Error code | 29 |
| Constant | SECURE_CERTIFICATE_CHAIN_REF (Java) STREAM_ERROR_SECURE_CERTIFICATE_CHAIN_REF (C/C++) ulStreamErrorSecureCertificateChainRef (Visual Basic) |

Probable cause An internal error has occurred in the network layer. Please contact technical support.

Unrecognized common name '%1!s!'

| Item | Value |
|-------------|---|
| Error code | 52 |
| Constant | SECURE_CERTIFICATE_COMMON_NAME (Java) STREAM_ERROR_SECURE_CERTIFICATE_COMMON_NAME (C/C++) ulStreamErrorSecureCertificateCommonName (Visual Basic) |
| Parameter 1 | The common name. |

Probable cause

The given common name is not in the certificate chain. Check the following:

- 1) The common name was properly entered.
- 2) The correct certificate file was specified.
- 3) The common name is in the certificate chain. You can verify this with the readcert utility.

Unrecognized organization '%1!s!'

| Item | Value |
|-------------|--|
| Error code | 21 |
| Constant | SECURE_CERTIFICATE_COMPANY_NAME (Java) STREAM_ERROR_SECURE_CERTIFICATE_COMPANY_NAME (C/C++) ulStreamErrorSecureCertificateCompanyName (Visual Basic) |
| Parameter 1 | The organization name. |

Probable cause

The given organization name is not in the certificate chain. Check the following:

- 1) The organization name was properly entered.
- 2) The correct certificate file was specified.
- 3) The organization name is in the certificate chain. You can verify this with the readcert utility.

Unrecognized organization unit '%1!s!'

| Item | Value |
|-------------|--|
| Error code | 51 |
| Constant | SECURE_CERTIFICATE_COMPANY_UNIT (Java) STREAM_ERROR_SECURE_CERTIFICATE_COMPANY_UNIT (C/C++) ulStreamErrorSecureCertificateCompanyUnit (Visual Basic) |
| Parameter 1 | The organization unit name. |

Probable cause

The given organization unit is not in the certificate chain. Check the following:

- 1) The in company name was properly entered.
- 2) The correct certificate file was specified.
- 3) The company name is in the certificate chain. You can verify this with the readcert utility.

No trusted certificates found

| Item | Value |
|------------|--|
| Error code | 42 |
| Constant | SECURE_CERTIFICATE_COUNT (Java) STREAM_ERROR_SECURE_CERTIFICATE_COUNT (C/C++) ulStreamErrorSecureCertificateCount (Visual Basic) |

Probable cause

The given file does not contain a certificate. Check the following:

- 1) The certificate file name was properly specified.
- 2) The certificate file contains one or more certificates.
- 3) The certificate file contains the correct certificate(s).

A certificate has expired

| Item | Value |
|------------|--|
| Error code | 50 |
| Constant | SECURE_CERTIFICATE_EXPIRED (Java) STREAM_ERROR_SECURE_CERTIFICATE_EXPIRED (C/C++) ulStreamErrorSecureCertificateExpired (Visual Basic) |

Probable cause A certificate in the certificate chain has expired. Obtain a new certificate with a later expiry date and retry the operation.

Unable to fetch a certificate expiry date

| Item | Value |
|------------|---|
| Error code | 37 |
| Constant | SECURE_CERTIFICATE_EXPIRY_DATE (Java) STREAM_ERROR_SECURE_CERTIFICATE_EXPIRY_DATE (C/C++) ulStreamErrorSecureCertificateExpiryDate (Visual Basic) |

Probable cause A certificate's expiry date could not be read. Check the following:

- 1) The password was entered correctly.
- 2) The certificate file contains one or more certificates.
- 3) The certificate file contains the correct certificate(s).
- 4) The certificate file is undamaged.

Unable to open certificate file '%1!s!'

| Item | Value |
|-------------|---|
| Error code | 33 |
| Constant | SECURE_CERTIFICATE_FILE_NOT_FOUND (Java) STREAM_ERROR_SECURE_CERTIFICATE_FILE_NOT_FOUND (C/C++) ulStreamErrorSecureCertificateFileNotFound (Visual Basic) |
| Parameter 1 | The certificate file name. |

Probable cause The certificate file could not be opened. Check the following:

- 1) The certificate file name was properly specified.

- 2) The certificate file exists.
- 3) The certificate file contains one or more certificates.
- 4) The certificate file contains the correct certificate(s).
- 5) The program attempting to open the certificate file has sufficient privileges to read the file. This only applies to operating systems having user and/or file permissions.

Server certificate not trusted

| Item | Value |
|------------|---|
| Error code | 24 |
| Constant | SECURE_CERTIFICATE_NOT_TRUSTED (Java) STREAM_ERROR_SECURE_CERTIFICATE_NOT_TRUSTED (C/C++) ulStreamErrorSecureCertificateNotTrusted (Visual Basic) |

Probable cause

The server's certificate was not signed by a trusted authority. Check the following:

- 1) The certificate file name was properly specified.
- 2) The certificate file contains one or more certificates.
- 3) The certificate file contains the correct certificate(s).
- 4) The client's list of trusted root certificates includes the server's root certificate.

Certificate error (4023)

| Item | Value |
|------------|--|
| Error code | 23 |
| Constant | SECURE_CERTIFICATE_REF (Java) STREAM_ERROR_SECURE_CERTIFICATE_REF (C/C++) ulStreamErrorSecureCertificateRef (Visual Basic) |

Probable cause

This is an internal error in the secure network layer. This is an internal error that should never occur.

Invalid root certificate

| Item | Value |
|------------|---|
| Error code | 20 |
| Constant | SECURE_CERTIFICATE_ROOT (Java) STREAM_ERROR_SECURE_CERTIFICATE_ROOT (C/C++) ulStreamErrorSecureCertificateRoot (Visual Basic) |

Probable cause The root certificate in the chain is invalid. At the time of publication, this error was defined but not used.

Unable to allocate a certificate

| Item | Value |
|------------|---|
| Error code | 43 |
| Constant | SECURE_CREATE_CERTIFICATE (Java) STREAM_ERROR_SECURE_CREATE_CERTIFICATE (C/C++) ulStreamErrorSecureCreateCertificate (Visual Basic) |

Probable cause The secure network layer was unable to allocate storage for a certificate. Free up system resources and retry the operation.

Unable to create a private key object

| Item | Value |
|------------|--|
| Error code | 49 |
| Constant | SECURE_CREATE_PRIVATE_KEY_OBJECT (Java) STREAM_ERROR_SECURE_CREATE_PRIVATE_KEY_OBJECT (C/C++) ulStreamErrorSecureCreatePrivateKeyObject (Visual Basic) |

Probable cause The secure network layer was unable to create a private key object, prior to loading the private key. The most likely cause is a shortage of system resources. Free up system resources and retry the operation.

Unable to duplicate security context

| Item | Value |
|------------|--|
| Error code | 25 |
| Constant | SECURE_DUPLICATE_CONTEXT (Java) STREAM_ERROR_SECURE_DUPLICATE_CONTEXT (C/C++) ulStreamErrorSecureDuplicateContext (Visual Basic) |

Probable cause The secure network layer was unable to duplicate a security context.
Free up system resources and retry the operation.

Internal error 4030

| Item | Value |
|------------|---|
| Error code | 30 |
| Constant | SECURE_ENABLE_NON_BLOCKING (Java) STREAM_ERROR_SECURE_ENABLE_NON_BLOCKING (C/C++) ulStreamErrorSecureEnableNonBlocking (Visual Basic) |

Probable cause An internal error has occurred in the network layer. Please contact technical support.

Unable to copy a certificate

| Item | Value |
|------------|---|
| Error code | 38 |
| Constant | SECURE_EXPORT_CERTIFICATE (Java) STREAM_ERROR_SECURE_EXPORT_CERTIFICATE (C/C++) ulStreamErrorSecureExportCertificate (Visual Basic) |

Probable cause The secure network layer was unable to copy a certificate. Free up system resources and retry the operation.

Handshake error

| Item | Value |
|------------|---|
| Error code | 53 |
| Constant | SECURE_HANDSHAKE (Java) STREAM_ERROR_SECURE_HANDSHAKE (C/C++) ulStreamErrorSecureHandshake (Visual Basic) |

Probable cause

The secure handshake failed. Check the following:

- 1) On the client, the correct host machine and port number were specified.
- 2) On the server, the correct port number was specified.
- 3) The correct certificate file was specified, both on the client and on the server.

Unable to import a certificate

| Item | Value |
|------------|---|
| Error code | 44 |
| Constant | SECURE_IMPORT_CERTIFICATE (Java) STREAM_ERROR_SECURE_IMPORT_CERTIFICATE (C/C++) ulStreamErrorSecureImportCertificate (Visual Basic) |

Probable cause

The secure network layer was unable to import a certificate. Check the following:

- 1) The certificate file name was properly specified.
- 2) The certificate file exists.
- 3) The certificate file contains one or more certificates.
- 4) The certificate file contains the correct certificate(s).

Unable to read certificates

| Item | Value |
|------------|---|
| Error code | 34 |
| Constant | SECURE_READ_CERTIFICATE (Java) STREAM_ERROR_SECURE_READ_CERTIFICATE (C/C++) ulStreamErrorSecureReadCertificate (Visual Basic) |

Probable cause

The certificate file could not be read. Check the following:

- 1) The password was entered correctly.
- 2) The certificate file contains one or more certificates.
- 3) The certificate file contains the correct certificate(s).
- 4) The certificate file is undamaged.

Unable to read the private key

| Item | Value |
|------------|--|
| Error code | 35 |
| Constant | SECURE_READ_PRIVATE_KEY (Java) STREAM_ERROR_SECURE_READ_PRIVATE_KEY (C/C++) ulStreamErrorSecureReadPrivateKey (Visual Basic) |

Probable cause

The private key could not be read from the certificate file. Check the following:

- 1) The password was entered correctly.
- 2) The certificate file contains one or more certificates.
- 3) The certificate file contains the correct certificate(s).
- 4) The certificate file is undamaged.

Internal error 4032

| Item | Value |
|------------|--|
| Error code | 32 |
| Constant | SECURE_SET_CHAIN_NUMBER (Java) STREAM_ERROR_SECURE_SET_CHAIN_NUMBER (C/C++) ulStreamErrorSecureSetChainNumber (Visual Basic) |

Probable cause An internal error has occurred in the network layer. Please contact technical support.

Internal error 4031

| Item | Value |
|------------|---|
| Error code | 31 |
| Constant | SECURE_SET_CIPHER_SUITES (Java) STREAM_ERROR_SECURE_SET_CIPHER_SUITES (C/C++) ulStreamErrorSecureSetCipherSuites (Visual Basic) |

Probable cause An internal error has occurred in the network layer. Please contact technical support.

Unable to attach the network layer to the security layer

| Item | Value |
|------------|---|
| Error code | 26 |
| Constant | SECURE_SET_IO (Java) STREAM_ERROR_SECURE_SET_IO (C/C++) ulStreamErrorSecureSetIo (Visual Basic) |

Probable cause The secure network layer was unable to attach to the network layer. Free up system resources and retry the operation.

Internal error 4027

| Item | Value |
|------------|--|
| Error code | 27 |
| Constant | SECURE_SET_IO_SEMANTICS (Java) STREAM_ERROR_SECURE_SET_IO_SEMANTICS (C/C++) ulStreamErrorSecureSetIoSemantics (Visual Basic) |

Probable cause An internal error has occurred in the network layer. Please contact technical support.

Unable to set the private key

| Item | Value |
|------------|---|
| Error code | 36 |
| Constant | SECURE_SET_PRIVATE_KEY (Java) STREAM_ERROR_SECURE_SET_PRIVATE_KEY (C/C++) ulStreamErrorSecureSetPrivateKey (Visual Basic) |

Probable cause The private key could not be used. Check the following:

- 1) The password was entered correctly.
- 2) The certificate file contains one or more certificates.
- 3) The certificate file contains the correct certificate(s).
- 4) The certificate file is undamaged.

Unable to set the protocol side (%1!s!)

| Item | Value |
|-------------|---|
| Error code | 47 |
| Constant | SECURE_SET_PROTOCOL_SIDE (Java) STREAM_ERROR_SECURE_SET_PROTOCOL_SIDE (C/C++) ulStreamErrorSecureSetProtocolSide (Visual Basic) |
| Parameter 1 | The server side being set. The value is 1 for server side, and 2 for client side. |

Probable cause The secure network layer was unable to establish the given protocol side. This is an internal error that should never occur.

Internal initialization error 4046

| Item | Value |
|------------|---|
| Error code | 46 |
| Constant | SECURE_SET_RANDOM_FUNC (Java) STREAM_ERROR_SECURE_SET_RANDOM_FUNC (C/C++) ulStreamErrorSecureSetRandomFunc (Visual Basic) |

Probable cause An internal error has occurred in the network layer. Please contact technical support.

Internal initialization error 4045

| Item | Value |
|------------|--|
| Error code | 45 |
| Constant | SECURE_SET_RANDOM_REF (Java) STREAM_ERROR_SECURE_SET_RANDOM_REF (C/C++) ulStreamErrorSecureSetRandomRef (Visual Basic) |

Probable cause An internal error has occurred in the network layer. Please contact technical support.

Internal initialization error 4055

| Item | Value |
|------------|---|
| Error code | 55 |
| Constant | SECURE_SET_READ_FUNC (Java) STREAM_ERROR_SECURE_SET_READ_FUNC (C/C++) ulStreamErrorSecureSetReadFunc (Visual Basic) |

Probable cause This initialization error is most likely due to a lack of system resources. Free up system resources and retry the operation.

Internal initialization error 4056

| Item | Value |
|------------|--|
| Error code | 56 |
| Constant | SECURE_SET_WRITE_FUNC (Java) STREAM_ERROR_SECURE_SET_WRITE_FUNC (C/C++) ulStreamErrorSecureSetWriteFunc (Visual Basic) |

Probable cause This initialization error is most likely due to a lack of system resources. Free up system resources and retry the operation.

Unable to find the trusted certificate file '%1!s!'

| Item | Value |
|-------------|--|
| Error code | 40 |
| Constant | SECURE_TRUSTED_CERTIFICATE_FILE_NOT_FOUND (Java) STREAM_ERROR_SECURE_TRUSTED_CERTIFICATE_FILE_NOT_FOUND (C/C++) ulStreamErrorSecureTrustedCertificateFileNotFound (Visual Basic) |
| Parameter 1 | The trusted certificate file name. |

Probable cause The certificate file could not be found. Check the following:

- 1) The certificate file name was properly specified.
- 2) The certificate file exists.
- 3) The certificate file contains one or more certificates.
- 4) The certificate file contains the correct certificate(s).
- 5) The program attempting to open the certificate file has sufficient privileges to see the file. This only applies to operating systems having user and/or file permissions.

Error reading from the trusted certificate file '%1!s!'

| Item | Value |
|-------------|--|
| Error code | 41 |
| Constant | SECURE_TRUSTED_CERTIFICATE_READ (Java) STREAM_ERROR_SECURE_TRUSTED_CERTIFICATE_READ (C/C++) ulStreamErrorSecureTrustedCertificateRead (Visual Basic) |
| Parameter 1 | The trusted certificate file name. |

Probable cause

The secure network layer was unable to read the trusted certificate file.
Check the following:

- 1) The certificate file name was properly specified.
- 2) The certificate file exists.
- 3) The certificate file contains one or more certificates.
- 4) The certificate file contains the correct certificate(s).
- 5) The program attempting to open the certificate file has sufficient privileges to see the file. This only applies to operating systems having user and/or file permissions.

Unable to seed the random number generator

| Item | Value |
|------------|--|
| Error code | 16 |
| Constant | SEED_RANDOM (Java) STREAM_ERROR_SEED_RANDOM (C/C++) ulStreamErrorSeedRandom (Visual Basic) |

Probable cause

The secure network layer could not seed its random number generator. Free up system resources, reconnect and retry the operation.

An error occurred during shutdown

| Item | Value |
|------------|--|
| Error code | 18 |
| Constant | SHUTTING_DOWN (Java) STREAM_ERROR_SHUTTING_DOWN (C/C++) ulStreamErrorShuttingDown (Visual Basic) |

Probable cause The MobiLink synchronization server encountered an error in the network layer during shutdown. It is possible that some network operations pending at the time of shutdown were affected.

Unable to bind a socket to port %1s!

| Item | Value |
|-------------|--|
| Error code | 62 |
| Constant | SOCKET_BIND (Java) STREAM_ERROR_SOCKET_BIND (C/C++) ulStreamErrorSocketBind (Visual Basic) |
| Parameter 1 | The port number. |

Probable cause The network layer was unable to bind a socket to the given port. Check the following.

- 1) (Server only) Verify that the port isn't already in use. If the port is in use, either shut down the application listening on that port, or specify a different port.
- 2) (Server only) Verify that there are no firewall restrictions on the use of the port.
- 3) (Client only) If the client_port option was used, verify that the given port isn't already in use. If only one client port was specified, consider using a range (eg. NNN-NNN). If a range was specified, consider making it a wider range, or a different range.
- 4) (Client only) If the client_port option was used, verify that there are no firewall restrictions on the use of the port.

Unable to cleanup the socket layer

| Item | Value |
|------------|---|
| Error code | 63 |
| Constant | SOCKET_CLEANUP (Java) STREAM_ERROR_SOCKET_CLEANUP (C/C++) ulStreamErrorSocketCleanup (Visual Basic) |

Probable cause The network layer was unable to clean up the socket layer. This error should only occur after all connections are finished, so no current connections should be affected.

Unable to close a socket

| Item | Value |
|------------|---|
| Error code | 64 |
| Constant | SOCKET_CLOSE (Java) STREAM_ERROR_SOCKET_CLOSE (C/C++) ulStreamErrorSocketClose (Visual Basic) |

Probable cause The network layer was unable to close a socket. The network session may or may not have terminated prematurely, due to pending writes that were not flushed. Check the following:

- 1) See if the other side of the network connection had any errors.
- 2) The other side of the connection is running normally.
- 3) The machine is still connected to the network, and the network is responsive.

Unable to connect a socket

| Item | Value |
|------------|---|
| Error code | 65 |
| Constant | SOCKET_CONNECT (Java) STREAM_ERROR_SOCKET_CONNECT (C/C++) ulStreamErrorSocketConnect (Visual Basic) |

Probable cause The network layer was unable to connect a socket. Check the following:

- 1) The machine is connected to the network.
- 2) The socket layer is properly initialized.
- 3) The correct host machine and port were specified.
- 4) The host server is running normally and listening on the correct port.
- 5) The host machine is listening for the proper socket type (TCP/IP vs. UDP).
- 6) If the `client_port` option was used, verify that there are no firewall restrictions on the use of the port.
- 7) If the device has a limit on the number of open sockets, verify that the limit has not been reached.
- 8) There are enough system resources available. Free up system resources if they are running low.

Unable to create a TCP/IP socket

| Item | Value |
|------------|---|
| Error code | 60 |
| Constant | SOCKET_CREATE_TCPIP (Java) STREAM_ERROR_SOCKET_CREATE_TCPIP (C/C++) ulStreamErrorSocketCreateTcpip (Visual Basic) |

Probable cause

The network layer was unable to create a TCP/IP socket. Check the following:

- 1) The machine is connected to the network.
- 2) The socket layer is properly initialized.
- 5) If the device has a limit on the number of open sockets, verify that the limit has not been reached.
- 6) There are enough system resources available. Free up system resources if they are running low.

Unable to create a UDP socket

| Item | Value |
|------------|---|
| Error code | 61 |
| Constant | SOCKET_CREATE_UDP (Java) STREAM_ERROR_SOCKET_CREATE_UDP (C/C++) ulStreamErrorSocketCreateUdp (Visual Basic) |

Probable cause

The network layer was unable to create a UDP socket. Check the following:

- 1) The machine is connected to the network.
- 2) The socket layer is properly initialized.
- 3) If the client_port option was used, verify that the given port isn't already in use. If only one client port was specified, consider using a range (eg. NNN-NNN). If a range was specified, consider making it a wider range, or a different range.
- 4) If the client_port option was used, verify that there are no firewall restrictions on the use of the port.
- 5) If the device has a limit on the number of open sockets, verify that the limit has not been reached.
- 6) There are enough system resources available. Free up system resources if they are running low.

Unable to get host by address

| Item | Value |
|------------|---|
| Error code | 58 |
| Constant | SOCKET_GET_HOST_BY_ADDR (Java) STREAM_ERROR_SOCKET_GET_HOST_BY_ADDR (C/C++) ulStreamErrorSocketGetHostByAddr (Visual Basic) |

Probable cause

The network layer was unable to get the name of a host using its IP address. At the time of publication, this error was defined but not used.

Unable to get a socket's local name

| Item | Value |
|------------|---|
| Error code | 66 |
| Constant | SOCKET_GET_NAME (Java) STREAM_ERROR_SOCKET_GET_NAME (C/C++) ulStreamErrorSocketGetName (Visual Basic) |

Probable cause

The network layer was unable to determine a socket's local name. In a TCP/IP connection, each end of the connection has a socket exclusively attached to a port. A socket's local name includes this port number, which is assigned by the network at connection time. Check the following:

- 1) The machine is still connected to the network, and the network is responsive.
- 2) The other side of the connection is running normally.
- 3) There are enough system resources available. Free up system resources if they are running low.

Unable to get socket option number %1!s!

| Item | Value |
|-------------|---|
| Error code | 67 |
| Constant | SOCKET_GET_OPTION (Java) STREAM_ERROR_SOCKET_GET_OPTION (C/C++) ulStreamErrorSocketGetOption (Visual Basic) |
| Parameter 1 | The socket option being retrieved. |

Probable cause

The network layer was unable to get a socket option. This error may be the first indication that a connection has been lost. Check the following:

- 1) The machine is still connected to the network, and the network is responsive.
- 2) The other side of the connection is running normally.
- 3) There are enough system resources available. Free up system resources if they are running low.

The host name '%1!s!' could not be found

| Item | Value |
|-------------|--|
| Error code | 57 |
| Constant | SOCKET_HOST_NAME_NOT_FOUND (Java) STREAM_ERROR_SOCKET_HOST_NAME_NOT_FOUND (C/C++) ulStreamErrorSocketHostNameNotFound (Visual Basic) |
| Parameter 1 | The name of the host. |

Probable cause

The given host name could not be found. Check the following:

- 1) The host name was correctly specified.
- 2) The host is accessible. Many systems include a "ping" utility that can be used to verify access to a named host.
- 3) The Domain Name Server (DNS), or its equivalent, is available. If the DNS is not available, try specifying the host's IP number (eg. NNN.NNN.NNN.NNN) instead of the host name.
- 4) The HOSTS file contains an entry that maps the host name to an IP number.

Unable to listen on a socket. The backlog is %1!s!

| Item | Value |
|-------------|--|
| Error code | 69 |
| Constant | SOCKET_LISTEN (Java) STREAM_ERROR_SOCKET_LISTEN (C/C++) ulStreamErrorSocketListen (Visual Basic) |
| Parameter 1 | The requested listener backlog. |

Probable cause

The server is unable to listen on a socket. The backlog refers to the maximum number of queued connection requests that may be pending at any given time. Check the following:

- 1) The machine is still connected to the network, and the network is responsive.
- 2) There are no firewall or other restrictions preventing a socket listener from running on the current machine.
- 3) The backlog setting is within the limit, if any, on the machine.

- 4) There are enough system resources available. Free up system resources if they are running low.

Unable to determine localhost

| Item | Value |
|------------|---|
| Error code | 59 |
| Constant | SOCKET_LOCALHOST_NAME_NOT_FOUND (Java) STREAM_ERROR_SOCKET_LOCALHOST_NAME_NOT_FOUND (C/C++) ulStreamErrorSocketLocalhostNameNotFound (Visual Basic) |

Probable cause

The network layer was unable to determine the IP address of "localhost". Check the following:

- 1) The Domain Name Server (DNS), or its equivalent, is available. If the DNS is not available, try explicitly specifying the localhost IP number (usually 127.0.0.1) instead.
- 2) The HOSTS file contains an entry that maps the "localhost" name to an IP number.
- 3) There are enough system resources available. Free up system resources if they are running low.

Invalid port number %1!s!. The value must be between zero and 65535

| Item | Value |
|-------------|--|
| Error code | 73 |
| Constant | SOCKET_PORT_OUT_OF_RANGE (Java) STREAM_ERROR_SOCKET_PORT_OUT_OF_RANGE (C/C++) ulStreamErrorSocketPortOutOfRange (Visual Basic) |
| Parameter 1 | The port number. |

Probable cause

An invalid port number was specified. The port number must be an integer between zero and 65535.

Unable to select a socket status

| Item | Value |
|------------|--|
| Error code | 71 |
| Constant | SOCKET_SELECT (Java) STREAM_ERROR_SOCKET_SELECT (C/C++) ulStreamErrorSocketSelect (Visual Basic) |

- Probable cause**
- The network layer encountered an error attempting to wait for a socket to be ready for reading or writing. Check the following:
- 1) The machine is connected to the network, and the network is responsive.

2) The other side of the connection is running normally.

3) There are enough system resources available. Free up system resources if they are running low.

Unable to set socket option number %1!s!

| Item | Value |
|-------------|---|
| Error code | 68 |
| Constant | SOCKET_SET_OPTION (Java) STREAM_ERROR_SOCKET_SET_OPTION (C/C++) ulStreamErrorSocketSetOption (Visual Basic) |
| Parameter 1 | The socket option being set. |

- Probable cause**
- The network layer was unable to set a socket option. This error may be the first indication that a connection has been lost. Check the following:
- 1) The machine is still connected to the network, and the network is responsive.

2) The other side of the connection is running normally.

3) There are enough system resources available. Free up system resources if they are running low.

Unable to shutdown a socket

| Item | Value |
|------------|--|
| Error code | 70 |
| Constant | SOCKET_SHUTDOWN (Java) STREAM_ERROR_SOCKET_SHUTDOWN (C/C++) ulStreamErrorSocketShutdown (Visual Basic) |

Probable cause

The network layer was unable to shut down a socket. Check the following:

- 1) The machine is connected to the network, and the network is responsive.
- 2) The other side of the connection is running normally.
- 3) There are enough system resources available. Free up system resources if they are running low.

Unable to initialize the sockets layer

| Item | Value |
|------------|---|
| Error code | 72 |
| Constant | SOCKET_STARTUP (Java) STREAM_ERROR_SOCKET_STARTUP (C/C++) ulStreamErrorSocketStartup (Visual Basic) |

Probable cause

The network layer was unable to initialize the socket layer. Check the following:

- 1) The sockets layer is properly installed. The correct network interface library must be present and accessible.
- 2) The machine is connected to the network, and the network is responsive.
- 3) There are enough system resources available. Free up system resources if they are running low.

The operation would cause blocking

| Item | Value |
|------------|--|
| Error code | 13 |
| Constant | WOULD_BLOCK (Java) STREAM_ERROR_WOULD_BLOCK (C/C++) ulStreamErrorWouldBlock (Visual Basic) |

Probable cause A requested operation would block where blocking is undesirable or unexpected.

Unable to write %1!s! bytes

| Item | Value |
|-------------|---|
| Error code | 9 |
| Constant | WRITE (Java) STREAM_ERROR_WRITE (C/C++) ulStreamErrorWrite (Visual Basic) |
| Parameter 1 | The number of bytes that could not be written. |

Probable cause Unable to write the given number of bytes to the network layer. Note that writes may occur as part of any larger network operation. For example, some network layers have sub-layers that perform several reads and writes as part of a basic operation in the upper layer.

The cause of a write error is usually one of the following:

1) The network had a problem that caused the write to fail.

Reconnect and retry the operation.

2) The connection timed out.

Reconnect and retry the operation.

3) The other side of the connection cleanly terminated the connection.

Consult the client and/or server logs for errors that indicate why the connection has been dropped.

Consult the output-log errors and fix the cause, then retry the operation.

4) The process at the other side of the connection was aborted.

Consult the client and/or server output logs for errors that indicate why the process was aborted.

If the process was shut down by other than normal means, there may not be any errors in its output log.

Reconnect and retry the operation.

5) The system is low on resources, and cannot perform the write.

Free up system resources, reconnect and retry the operation. If subsequent retry attempts fail, consult your network administrator.

MobiLink synchronization server Warning Messages

About this chapter This chapter lists MobiLink synchronization server warnings, as well as their probable causes.

 The warning messages are written to the MobiLink synchronization server message log.

Contents

| Topic | Page |
|--|------|
| MobiLink synchronization server warning messages sorted by code | 684 |
| MobiLink synchronization server warning messages sorted by message | 688 |
| MobiLink synchronization server warning descriptions | 692 |

MobiLink synchronization server warning messages sorted by code

| Warning code | Level | Warning message |
|---------------------|--------------|--|
| 10001 | 1 | "Maximum number of database connections set to %1!lu! (must be at least the number of worker threads plus one)" on page 696 |
| 10002 | 1 | "If needed, ODBC cursors will be used, via the Microsoft ODBC Cursor Library, to simulate SQLSETPOS for inserts, updates, and deletes" on page 694 |
| 10003 | 1 | "ODBC Isolation level (%1!s!) is not supported" on page 697 |
| 10004 | 1 | "ODBC function %1!s! is not supported by the driver" on page 697 |
| 10005 | 1 | "ODBC statement option %1!s! has changed from %2!s! (%3!lu!) to %4!s! (%5!lu!)" on page 698 |
| 10006 | 1 | "ODBC statement option %1!s! has changed from %2!lu! to %3!lu!" on page 697 |
| 10007 | 2 | "Retrying the begin_connection transaction after deadlock in the consolidated database" on page 698 |
| 10008 | 2 | "Retry on deadlock set to FALSE. MobiLink synchronization server is using an internal work-around which requires this setting" on page 698 |
| 10009 | 2 | "MobiLink table '%1!s!'" is damaged" on page 696 |
| 10010 | 2 | "No error-handling script is defined. The default action code (%1!ld!) will decide error behavior" on page 697 |
| 10011 | 2 | "Unrecognized value (%1!ld!) in ml_user.commit_state. The state information for this user is probably corrupted" on page 705 |
| 10012 | 2 | "The consolidated and remote databases disagree on when the last synchronization took place. The remote is being asked to send a new upload that starts at the last known synchronization point" on page 701 |
| 10013 | 4 | "Expecting %1!ld! parameter(s) in cursor, but found %2!ld!" on page 694 |

| Warning code | Level | Warning message |
|--------------|-------|---|
| 10014 | 4 | "Expecting at most %1!d! parameter(s) in cursor, but found %2!d!" on page 694 |
| 10015 | 3 | "Table '%1!s!' has at least one timestamp column. Due to a timestamp precision mismatch, uploaded timestamps can lose precision, defeating download filtering" on page 700 |
| 10016 | 3 | "Table '%1!s!' has at least one timestamp column. Due to a timestamp precision mismatch, downloaded timestamps can lose precision, resulting in inconsistent data" on page 700 |
| 10017 | 3 | "The consolidated and remote databases have different timestamp precisions. Consolidated database timestamps are precise to %1!d! digit(s) in the fractional second while the remote database timestamps are precise to %2!d! digit(s)" on page 701 |
| 10018 | 3 | "You may resolve the timestamp precision mismatch by setting the DEFAULT_TIMESTAMP_INCREMENT option on the remote database to %1!d! and TRUNCATE_TIMESTAMP_VALUES to 'On'" on page 705 |
| 10019 | 3 | "The remote database is not capable of matching the timestamp precision of the consolidated database. Your application, schema, and scripts must contain logic that copes with the precision mismatch" on page 702 |
| 10020 | 3 | "The timestamp precision mismatch may affect upload conflict detection. You can use the -zp option to cause MobiLink synchronization server to use the lowest timestamp precision for conflict detection purposes" on page 702 |
| 10021 | 3 | "The remote and consolidated databases have different timestamp precisions, and a timestamp value with a precision higher than the lower-precision side was used for conflict detection purposes. Consider using the -zp option" on page 701 |
| 10022 | 3 | "Publication '%1!s!' is not referenced by any table" on page 698 |

| Warning code | Level | Warning message |
|--------------|-------|--|
| 10023 | 3 | "The upload will be rolled back and the synchronization aborted. The next time this remote synchronizes, it will ask what happened to the previous upload" on page 703 |
| 10024 | 3 | "Table '%1s!' has no entry in the %2s! table" on page 700 |
| 10025 | 5 | "Invalid character data encountered in upload -- substituting '?'" on page 695 |
| 10026 | 5 | "Invalid character data encountered in upload -- using NULL" on page 695 |
| 10027 | 5 | "Invalid character data encountered in upload -- using empty string" on page 695 |
| 10028 | 5 | "Multi-byte characters truncated on upload" on page 696 |
| 10029 | 5 | "Unable to convert character data for download -- substituting '?'" on page 703 |
| 10030 | 5 | "Unable to convert character data for download -- using NULL" on page 703 |
| 10031 | 5 | "Unable to convert character data for download -- using empty string" on page 703 |
| 10032 | 2 | "Unable to open the file to store the client synchronization logs. The filename is '%1s!'" on page 704 |
| 10033 | 2 | "An error occurred reading the remote client's synchronization log" on page 693 |
| 10034 | 2 | "Unable to write to the local file that contains remote synchronization logs" on page 704 |
| 10035 | 2 | "The remote client's synchronization log ended prematurely, and was probably truncated" on page 701 |
| 10036 | 2 | "Client synchronization logs will be shown in the MobiLink synchronization server output file or the console" on page 693 |
| 10037 | 5 | "Ignoring updated row (new values)" on page 694 |
| 10038 | 5 | "Ignoring updated row (old values)" on page 695 |
| 10039 | 2 | "Error detected while using multi-row cursor -- retrying with single row cursor" on page 693 |

| Warning code | Level | Warning message |
|--------------|-------|---|
| 10040 | 2 | "%1!lu! row(s) were ignored in updating table %2!s!" on page 692 |
| 10041 | 2 | "The upload will be committed and the synchronization aborted. The next time this remote synchronizes, it will ask what happened to the previous upload" on page 702 |
| 10042 | 1 | "NT Performance Monitor data area failed to initialize" on page 696 |
| 10043 | 4 | "Unable to determine current timestamp" on page 704 |
| 10044 | 5 | "A row in table '%1!s!' could not be updated because it no longer exists in the consolidated database" on page 692 |
| 10045 | 2 | "Retrying the upload after deadlock in the consolidated database" on page 699 |
| 10046 | 2 | "Retrying the upload. Working around a known ODBC driver problem" on page 700 |
| 10047 | 4 | "Cannot directly determine the name of the table referenced by the cursor. The table name is required for inserts, updates, and deletes when using the Microsoft ODBC Cursor Library" on page 693 |
| 10048 | 2 | "Retrying the begin_synchronization transaction after deadlock in the consolidated database" on page 699 |
| 10049 | 2 | "Retrying the end_synchronization transaction after deadlock in the consolidated database" on page 699 |
| 10050 | 4 | "%1!s!" on page 692 |
| 10051 | 1 | "Unrecognized ODBC driver '%1!s!'. The functionality and quality of ODBC drivers varies greatly. This driver may lack functionality required for successful synchronizations. Use at your own risk" on page 704 |

MobiLink synchronization server warning messages sorted by message

| Warning code | Level | Warning message |
|---------------------|--------------|---|
| 10040 | 2 | "%1!lu! row(s) were ignored in updating table %2!s!" on page 692 |
| 10050 | 4 | "%1!s!" on page 692 |
| 10044 | 5 | "A row in table '%1!s!' could not be updated because it no longer exists in the consolidated database" on page 692 |
| 10033 | 2 | "An error occurred reading the remote client's synchronization log" on page 693 |
| 10047 | 4 | "Cannot directly determine the name of the table referenced by the cursor. The table name is required for inserts, updates, and deletes when using the Microsoft ODBC Cursor Library" on page 693 |
| 10036 | 2 | "Client synchronization logs will be shown in the MobiLink synchronization server output file or the console" on page 693 |
| 10039 | 2 | "Error detected while using multi-row cursor -- retrying with single row cursor" on page 693 |
| 10013 | 4 | "Expecting %1!ld! parameter(s) in cursor, but found %2!ld!" on page 694 |
| 10014 | 4 | "Expecting at most %1!ld! parameter(s) in cursor, but found %2!ld!" on page 694 |
| 10002 | 1 | "If needed, ODBC cursors will be used, via the Microsoft ODBC Cursor Library, to simulate SQLSETPOS for inserts, updates, and deletes" on page 694 |
| 10037 | 5 | "Ignoring updated row (new values)" on page 694 |
| 10038 | 5 | "Ignoring updated row (old values)" on page 695 |
| 10025 | 5 | "Invalid character data encountered in upload -- substituting '?'" on page 695 |
| 10026 | 5 | "Invalid character data encountered in upload -- using NULL" on page 695 |
| 10027 | 5 | "Invalid character data encountered in upload -- using empty string" on page 695 |

| Warning code | Level | Warning message |
|--------------|-------|--|
| 10001 | 1 | "Maximum number of database connections set to %1!lu! (must be at least the number of worker threads plus one)" on page 696 |
| 10009 | 2 | "MobiLink table '%1!s!' is damaged" on page 696 |
| 10028 | 5 | "Multi-byte characters truncated on upload" on page 696 |
| 10042 | 1 | "NT Performance Monitor data area failed to initialize" on page 696 |
| 10010 | 2 | "No error-handling script is defined. The default action code (%1!ld!) will decide error behavior" on page 697 |
| 10003 | 1 | "ODBC Isolation level (%1!s!) is not supported" on page 697 |
| 10004 | 1 | "ODBC function %1!s! is not supported by the driver" on page 697 |
| 10006 | 1 | "ODBC statement option %1!s! has changed from %2!lu! to %3!lu!" on page 697 |
| 10005 | 1 | "ODBC statement option %1!s! has changed from %2!s! (%3!lu!) to %4!s! (%5!lu!)" on page 698 |
| 10022 | 3 | "Publication '%1!s!' is not referenced by any table" on page 698 |
| 10008 | 2 | "Retry on deadlock set to FALSE. MobiLink synchronization server is using an internal work-around which requires this setting" on page 698 |
| 10007 | 2 | "Retrying the begin_connection transaction after deadlock in the consolidated database" on page 698 |
| 10048 | 2 | "Retrying the begin_synchronization transaction after deadlock in the consolidated database" on page 699 |
| 10049 | 2 | "Retrying the end_synchronization transaction after deadlock in the consolidated database" on page 699 |
| 10045 | 2 | "Retrying the upload after deadlock in the consolidated database" on page 699 |
| 10046 | 2 | "Retrying the upload. Working around a known ODBC driver problem" on page 700 |
| 10016 | 3 | "Table '%1!s!' has at least one timestamp column. Due to a timestamp precision mismatch, downloaded timestamps can lose precision, resulting in inconsistent data" on page 700 |

| Warning code | Level | Warning message |
|--------------|-------|---|
| 10015 | 3 | "Table '%1s!'" has at least one timestamp column. Due to a timestamp precision mismatch, uploaded timestamps can lose precision, defeating download filtering" on page 700 |
| 10024 | 3 | "Table '%1s!'" has no entry in the '%2s!' table" on page 700 |
| 10012 | 2 | "The consolidated and remote databases disagree on when the last synchronization took place. The remote is being asked to send a new upload that starts at the last known synchronization point" on page 701 |
| 10017 | 3 | "The consolidated and remote databases have different timestamp precisions. Consolidated database timestamps are precise to %1!d! digit(s) in the fractional second while the remote database timestamps are precise to %2!d! digit(s)" on page 701 |
| 10021 | 3 | "The remote and consolidated databases have different timestamp precisions, and a timestamp value with a precision higher than the lower-precision side was used for conflict detection purposes. Consider using the -zp option" on page 701 |
| 10035 | 2 | "The remote client's synchronization log ended prematurely, and was probably truncated" on page 701 |
| 10019 | 3 | "The remote database is not capable of matching the timestamp precision of the consolidated database. Your application, schema, and scripts must contain logic that copes with the precision mismatch" on page 702 |
| 10020 | 3 | "The timestamp precision mismatch may affect upload conflict detection. You can use the -zp option to cause MobiLink synchronization server to use the lowest timestamp precision for conflict detection purposes" on page 702 |
| 10041 | 2 | "The upload will be committed and the synchronization aborted. The next time this remote synchronizes, it will ask what happened to the previous upload" on page 702 |

| Warning code | Level | Warning message |
|--------------|-------|--|
| 10023 | 3 | "The upload will be rolled back and the synchronization aborted. The next time this remote synchronizes, it will ask what happened to the previous upload" on page 703 |
| 10029 | 5 | "Unable to convert character data for download -- substituting '?'" on page 703 |
| 10030 | 5 | "Unable to convert character data for download -- using NULL" on page 703 |
| 10031 | 5 | "Unable to convert character data for download -- using empty string" on page 703 |
| 10043 | 4 | "Unable to determine current timestamp" on page 704 |
| 10032 | 2 | "Unable to open the file to store the client synchronization logs. The filename is '%!s!'" on page 704 |
| 10034 | 2 | "Unable to write to the local file that contains remote synchronization logs" on page 704 |
| 10051 | 1 | "Unrecognized ODBC driver '%!s!'. The functionality and quality of ODBC drivers varies greatly. This driver may lack functionality required for successful synchronizations. Use at your own risk" on page 704 |
| 10011 | 2 | "Unrecognized value (%!ld!) in ml_user.commit_state. The state information for this user is probably corrupted" on page 705 |
| 10018 | 3 | "You may resolve the timestamp precision mismatch by setting the DEFAULT_TIMESTAMP_INCREMENT option on the remote database to '%!d!' and TRUNCATE_TIMESTAMP_VALUES to 'On'" on page 705 |

MobiLink synchronization server warning descriptions

This section provides a full listing of warning messages and descriptions.

Warnings with an ODBC state marked "handled by ODBC driver" are not returned to ODBC applications, as the ODBC driver carries out the required actions.

%1!lu! row(s) were ignored in updating table %2!s!

| Item | Value |
|--------------|-------|
| Warning code | 10040 |
| Level | 2 |

%1!s!

| Item | Value |
|--------------|----------------------------|
| Warning code | 10050 |
| Level | 4 |
| Parameter 1 | A message from ODBC driver |

Probable cause Calling ODBC functions succeeded, but the ODBC driver could show a warning message.

A row in table '%1!s!' could not be updated because it no longer exists in the consolidated database

| Item | Value |
|--------------|------------|
| Warning code | 10044 |
| Level | 5 |
| Parameter 1 | Table name |

Probable cause

An update statement failed because the table in the consolidated database doesn't have the row.

An error occurred reading the remote client's synchronization log

| Item | Value |
|--------------|-------|
| Warning code | 10033 |
| Level | 2 |

Cannot directly determine the name of the table referenced by the cursor. The table name is required for inserts, updates, and deletes when using the Microsoft ODBC Cursor Library

| Item | Value |
|--------------|-------|
| Warning code | 10047 |
| Level | 4 |

Client synchronization logs will be shown in the MobiLink synchronization server output file or the console

| Item | Value |
|--------------|-------|
| Warning code | 10036 |
| Level | 2 |

Error detected while using multi-row cursor -- retrying with single row cursor

| Item | Value |
|--------------|-------|
| Warning code | 10039 |
| Level | 2 |

Probable cause Errors were detected when MobiLink synchronization server applied the upload stream using multi-row cursors (bulk operation). It will roll back the upload stream and retry the upload transaction using single-row cursors.

Expecting %1!!d! parameter(s) in cursor, but found %2!!d!

| Item | Value |
|--------------|-------|
| Warning code | 10013 |
| Level | 4 |

Expecting at most %1!!d! parameter(s) in cursor, but found %2!!d!

| Item | Value |
|--------------|-------|
| Warning code | 10014 |
| Level | 4 |

If needed, ODBC cursors will be used, via the Microsoft ODBC Cursor Library, to simulate SQLSETPOS for inserts, updates, and deletes

| Item | Value |
|--------------|-------|
| Warning code | 10002 |
| Level | 1 |

Ignoring updated row (new values)

| Item | Value |
|--------------|-------|
| Warning code | 10037 |
| Level | 5 |

Ignoring updated row (old values)

| Item | Value |
|--------------|-------|
| Warning code | 10038 |
| Level | 5 |

Invalid character data encountered in upload -- substituting '?'

| Item | Value |
|--------------|-------|
| Warning code | 10025 |
| Level | 5 |

Invalid character data encountered in upload -- using NULL

| Item | Value |
|--------------|-------|
| Warning code | 10026 |
| Level | 5 |

Invalid character data encountered in upload -- using empty string

| Item | Value |
|--------------|-------|
| Warning code | 10027 |
| Level | 5 |

Maximum number of database connections set to %1!lu! (must be at least the number of worker threads plus one)

| Item | Value |
|--------------|---|
| Warning code | 10001 |
| Level | 1 |
| Parameter 1 | Maximum number of connections given in the MobiLink synchronization server command line |

Probable cause

MobiLink synchronization server makes one connection for each worker thread and an extra connection for main thread. Therefore, the maximum number of connections must be at least the number of worker threads plus one.

MobiLink table '%1!s!' is damaged

| Item | Value |
|--------------|-------|
| Warning code | 10009 |
| Level | 2 |

Multi-byte characters truncated on upload

| Item | Value |
|--------------|-------|
| Warning code | 10028 |
| Level | 5 |

NT Performance Monitor data area failed to initialize

| Item | Value |
|--------------|-------|
| Warning code | 10042 |
| Level | 1 |

No error-handling script is defined. The default action code (%1!ld!) will decide error behavior

| Item | Value |
|--------------|-------|
| Warning code | 10010 |
| Level | 2 |

ODBC Isolation level (%1!s!) is not supported

| Item | Value |
|--------------|------------------------------|
| Warning code | 10003 |
| Level | 1 |
| Parameter 1 | The required isolation level |

ODBC function %1!s! is not supported by the driver

| Item | Value |
|--------------|--------------------|
| Warning code | 10004 |
| Level | 1 |
| Parameter 1 | ODBC function name |

ODBC statement option %1!s! has changed from %2!lu! to %3!lu!

| Item | Value |
|--------------|-------|
| Warning code | 10006 |
| Level | 1 |

ODBC statement option %1!s! has changed from %2!s! (%3!lu!) to %4!s! (%5!lu!)

| Item | Value |
|--------------|-------|
| Warning code | 10005 |
| Level | 1 |

Publication '%1!s!' is not referenced by any table

| Item | Value |
|--------------|-------|
| Warning code | 10022 |
| Level | 3 |

Retry on deadlock set to FALSE. MobiLink synchronization server is using an internal work-around which requires this setting

| Item | Value |
|--------------|-------|
| Warning code | 10008 |
| Level | 2 |

Retrying the begin_connection transaction after deadlock in the consolidated database

| Item | Value |
|--------------|-------|
| Warning code | 10007 |
| Level | 2 |

Retrying the begin_synchronization transaction after deadlock in the consolidated database

| Item | Value |
|--------------|-------|
| Warning code | 10048 |
| Level | 2 |

Probable cause Deadlock occurred when MobiLink sever executed the begin_synchronization script. It will rollback the transaction and retry this script.

Retrying the end_synchronization transaction after deadlock in the consolidated database

| Item | Value |
|--------------|-------|
| Warning code | 10049 |
| Level | 2 |

Probable cause Deadlock occurred when MobiLink synchronization server executed the end_synchronization script. It will rollback the transaction and retry this script.

Retrying the upload after deadlock in the consolidated database

| Item | Value |
|--------------|-------|
| Warning code | 10045 |
| Level | 2 |

Probable cause Deadlock occurred when MobiLink synchronization server was applying the upload stream. It will rollback the transaction and retry this script.

Retrying the upload. Working around a known ODBC driver problem

| Item | Value |
|--------------|-------|
| Warning code | 10046 |
| Level | 2 |

Table '%1!s!' has at least one timestamp column. Due to a timestamp precision mismatch, downloaded timestamps can lose precision, resulting in inconsistent data

| Item | Value |
|--------------|-------|
| Warning code | 10016 |
| Level | 3 |

Table '%1!s!' has at least one timestamp column. Due to a timestamp precision mismatch, uploaded timestamps can lose precision, defeating download filtering

| Item | Value |
|--------------|-------|
| Warning code | 10015 |
| Level | 3 |

Table '%1!s!' has no entry in the %2!s! table

| Item | Value |
|--------------|-------|
| Warning code | 10024 |
| Level | 3 |

The consolidated and remote databases disagree on when the last synchronization took place. The remote is being asked to send a new upload that starts at the last known synchronization point

| Item | Value |
|--------------|-------|
| Warning code | 10012 |
| Level | 2 |

The consolidated and remote databases have different timestamp precisions. Consolidated database timestamps are precise to %1!d! digit(s) in the fractional second while the remote database timestamps are precise to %2!d! digit(s)

| Item | Value |
|--------------|-------|
| Warning code | 10017 |
| Level | 3 |

The remote and consolidated databases have different timestamp precisions, and a timestamp value with a precision higher than the lower-precision side was used for conflict detection purposes. Consider using the -zp option

| Item | Value |
|--------------|-------|
| Warning code | 10021 |
| Level | 3 |

The remote client's synchronization log ended prematurely, and was probably truncated

| Item | Value |
|--------------|-------|
| Warning code | 10035 |
| Level | 2 |

The remote database is not capable of matching the timestamp precision of the consolidated database. Your application, schema, and scripts must contain logic that copes with the precision mismatch

| Item | Value |
|--------------|-------|
| Warning code | 10019 |
| Level | 3 |

The timestamp precision mismatch may affect upload conflict detection. You can use the -zp option to cause MobiLink synchronization server to use the lowest timestamp precision for conflict detection purposes

| Item | Value |
|--------------|-------|
| Warning code | 10020 |
| Level | 3 |

The upload will be committed and the synchronization aborted. The next time this remote synchronizes, it will ask what happened to the previous upload

| Item | Value |
|--------------|-------|
| Warning code | 10041 |
| Level | 2 |

**The upload will be rolled back and the synchronization aborted.
The next time this remote synchronizes, it will ask what happened
to the previous upload**

| Item | Value |
|--------------|-------|
| Warning code | 10023 |
| Level | 3 |

Unable to convert character data for download -- substituting '?'

| Item | Value |
|--------------|-------|
| Warning code | 10029 |
| Level | 5 |

Unable to convert character data for download -- using NULL

| Item | Value |
|--------------|-------|
| Warning code | 10030 |
| Level | 5 |

**Unable to convert character data for download -- using empty
string**

| Item | Value |
|--------------|-------|
| Warning code | 10031 |
| Level | 5 |

Unable to determine current timestamp

| Item | Value |
|--------------|-------|
| Warning code | 10043 |
| Level | 4 |

Unable to open the file to store the client synchronization logs. The filename is '%1!s!'

| Item | Value |
|--------------|-------|
| Warning code | 10032 |
| Level | 2 |

Unable to write to the local file that contains remote synchronization logs

| Item | Value |
|--------------|-------|
| Warning code | 10034 |
| Level | 2 |

Unrecognized ODBC driver '%1!s!'. The functionality and quality of ODBC drivers varies greatly. This driver may lack functionality required for successful synchronizations. Use at your own risk

| Item | Value |
|--------------|------------------------------|
| Warning code | 10051 |
| Level | 1 |
| Parameter 1 | The file name of ODBC driver |

Unrecognized value (%1!d!) in ml_user.commit_state. The state information for this user is probably corrupted

| Item | Value |
|--------------|-------|
| Warning code | 10011 |
| Level | 2 |

You may resolve the timestamp precision mismatch by setting the DEFAULT_TIMESTAMP_INCREMENT option on the remote database to %1!d! and TRUNCATE_TIMESTAMP_VALUES to 'On'

| Item | Value |
|--------------|-------|
| Warning code | 10018 |
| Level | 3 |

A P P E N D I X A

ODBC Drivers

About this
appendix
Contents

This appendix describes the ODBC drivers available for use with MobiLink.

| Topic | Page |
|------------------------------------|------|
| ODBC drivers supported by MobiLink | 708 |

ODBC drivers supported by MobiLink

The following list shows ODBC drivers and databases used with each major operating system at the time of publication of this document. For updated information and complete functional specifications, see <http://www.sybase.com/detail?id=1011880>. The Web page data may contain driver information; exact version numbers, for example, are not present in the table below. The web page overrides this table data and is to be considered the most appropriate source for driver information.

MobiLink synchronization server can work with a variety of consolidated databases and ODBC drivers as shown in the table below. Some drivers, though compatible for use with MobiLink, may have functional restrictions associated with their use.

| Database | ODBC Driver |
|---|---|
| Oracle 8i | iAnywhere Solutions 8 - Oracle 8, 8i & 9i ODBC Driver Merant DataDirect Connect ODBC Driver for Oracle |
| Oracle 9i | iAnywhere Solutions 8 - Oracle 8, 8i & 9i ODBC Driver Oracle 9i ODBC Driver |
| Microsoft SQL Server 7, Microsoft SQL Server 2000 | Microsoft SQL Server ODBC Driver Merant DataDirect SQL Server Wire Protocol ODBC Driver |
| Sybase Adaptive Server Enterprise 11.5 or later | iAnywhere Solutions 8 - Sybase ASE ODBC Driver Sybase ASE ODBC Driver Merant DataDirect Sybase Wire Protocol ODBC Driver Merant DataDirect Connect ODBC Driver for Sybase ASE Merant Connect ODBC Driver for Sybase ASE |
| IBM DB2 UDB 7.1, 7.2 | IBM DB2 UDB 7.1 ODBC driver IBM DB2 UDB 7.2 ODBC driver Merant DataDirect DB2 Wire Protocol ODBC Driver |
| Sybase Adaptive Server Anywhere 8 | Adaptive Server Anywhere 8.0 ODBC Driver |

C H A P T E R 2 7

Deploying MobiLink Applications

About this chapter

This chapter describes how to deploy the MobiLink server and MobiLink clients in a production environment. It identifies the files required for deployment.

Check your license agreement
Redistribution of files is subject to your license agreement. No statements in this document override anything in your license agreement. Please check your license agreement before considering deployment.

Contents

| Topic | Page |
|---|------|
| Deployment overview | 712 |
| Deploying the MobiLink server | 713 |
| Deploying Adaptive Server Anywhere MobiLink clients | 714 |
| Deploying UltraLite MobiLink clients | 715 |

Deployment overview

Deploying MobiLink applications involves the following activities:

- ◆ Deploy the MobiLink server into a production setting.
- ◆ Deploy any Adaptive Server Anywhere MobiLink clients.
- ◆ Deploy any UltraLite MobiLink clients.

This chapter describes the files you need to include in your application's install program for each of these items.

Deploying the MobiLink server

The simplest way to deploy a MobiLink synchronization server into a production environment is to install a licensed copy of SQL Anywhere Studio onto the production machine.

If you are redistributing MobiLink synchronization server in a separate install program (subject to your license agreement) you need to include the following files in your installation:

The files should be in the same directory, unless otherwise mentioned.

| Description | Windows | UNIX |
|---|--|--|
| MobiLink synchronization server | <i>dbmlsrv8.exe</i> <i>dbmlsv8.dll</i> <i>libunic.dll</i> | <i>dbmlsrv8</i> <i>dbmlsv8.so</i> |
| Language library | <i>dblgXX8.dll</i> ¹ | <i>dblgXX8.so</i> ¹ |
| Windows Monitor support | <i>dbmlctr8.dll</i> ² <i>dbmlctr8.ini</i> <i>dbmlctr8.h</i> | N/A |
| Synchronization stream libraries (deploy the ones you use) | <i>dbsock8.dll</i> <i>dbhttp8.dll</i> <i>dbhttps8.dll</i> <i>dbrsa8.dll</i> | <i>dbsock8.so</i> <i>dbhttp8.so</i> <i>dbhttps8.so</i> <i>dbrsa8.so</i> |
| Security option (separately licensable) | <i>dbtls8.dll</i> | <i>dbtls8.so</i> |
| Script files (deploy the ones for your consolidated database) | %ASANY8%\MobiLink\ <i>setup</i> %ASANY8%\MobiLink\ <i>upgrade</i> | %ASANY8%\MobiLink\ <i>setup</i> %ASANY8%\MobiLink\ <i>upgrade</i> |

¹ XX is a two-letter identifier for the language in which informational and error messages are displayed: EN, FR, DE, JA, ZH, and so on.

² Your setup program must self-register this file.

Deploying Adaptive Server Anywhere MobiLink clients

For Adaptive Server Anywhere clients, you need to deploy an Adaptive Server Anywhere database server and the MobiLink client.

☞ For information on deploying Adaptive Server Anywhere databases, see "Deploying Databases and Applications" on page 373 of the book *ASA Programming Guide*.

If you are redistributing MobiLink synchronization clients (subject to your license agreement) you need to include the following files in your installation in addition to those required for the Adaptive Server Anywhere database:

The files should be in the same directory, unless otherwise mentioned.

| Description | Windows | UNIX |
|--|--|--|
| MobiLink synchronization client | <i>dbmlsync.exe</i> <i>dbtool8.dll</i> <i>dblgen8.dll</i> | <i>dbmlsync</i> <i>dbtool8.so</i> <i>dblgen8.so</i> |
| Synchronization stream libraries (deploy the ones you use) | <i>dbsock8.dll</i> <i>dbhttp8.dll</i> <i>dbhttps8.dll</i> <i>dbrsa8.dll</i> | <i>dbsock8.so</i> <i>dbhttp8.so</i> <i>dbhttps8.so</i> <i>dbrsa8.so</i> |
| Security option (separately licensable) | <i>dbtls8.dll</i> | <i>dbtls8.so</i> |

Deploying UltraLite MobiLink clients

For UltraLite clients, the UltraLite runtime library includes the required synchronization stream functions. The UltraLite runtime library is either compiled into your application or is provided as a *ulrt8.dll* for Windows CE clients. Deployment is subject to your license agreement.

☞ For information on deploying UltraLite applications, see "Deploying UltraLite applications" on page 104 of the book *UltraLite User's Guide*.

UltraLite clients developed with the UltraLite Component Suite do not require the UltraLite runtime, as it is compiled into the application.

- ◆ For AppForge MobileVB applications you must deploy *ulingot8.dll* (Windows CE) or *ulingot8.prc* (Palm Computing Platform), which can be found in subdirectories of the *UltraLite\UltraLiteForMobileVB* subdirectory of your SQL Anywhere installation.
- ◆ For eMbedded Visual Basic applications you must deploy *uldo8.dll*, which can be found in subdirectories of the *UltraLite\UltraLiteForActiveX* subdirectory of your SQL Anywhere installation.
- ◆ For Native UltraLite for Java applications, you must deploy *jul8.jar* and *jul8.dll*, which can be found in subdirectories of the *UltraLite\Native UltraLite for Java* subdirectory of your SQL Anywhere installation.

☞ For more information, see the UltraLite Component Suite documentation. Click Start ► Programs ► SQL Anywhere 8 ► UltraLite ► Online Books.

Index

#

#hook_dict table
 about, 160
 dbmlsync, 592
 unique primary keys, 98

▪

.NET
 about support in MobiLink, 187
 MobiLink API reference, 203
 MobiLink data types, 195
 synchronization logic, 38
 synchronization scripts for MobiLink, 187

.NET classes
 instantiation for .NET synchronization logic, 194

.NET CLR
 MobiLink options, 390

.NET MobiLink API
 API reference, 203
 benefits, 41

.NET synchronization logic
 .NET class instantiations, 194
 about, 38
 API, 203
 DBCommand, 206
 DBConnection, 211
 DBConnectionContext, 205
 DBParameter, 211
 DBParameterCollection, 213
 DBRowReader, 216
 InOutInteger, 205
 methods, 196

sample, 200
ServerContext, 203
ServerException, 204
setup, 189
ShutdownCallback, 204
SQLType, 208
supported languages, 188

@

@EmployeeID variable
 using with primary key pools, 101

@LastDownload timestamp variable, 86

A

-a option
 MobiLink [dbmlsrv8], 383

-ac option
 MobiLink [mlxtract], 614

ActiveSync
 about, 143
 class name for dbmlsync, 431
 CREATE SYNCHRONIZATION USER
 statement, 144
 installing the MobiLink provider, 145, 610
 MobiLink ActiveSync provider [dbasinst], 610
 MobiLink remote databases, 144
 registering applications, 146

ActiveSync provider installation utility [dbasinst]
 syntax, 610

Adaptive Server Anywhere
 as MobiLink clients, 21
 as MobiLink consolidated databases, 14
 version differences, 132

- Adaptive Server Anywhere clients
 - dbmsync, 410
 - MobiLink, 117
- Adaptive Server Enterprise
 - as MobiLink consolidated databases, 14
 - conversion of data types in MobiLink
 - synchronization, 626
 - MobiLink synchronization, 81
 - StaticCursorLongColBuffLen, 81
- add connection script wizard
 - using, 63
- add service wizard
 - using, 277
- add synchronized table wizard
 - using, 63
- add synchronizing table script wizard
 - using, 64, 335
- add user wizard
 - using, 257
- add version wizard
 - using, 62, 335
- adding
 - articles, 122
 - columns to remote MobiLink databases, 116
 - elliptic-curve and RSA certificates, 621
 - MobiLink .NET connection scripts, 588
 - MobiLink .NET table scripts, 589
 - MobiLink Java connection scripts, 589
 - MobiLink Java table scripts, 590
 - MobiLink SQL connection scripts, 586
 - MobiLink SQL table scripts, 587
 - MobiLink users to a remote database, 125
 - synchronization scripts with Sybase Central, 63
 - tables to remote MobiLink databases, 116
 - user names in MobiLink, 618
- adding a script version, 62
- adding and deleting scripts in your consolidated database, 63
- adding MobiLink users to a remote database, 125
- adding synchronization scripts
 - using stored procedures, 64
- al option
 - MobiLink [mlxtract], 614
- altering
 - articles, 122
 - publications, 122
 - synchronization subscriptions, 129
- altering MobiLink subscriptions, 129
- an option
 - MobiLink [mlxtract], 614
- Apache
 - configuring servlet Redirector for MobiLink, 273
- API reference
 - MobiLink .NET API, 203
 - MobiLink Java API, 183
- applications
 - deploying, 711
 - differentiating MobiLink scripts, 61
- article creation wizard
 - using, 123
- articles
 - adding, 122
 - altering, 122
 - creating, 119
 - MobiLink synchronization subscriptions, 128
 - removing, 122
- assemblies
 - implementing in MobiLink, 191
 - locating in MobiLink .NET synchronization logic, 189
- auth_status synchronization parameter
 - about, 446
- authenticate_user
 - about, 261
 - connection event, 446
- authenticate_user_hashed
 - connection event, 450
- authenticating
 - users in MobiLink, 618
- automated script generation
 - MobiLink, 322
- automatic synchronization script generation, 50
- automating scripts
 - MobiLink synchronization, 50

B

- bc option
 - MobiLink [dbmlsrv8], 384
- begin_connection
 - connection event, 452
 - example, 364
- begin_download
 - connection event, 454
 - table event, 456
- begin_download_deletes
 - table event, 458
- begin_download_rows
 - table event, 460
- begin_synchronization
 - connection event, 462
 - table event, 464
- begin_upload
 - connection event, 466
 - table event, 468
- begin_upload_deletes
 - table event, 470
- begin_upload_rows
 - table event, 472
- blob cache size
 - MobiLink performance, 222
- BLOBs
 - downloaded from Adaptive Server Enterprise, 81
- bn option
 - MobiLink [dbmlsrv8], 384
- bottlenecks
 - MobiLink performance, 224
- buffering
 - MobiLink downloads, 221

C

- c option
 - MobiLink [dbmlsrv8], 384
 - MobiLink [dbmlsync], 413
 - MobiLink [dbmluser], 618
- MobiLink [gencert], 621
- MobiLink [mlxtract], 614
- C#
 - MobiLink options, 390
 - support in MobiLink .NET, 188
- C++
 - support in MobiLink .NET, 188
- cascading deletes
 - during MobiLink synchronization, 35
- Certicom
 - obtaining certificates, 307
- certificate authorities, 298
- certificate chains, 299
- certificate generation utility [gencert]
 - syntax, 621
- certificate reader utility [readcert]
 - syntax, 620
- certificate stream parameter
 - HTTP synchronization, 399
 - HTTPS synchronization, 401
 - TCP/IP synchronization, 398
- certificate_password stream parameter
 - HTTP synchronization, 399
 - HTTPS synchronization, 401
 - TCP/IP synchronization, 398
- certificates
 - generating elliptic-curve, 621
 - generating RSA, 621
 - reading elliptic-curve, 620
 - reading RSA, 620
 - sample certificates for MobiLink, 293
- chains of certificates
 - using, 290
- CHAR data type
 - MobiLink and other DBMSs, 82
- character sets
 - MobiLink synchronization, 42

- character-set translation
 - by ODBC drivers, 44
 - during MobiLink synchronization under other platforms, 44
 - during MobiLink synchronization under Windows, 42
- ciphers
 - MobiLink transport-layer security, 284
- class names
 - ActiveSync, 431
- CLASSPATH environment variable
 - MobiLink Java synchronization logic, 167
- classpath option
 - MobiLink [dbmlsrv8] -sl dnet, 390
 - MobiLink [dbmlsrv8] -sl java, 391
- client database extraction utility [mlxtract]
 - syntax, 614
- client event-hook procedures, 592
- client_port, 397
- client_port stream parameter
 - HTTP synchronization, 398
- clients
 - Adaptive Server Anywhere as MobiLink, 21
 - Adaptive Server Anywhere MobiLink clients, 117
 - dbmlsync, 410
 - MobiLink synchronization, 21
 - UltraLite applications as MobiLink, 21
- CLR
 - MobiLink options, 390
- cn option
 - MobiLink [dbmlsrv8], 385
- collation sequences
 - MobiLink synchronization, 42
- columns
 - adding to remote MobiLink databases, 116
- command line
 - starting dbmlsrv8, 380
 - starting dbmlsync, 410
- command line utilities
 - dbasinst command line syntax, 610
 - MobiLink certificate generator [gencert], 621
 - MobiLink client database extraction [mlxtract], 614
 - MobiLink stop utility [dbmlstop], 613
 - MobiLink synchronization, 609
 - MobiLink user authentication [dbmluser], 618
 - readcert syntax, 620
- COMMIT statement
 - event-hook procedures, 592
- commit_state column
 - about, 22
- common language runtime
 - MobiLink options, 390
- CommunicationAddress
 - about, 414
- communications
 - specifying for MobiLink, 22
- communications faults
 - MobiLink synchronization recovery, 32, 33
- CommunicationType
 - about, 414
- concurrency
 - MobiLink performance, 220
 - MobiLink synchronization, 140
 - MobiLink upload-stream processing, 34
- configuring
 - Microsoft Web servers, 272
 - Netscape Web servers, 270
 - Redirectors (all versions), 268
 - servlet Redirector, 273
 - Tomcat, 273
- configuring Adaptive Server Anywhere remote databases for ActiveSync, 144
- configuring MobiLink user properties, 126
- conflict detection
 - cursor-based uploads, 106
 - MobiLink, 104
 - MobiLink statement-based uploads, 105
- conflict resolution
 - Contact sample, 374
 - cursor-based uploads, 106
 - forcing in MobiLink, 107
 - MobiLink, 104
 - MobiLink conflict detection, 104

- MobiLink statement-based uploads, 105
- user-specific logic, 108
- ConflictRetries synchronization option
 - about, 141, 414
- conflicts
 - MobiLink, 104
- connection parameters
 - priority order, 129
- connection scripts
 - about, 58
 - adding .NET scripts, 588
 - adding Java scripts, 589
 - adding SQL scripts, 586
 - adding with Sybase Central, 63
 - defined, 55
 - deleting .NET scripts, 588
 - deleting Java scripts, 589
 - deleting SQL scripts, 586
- consolidated databases
 - Adaptive Server Anywhere as MobiLink, 14
 - Adaptive Server Enterprise as MobiLink, 14
 - adding synchronization scripts to, 63
 - creating MobiLink, 13
 - databases other than Adaptive Server Anywhere, 80
 - DBMS dependencies, 80
 - IBM DB2 as MobiLink, 14
 - MobiLink, 12
 - MobiLink system tables, 15
 - MobiLink user names, 22
 - Oracle as MobiLink, 14
 - relating tables to MobiLink remote tables, 13
 - requirements for, 12
 - SQL Server as MobiLink, 14
- constructors
 - MobiLink synchronization, 171, 195
- Contact MobiLink sample
 - about, 365
 - building, 365
 - Contact table, 372
 - Customer table, 370
 - monitoring statistics, 375
 - Product table, 374
 - running, 366
 - SalesRep table, 370
 - tables, 367
 - users, 369
- contd_timeout stream parameter
 - HTTP synchronization, 399
 - HTTPS synchronization, 401
 - synchronizing across firewalls, 266
- contention
 - MobiLink performance, 220
 - MobiLink performance explanation, 225
- conventions
 - documentation, xvii
- conversion
 - of data types in MobiLink synchronization, 625
 - to Adaptive Server Enterprise data types in MobiLink synchronization, 626
 - to IBM DB2 data types in MobiLink synchronization, 627
 - to Microsoft SQL Server data types in MobiLink synchronization, 630
 - to Oracle data types in MobiLink synchronization, 629
- cp option
 - MobiLink [dbmlsrv8] -sl dnet, 390
 - MobiLink [dbmlsrv8] -sl java, 391
- cr option
 - MobiLink [dbmlsrv8], 385
- create database wizard
 - using, 331
- CREATE SYNCHRONIZATION SUBSCRIPTION statement
 - ActiveSync, 144
- CREATE SYNCHRONIZATION USER statement
 - ActiveSync, 144
- creating
 - Adaptive Server Anywhere remote databases, 118
 - articles, 119
 - MobiLink client databases, 614
 - MobiLink consolidated databases, 13
 - MobiLink users in remote databases, 125
 - new certificates, 621
 - publications, 119
 - publications with column-wise partitioning, 120
 - publications with row-wise partitioning, 121
 - publications with whole tables, 119
- creating a consolidated databases, 13

creating a remote database
Adaptive Server Anywhere clients, 118

creating MobiLink users, 125

creating the certificates, 301

cryptography
public key, 284

-ct option
MobiLink [dbmlsrv8], 385

cursor scripts
defined, 55
list, 55

cursor-based scripts
about, 66

cursor-based uploads
conflict detection, 106
performance, 221

custase.sql
location, 363

CustDB application
synchronization scripts, 363

CustDB database
DB2, 362

custdb.sqc
location, 363

custdb.sql
location, 363

custmss.sql
location, 363

customizing
MobiLink, 157

customizing a prototype remote database, 148

custora.sql
location, 363

D

-d option
MobiLink [dbasinst], 610
MobiLink [dbmlsrv8], 386

MobiLink [dbmlsync], 413
MobiLink [dbmluser], 618

daemon
running MobiLink as a, 275

data entry
synchronization techniques, 110

data sources
ODBC for MobiLink synchronization, 15

data types
conversion of in MobiLink synchronization, 625
conversion to Adaptive Server Enterprise in
MobiLink synchronization, 626
conversion to IBM DB2 in MobiLink
synchronization, 627
conversion to Microsoft SQL Server in MobiLink
synchronization, 630
conversion to Oracle in MobiLink
synchronization, 629
MobiLink .NET and SQL, 195
MobiLink Java and SQL, 171

database connections
MobiLink performance, 227

database extraction utility
MobiLink, 614

database schemas
relating consolidated tables to MobiLink remote
tables, 13

databases
MobiLink consolidated, 12
MobiLink synchronization requirements for
consolidated, 12
synchronizing with MobiLink, 9, 283

DB2
as MobiLink consolidated databases, 14
consolidated database, 362
conversion of data types in MobiLink
synchronization, 627
CustDB database, 362
maximum identifier length in IBM, 15
session-wide variables, 81

dbasdesk.dll
installing, 610

dbasdev.dll
installing, 610

- dbasinst utility
 - installing the MobiLink provider for ActiveSync, 145
 - options, 610
 - syntax, 610
- DBCommand
 - MobiLink .NET API, 206
- DBConnection
 - MobiLink .NET API, 211
- DBConnectionContext
 - MobiLink .NET API, 205
 - MobiLink Java API, 185
- dbmlsrv8
 - automating script generation, 50
 - options, 380
 - reports error context in output log, 387
 - using, 18
- dbmlstop utility
 - MobiLink, 19
 - options, 613
 - syntax, 613
 - using, 18
- dbmlsync
 - event hooks, 592
- dbmlsync utility
 - #hook_dict table, 592
 - ActiveSync usage, 143
 - changing passwords, 260
 - concurrency, 141
 - customizing MobiLink synchronization, 157
 - d option, 141
 - example, 138
 - extended options, 414
 - multiple users, 139
 - options, 410
 - passwords, 259
 - permissions, 138
 - sp_hook_dbmlsync_abort hook, 593
 - sp_hook_dbmlsync_begin, 594
 - sp_hook_dbmlsync_delay, 595
 - sp_hook_dbmlsync_download_begin, 596
 - sp_hook_dbmlsync_download_com_error, 596
 - sp_hook_dbmlsync_download_end, 597
 - sp_hook_dbmlsync_download_fatal_sql_error, 598
 - sp_hook_dbmlsync_download_log_ri_violation, 599
 - sp_hook_dbmlsync_download_ri_violation, 600
 - sp_hook_dbmlsync_download_sql_error, 601
 - sp_hook_dbmlsync_download_table_begin, 602
 - sp_hook_dbmlsync_download_table_end, 602
 - sp_hook_dbmlsync_end, 603
 - sp_hook_dbmlsync_logscan_begin, 604
 - sp_hook_dbmlsync_logscan_end, 605
 - sp_hook_dbmlsync_upload_begin, 606
 - sp_hook_dbmlsync_upload_end, 607
 - syntax, 410
 - transaction logs, 140
 - using, 138
 - writing your own, 141
- dbmluser utility
 - options, 618
 - syntax, 618
 - using, 259
- DBMS-dependent scripts, 80
- DBParameter
 - MobiLink .NET API, 211
- DBParameterCollection
 - MobiLink .NET API, 213
- DBRowReader
 - MobiLink .NET API, 216
- dbtools.h
 - dbmlsync features, 141
 - synchronization, 141
- DDL statements
 - remote MobiLink databases, 116
- deadlocks
 - MobiLink upload-stream processing, 34
- debugging
 - MobiLink connections, 282
 - MobiLink synchronization server log, 19
 - MobiLink synchronization using Java classes, 173
- DECIMAL data type
 - MobiLink and Adaptive Server Enterprise, 82
- default global autoincrement
 - declaring, 97
- default_download_cursor
 - about, 77

- default_upload_cursor
 - about, 77
- deletes
 - stopping upload of using MobiLink, 156
- deleting
 - articles, 122
 - MobiLink .NET connection scripts, 588
 - MobiLink .NET table scripts, 589
 - MobiLink Java connection scripts, 589
 - MobiLink Java table scripts, 590
 - MobiLink SQL connection scripts, 586
 - MobiLink SQL table scripts, 587
 - publications, 124
- deleting rows
 - synchronization, 72
 - synchronization techniques, 111
- deleting rows with the download_delete_cursor script, 72
- deploying
 - about, 711
 - Adaptive Server Anywhere MobiLink clients, 714
 - applications and databases, 711
 - MobiLink remote database sample, 148
 - MobiLink remote databases, 148
 - MobiLink synchronization server, 713
 - troubleshooting MobiLink deployment, 154
 - UltraLite applications, 715
- deprecated features
 - MobiLink differences from version 7, 132
- development tips
 - synchronization, 85
- digital certificates, 287
- direct inserts of scripts, 64
- disjoint partitioning
 - defined, 91
 - synchronization, 91
- dl option
 - MobiLink [dbmlsrv8], 386
 - MobiLink [dbmlsync], 413
 - MobiLink [dbmluser], 618
- DMLStartClass
 - Java user-defined start classes, 174
- DMLStartClasses
 - MobiLink [dbmlsrv8] -sl java, 391
- documentation
 - conventions, xvii
 - SQL Anywhere Studio, xiv
- domain configuration files
 - about, 192
- download acknowledgement
 - MobiLink performance, 221
- download buffer
 - MobiLink performance, 221
- download cache size
 - MobiLink performance, 222
- download events
 - MobiLink synchronization, 444
- download stream
 - defined, 24
 - events, 70
 - failed downloads, 112
 - MobiLink performance, 223
 - MobiLink transactions, 32
- download_cursor
 - timestamp-based synchronization, 87
- download_cursor
 - about, 55
 - Contact sample, 372, 374
 - cursor event, 474
 - disjoint partitioning, 91
 - example, 364
 - example using a stored procedure call, 113
 - partitioning child tables, 94
 - partitioning with overlaps, 93
 - performance, 223
 - using a stored procedure call, 113
- download_cursor table script
 - Contact sample, 371
- download_delete_cursor
 - about, 55, 72
 - Contact sample, 371, 373, 374
 - cursor event, 477
 - disjoint partitioning, 91
 - example using a stored procedure call, 113
 - partitioning child tables, 94
 - partitioning with overlaps, 93

- performance, 223
 - using a stored procedure call, 113
- download_delete_cursor timestamp-based
 - synchronization, 87
- download_statistics
 - connection event, 479
 - table event, 482
- DownloadBufferSize
 - about, 414
- downloading a result set from a stored procedure
 - call
 - synchronization techniques, 113
- downloading rows
 - resolving MobiLink RI violations, 599
 - synchronization scripts, 70
- DROP PUBLICATION statement
 - about, 124
- DROP SYNCHRONIZATION SUBSCRIPTION
 - statement
 - about, 130
- dropping
 - MobiLink subscriptions, 130
 - MobiLink users from a remote database, 127
- dropping publications, 124

E

- e option
 - MobiLink [dbmlsrv8], 386
 - MobiLink [dbmlsync], 414
- eh option
 - MobiLink [dbmlsync], 423
- ek option
 - MobiLink [dbmlsync], 423
- elliptic-curve certificates
 - generating, 621
 - reading, 620
- encryption
 - MobiLink, 284
- end_connection
 - connection event, 485
- end_download
 - connection event, 487
 - table event, 489
- end_download_deletes
 - table event, 491
- end_download_rows
 - table event, 493
- end_synchronization
 - connection event, 495
 - table event, 497
- end_upload
 - connection event, 499
 - table event, 502
- end_upload_deletes
 - table event, 504
- end_upload_rows
 - table event, 506
- enterprise root certificates, 300
 - creating, 301
- ep option
 - MobiLink [dbmlsync], 423
- error handling
 - during MobiLink synchronization, 75
- error logs
 - MobiLink server [dbmlsrv8], 386
- ErrorLogSendLimit synchronization option
 - about, 414
- errors
 - handling during MobiLink synchronization, 75
 - multiple, 76
 - recording, 75
- et option
 - MobiLink [dbmlsrv8], 387
- eu option
 - MobiLink [dbmlsync], 423
- event hooks
 - #hook_dict table, 160
 - commits not allowed, 592
 - connections, 159
 - event arguments, 160
 - fatal errors, 159
 - ignoring errors, 162

- MobiLink, 157
- procedure owner, 159
- rollbacks not allowed, 592
- sp_hook_dbmlsync_abort, 593
- sp_hook_dbmlsync_begin, 594
- sp_hook_dbmlsync_delay, 595
- sp_hook_dbmlsync_download_begin, 596
- sp_hook_dbmlsync_download_com_error, 596
- sp_hook_dbmlsync_download_fatal_SQL_error, 598
- sp_hook_dbmlsync_download_log_ri_violation, 599
- sp_hook_dbmlsync_download_ri_violation, 600
- sp_hook_dbmlsync_download_sql_error, 601
- sp_hook_dbmlsync_download_table_begin, 602
- sp_hook_dbmlsync_download_table_end, 602
- sp_hook_dbmlsync_end, 603
- sp_hook_dbmlsync_logscan_begin, 604
- sp_hook_dbmlsync_logscan_end, 605
- sp_hook_dbmlsync_upload_begin, 606
- sp_hook_dbmlsync_upload_end, 607
- synchronization event hooks, 592
- using, 159

event names

- defined, 48

event-hooks

- sp_hook_dbmlsync_begin, 596
- sp_hook_dbmlsync_download_end, 597

events

- about MobiLink synchronization, 436
- Adaptive Server Anywhere client, 157
- introduction to MobiLink events, 48
- MobiLink, 25, 433
- synchronization logic and, 48

events during download, 70

events during upload, 66

example scripts

- generating, 51

example scripts for UltraLite, 77

example synchronization script generation, 51

example_download_cursor

- about, 77

example_upload_cursor

- about, 77
- cursor event, 508

example_upload_delete

- table event, 509

example_upload_insert

- table event, 510

example_upload_update

- table event, 511

examples

- synchronization scripts, 77

extended options

- configuring at remote databases, 126
- dbmlsync, 414
- MobiLink synchronization, 162
- priority order, 129

extended options for performance tuning

- MobiLink, 139

extracting

- MobiLink client databases, 614

extracting remote databases, 149

extraction utility

- MobiLink, 614

F

-f option

- MobiLink [dbmlsrv8], 387
- MobiLink [dbmlstop], 613
- MobiLink [dbmluser], 618

failed downloads

- synchronization techniques, 112

failover

- Redirector, 264

fault recovery

- MobiLink, 32

faults

- MobiLink synchronization recovery, 32, 33

feedback

- documentation, xxi
- providing, xxi

FireTriggers synchronization option

- about, 414

firewalls

- configuring MobiLink clients, 266
- configuring MobiLink synchronization server, 266
- routing requests, 264

FOR UPDATE clause

- SELECT statement, 68

forced conflict resolution

- cursor-based uploads, 108
- MobiLink, 107
- MobiLink statement-based uploads, 107

forcing conflicts

- cursor-based uploads, 108
- MobiLink, 107
- MobiLink statement-based uploads, 107

G

gencert utility

- options, 621
- syntax, 621

generating

- elliptic-curve certificates, 621
- RSA certificates, 621

generating example scripts, 51

generating scripts automatically, 50

getServerContext method

- DBConnectionContext class, 186, 205

global assembly cache

- implementing in MobiLink, 191

global autoincrement

- algorithm, 99
- declaring, 97
- setting GLOBAL_DATABASE_ID, 98
- using to generate unique values, 96

GLOBAL_DATABASE_ID option

- setting in MobiLink, 98

globally signed certificates, 305

H

-h option

- MobiLink [dbmlstop], 613

handle_error

- connection event, 512
- synchronization scripts, 75

handle_odbc_error

- connection event, 515

handling deletes

- synchronization techniques, 111

handling failed downloads

- synchronization techniques, 112

handling multiple errors on a single SQL statement, 76

hooks

- Adaptive Server Anywhere client, 157
- sp_hook_dbmlsync_abort, 593
- sp_hook_dbmlsync_begin, 594
- sp_hook_dbmlsync_delay, 595
- sp_hook_dbmlsync_download_begin, 596
- sp_hook_dbmlsync_download_com_error, 596
- sp_hook_dbmlsync_download_end, 597, 603
- sp_hook_dbmlsync_download_fatal_sql_error, 598
- sp_hook_dbmlsync_download_log_ri_violation, 599
- sp_hook_dbmlsync_download_ri_violation, 600
- sp_hook_dbmlsync_download_sql_error, 601
- sp_hook_dbmlsync_download_table_begin, 602
- sp_hook_dbmlsync_download_table_end, 602
- sp_hook_dbmlsync_logscan_begin, 604
- sp_hook_dbmlsync_logscan_end, 605
- sp_hook_dbmlsync_upload_begin, 606
- sp_hook_dbmlsync_upload_end, 607
- synchronization event hooks, 592

host stream parameter

- HTTP synchronization, 399
- HTTPS synchronization, 401
- synchronizing across firewalls, 266
- TCP/IP synchronization, 397

how remote tables relate to consolidated tables, 13

HTTP

- dbmlsrv8 -x command line option, 398
- synchronization parameters, 397

HTTPS

- dbmlsrv8 -x command line option, 400
- synchronization parameters, 397

I**-i option**

- MobiLink [dbmlsync], 424

iaredirect.dll

- configuring the ISAPI Redirector, 272
- configuring the NSAPI Redirector, 270

IBM DB2

- as MobiLink consolidated databases, 14
- conversion of data types in MobiLink synchronization, 627
- maximum identifier length in, 15
- session-wide variables, 81

icons

- used in manuals, xviii

-id option

- MobiLink [mlxtract], 614

identifiers

- maximum length in IBM DB2, 15

IIS

- configuring for ISAPI, 272

Increment synchronization option

- about, 414

indexes

- MobiLink performance, 223

initiating

- MobiLink synchronization from UltraLite applications, 21
- synchronization, 138

initiating synchronization from an application, 141**InOutByteArray**

- MobiLink Java API, 183

InOutInteger

- MobiLink Java API, 183

InOutString

- MobiLink Java API, 183

inserting

- scripts in MobiLink, 64

installing

- MobiLink provider for ActiveSync, 145
- servlets into EAServer, 273

introduction to synchronization scripts, 48**invoking transport-layer security, 293****iPlanet**

- configuring for the NSAPI Redirector, 270

-is option

- MobiLink [dbmlsync], 424

ISAPI Redirector

- configuring, 272

isolation level

- MobiLink, 24

-it option

- MobiLink [mlxtract], 614

J**-j option**

- MobiLink [mlxtract], 614

Java

- MobiLink data types, 171
- MobiLink Java API reference, 183
- synchronization logic, 38
- synchronization scripts for MobiLink, 165

Java classes

- instantiation for Java synchronization logic, 170

Java MobiLink API

- benefits, 40

Java synchronization logic

- about, 38
- API, 183
- DBConnectionContext, 185
- InOutByteArray, 183
- InOutInteger, 183, 186
- InOutString, 183
- Java class instantiations, 170
- methods, 172
- sample, 177
- ServerContext, 184
- ServerException, 185

- setup, 167, 261
- ShutdownListener, 185
- specifying in MobiLink server command line, 169

Java VM

- MobiLink options, 391

Java vs. SQL synchronization logic

- MobiLink performance, 223

Javadoc

- MobiLink, 183

-jrepath option

- MobiLink [dbmlsrv8] -sl dnet, 390
- MobiLink [dbmlsrv8] -sl java, 391

K

-k option

- MobiLink [dbasinst], 610
- MobiLink [dbmlsync], 424

keep_alive stream parameter

- HTTP synchronization, 399
- HTTPS synchronization, 401
- TCP/IP synchronization, 398

key pools

- MobiLink synchronization application, 100

L

-l option

- MobiLink [dbmlsync], 424
- MobiLink [mlxtract], 614

last download timestamp

- Contact sample, 372
- maintaining, 372
- modify_last_download_timestamp connection event, 517
- modify_next_last_download_timestamp connection event, 519
- script parameter, 60
- timestamp-based synchronization, 86

library functions

- ULSynchronize, 21

loading assemblies in MobiLink, 191

locking

- MobiLink synchronization, 140

LockTables synchronization option

- about, 141, 414

log files

- MobiLink, 322, 342
- MobiLink synchronization server, 19

logging

- MobiLink performance, 222
- MobiLink RI violations, 599
- MobiLink synchronization server actions, 19

LONG data type

- Oracle synchronization, 629

M

- maintaining unique primary keys using global autoincrement, 96

- maintaining unique primary keys using key pools, 100

- maintaining unique primary keys using UUIDs, 95

- making a new self-signed certificate, 295

many-to-many relationships

- partitioning, 92
- synchronization, 92

Memory synchronization option

- about, 414

Microsoft SQL Server

- as MobiLink consolidated databases, 14
- conversion of data types in MobiLink synchronization, 630
- stored procedure calls, 81

ML directive

- Redirector, 268

ml_add_connection_script stored procedure

- SQL syntax, 586

ml_add_dnet_connection_script stored procedure

- SQL syntax, 588

ml_add_dnet_table_script stored procedure

- SQL syntax, 589

- ml_add_java_connection_script stored procedure
 - SQL syntax, 589
- ml_add_java_table_script stored procedure
 - SQL syntax, 590
- ml_add_table_script stored procedure
 - SQL syntax, 587
- ML_CLIENT_TIMEOUT directive
 - Redirector, 268
- ml_connection_script
 - MobiLink system table, 15
- ml_script
 - MobiLink system table, 16
- ml_script_version
 - MobiLink system table, 16
- ml_scripts_modified
 - MobiLink system table, 16
- ml_subscription
 - MobiLink system table, 16
- ml_table
 - MobiLink system table, 17
- ml_table_script
 - MobiLink system table, 17
- ml_user
 - installing a remote database over an old one, 154
 - MobiLink system table, 17
- ml_username
 - about, 22
- MLAutoLoadPath option
 - about, 191
 - MobiLink [dbmlsrv8] -sl dnet, 390
- mlDomConfig.xml
 - about, 192
- MLDomConfigFile option
 - about, 191
 - MobiLink [dbmlsrv8] -sl dnet, 390
- mlMonitorSettings
 - MobiLink Monitor settings, 240
- MLStartClasses
 - .NET user-defined start classes, 197
- MLStartClasses
 - MobiLink [dbmlsrv8] -sl dnet, 390

- mlxtract utility
 - options, 614
 - sp_hook_dbxtract_begin procedure, 98
 - syntax, 614
 - using, 149
- mn option
 - MobiLink [dbmlsync], 425
- MobiLink
 - .NET synchronization logic, 187
 - a simple synchronization script, 49
 - Adaptive Server Anywhere clients, 117
 - architecture, 10
 - characteristics, 5
 - common terms used in MobiLink
 - synchronization, 7
 - complete event model, 436
 - configuring Web servers, 264
 - database connections, 227
 - deprecated features from version 7, 132
 - development tips, 85
 - events, 433
 - events during download, 70
 - features, 4
 - isolation level, 24
 - Java synchronization logic, 165
 - key factors, 224
 - logging RI violations, 599
 - ODBC drivers, 708
 - options for writing synchronization logic, 38
 - overview of event process, 436
 - performance, 219
 - process overview, 24
 - running, 275
 - sample application, 365
 - starting, 18
 - stopping, 613
 - stopping the MobiLink server, 19
 - synchronization logic, 48
 - synchronization techniques, 83, 360
 - transport-layer security, 284
 - Tutorial - Using Adaptive Server Anywhere, 315
 - Tutorial - Using an Oracle database, 347
 - Tutorial - Using MobiLink sample applications, 359
 - Tutorial - Using Sybase Central, 329
- MobiLink .NET API reference, 203
- MobiLink ActiveSync provider installation utility
 - [dbasinst]
 - syntax, 610

- MobiLink certificate generation utility [gencert]
 - syntax, 621
- MobiLink certificate reader utility [readcert]
 - syntax, 620
- MobiLink client database extraction utility
 - [mlxtract]
 - syntax, 614
- MobiLink clients
 - deploying, 714
- MobiLink connections
 - debugging, 282
- MobiLink consolidated databases
 - Adaptive Server Anywhere as, 14
 - Adaptive Server Enterprise as, 14
 - IBM DB2 as, 14
 - Oracle as, 14
 - SQL Server as, 14
- MobiLink data types
 - .NET and SQL, 195
 - Java and SQL, 171
- MobiLink download stream
 - defined, 24
- MobiLink events
 - listed, 433
- MobiLink Java API reference, 183
- MobiLink Monitor
 - about, 232
 - Chart pane, 237
 - description of user interface, 235
 - Details Table pane, 236
 - Options, 240
 - Overview pane, 239
 - Properties, 240
 - restoring defaults, 240
 - saving data, 243
 - specifying watches, 244
 - starting, 233
 - statistical properties, 247
 - using, 235
 - viewing in MS Excel, 243
 - Watch Manager, 244
- MobiLink performance
 - about, 219
 - estimate number of upload rows, 430
 - key factors, 224
- MobiLink security
 - changing passwords, 260
 - choosing a user authentication mechanism, 254
 - custom user authentication, 261, 450
 - new users, 258
 - passwords, 257
 - user authentication, 251
 - user authentication architecture, 255
 - user authentication passwords, 259
- MobiLink server
 - troubleshooting startup, 293
- MobiLink stop utility [dbmlstop]
 - syntax, 613
- MobiLink synchronization
 - Adaptive Server Anywhere clients, 117
 - writing .NET classes, 196
 - writing Java classes, 172
- MobiLink synchronization client
 - automated script generation, 324
 - command line, 324
 - dbmlsync options, 410
 - using extended options, 324
 - using the verbosity option, 324
- MobiLink synchronization logic
 - .NET and SQL data types, 195
 - data types for .NET and SQL, 195
 - data types for Java and SQL, 171
 - Java and SQL data types, 171
- MobiLink synchronization scripts
 - constructing .NET classes, 195
 - constructing Java classes, 171
 - database transactions and .NET classes, 195
 - database transactions and Java classes, 171
 - debugging Java classes, 173
 - preserving database transactions, 171, 195
 - writing .NET classes, 196
 - writing Java classes, 172
- MobiLink synchronization server
 - about, 18
 - deploying, 713
 - multiple instances, 268
 - options, 380
 - starting, 18
 - stopping, 18, 613
 - switches, 380
 - syntax, 380

MobiLink synchronization subscriptions
about, 128

MobiLink system tables
about, 15
creating in consolidated database, 13

MobiLink upload stream
defined, 24
processing, 34

MobiLink user authentication utility [dbmluser]
syntax, 618

MobiLink user creation wizard
using, 125

MobiLink user name, 22
script parameter, 60

MobiLink user names
about, 252
Contact sample, 369

MobiLink users
about, 252
adding to a remote database, 125
authenticating, 251
configuring properties at a remote database, 126
creating in remote databases, 125
dropping from a remote database, 127
passwords, 126

MobiLink utilities
MobiLink ActiveSync provider [dbasinst], 610
MobiLink certificate generator [gencert], 621
MobiLink certificate reader [readcert], 620
MobiLink client database extraction [mlxtract],
614
MobiLink stop utility [dbmlstop], 613
MobiLink user authentication [dbmluser], 618

modify_last_download_timestamp
connection event, 517

modify_next_last_download_timestamp
connection event, 519

modify_user
connection event, 521

monitoring
logging MobiLink RI violations, 599
synchronizations in MobiLink, 231

-mp option
MobiLink [dbmlsync], 425

multiple applications
differentiating MobiLink scripts, 61

N

-n option
MobiLink [dbasinst], 610
MobiLink [dbmlsync], 425

Netscape Web servers
configuring the NSAPI Redirector, 270

new users
MobiLink user authentication, 258

new_row_cursor
cursor event, 523
storing user name, 108

NewMobiLinkPwd synchronization option
about, 414

newsgroups
technical support, xxi

NSAPI Redirector
configuring, 270

NUMERIC data type
MobiLink and Adaptive Server Enterprise, 82

O

-o option
MobiLink [dbmlsrv8], 387
MobiLink [dbmlsync], 426
MobiLink [dbmluser], 618
MobiLink [mlxtract], 614

objects
MobiLink .NET API, 203
MobiLink Java API, 183

ODBC
multiple errors, 76

ODBC data sources
for MobiLink synchronization, 15

ODBC drivers
 for use with MobiLink, 708
 MobiLink character-set translation by, 44

OfflineDirectory synchronization option
 about, 414

old_row_cursor
 cursor event, 525
 storing user name, 108

options
 dbmlsrv8, 380
 dbmlsync, 410
 MobiLink ActiveSync provider [dbasinst], 610
 MobiLink certificate generator [gencert], 621
 MobiLink certificate reader [readcert], 620
 MobiLink client [dbmlsync], 410
 MobiLink client database extraction [mlxtract],
 614
 MobiLink server [dbmlsrv8], 380
 MobiLink stop utility [dbmlstop], 613
 MobiLink user authentication [dbmluser], 618
 priority order for MobiLink extended options,
 129

options for performance tuning
 MobiLink, 139

options for writing synchronization logic, 38

-oq option
 MobiLink [dbmlsrv8], 388

Oracle
 as MobiLink consolidated databases, 14
 conversion of data types in MobiLink
 synchronization, 629
 data types, 629
 MobiLink tutorial, 347
 ODBC configuration, 629
 packages in MobiLink synchronization, 80
 sequences in MobiLink synchronization, 81
 synchronizing LONG data, 629

-os option
 MobiLink [dbmlsrv8], 388
 MobiLink [dbmlsync], 426
 MobiLink [dbmluser], 618

-ot option
 MobiLink [dbmlsrv8], 389
 MobiLink [dbmlsync], 426
 MobiLink [dbmluser], 618

overlaps
 partitioning, 91

P

-p option
 MobiLink [dbmlsync], 427
 MobiLink [dbmluser], 618
 MobiLink [mlxtract], 614

packages
 session-wide information, 80

parameters
 last download timestamp, 60
 MobiLink primary keys, 60
 MobiLink user name, 60
 synchronization scripts, 60
 table name, 60

partitioning
 column-wise, 120
 data among MobiLink remote databases, 155
 defined, 91
 disjoint, 91
 row-wise, 121

partitioning rows
 Contact sample, 370, 372

partitioning tables
 example, 91

parts of the synchronization system, 10

passwords
 changing for MobiLink, 260
 MobiLink dbmluser utility, 618
 MobiLink user authentication, 257, 259
 MobiLink users, 126

-pc option
 MobiLink [dbmluser], 618

performance
 downloads, 223
 MobiLink, 219
 MobiLink upload stream processing, 35

performance tips
 MobiLink, 220

Personal Web Manager
 configuring, 272

- pi option
 - MobiLink [dbmlsync], 427
- pinging
 - MobiLink synchronization server, 427
- port stream parameter
 - HTTP synchronization, 399
 - HTTPS synchronization, 401
 - synchronizing across firewalls, 266
 - TCP/IP synchronization, 397
- pp option
 - MobiLink [dbmlsync], 428
- prepare_for_download
 - connection event, 527
- preparing
 - remote databases for MobiLink, 149
- primary key pools
 - example, 101
 - generating unique values using default global autoincrement, 96
 - synchronization, 100
- primary keys
 - MobiLink and Adaptive Server Enterprise, 82
 - Oracle sequences, 81
 - primary key pools, 101
 - uniqueness in synchronization, 95
- priority order for extended options and connection parameters, 129
- priority synchronization
 - MobiLink performance, 223
- private assemblies
 - implementing in MobiLink, 191
- procedural language
 - role of in MobiLink synchronization, 31
- protocols
 - HTTP synchronization, 398
 - HTTPS synchronization, 400
 - MobiLink synchronization, 10
 - TCP/IP synchronization, 397
- ps option
 - MobiLink [dbmlsrv8], 389
- public key cryptography
 - about, 284
- publication creation wizard
 - column-wise partitioning, 120
 - creating MobiLink publications, 119
 - row-wise partitioning, 121
- publications
 - altering, 122
 - column-wise partitioning, 120
 - creating, 119
 - dropping, 124
 - row-wise partitioning, 121
 - simple, 119
 - using a WHERE clause, 121
- publishing
 - selected columns, 120
 - selected rows, 121
 - tables, 119
 - whole tables, 119
- publishing data, 119
- publishing only some columns in a table, 120
- publishing only some rows in a table, 121
- publishing whole tables, 119

Q

- q option
 - MobiLink [dbmlsrv8], 389
 - MobiLink [dbmlstop], 613
 - MobiLink [dbmluser], 618
 - MobiLink [gencert], 621
 - MobiLink [mlxtract], 614
 - MobiLink client [dbmlsync], 428

R

- r option
 - MobiLink [dbmlsrv8], 389
 - MobiLink [dbmlsync], 429
 - MobiLink [gencert], 621
 - MobiLink [mlxtract], 614
- rd option
 - MobiLink [dbmlsrv8], 390

- readcert utility
 - options, 620
 - syntax, 620
- reading
 - elliptic-curve certificates, 620
 - RSA certificates, 620
- recording errors during synchronization, 75
- Redirector
 - about, 263
 - configuring (all versions), 268
 - configuring for servlet version, 273
 - configuring the ISAPI version for Microsoft Web servers, 272
 - configuring the NSAPI version, 270
 - MobiLink requests, 264
 - specifying the location, 268
- redirector.config
 - configuring, 268
 - location, 268
- REDIRECTOR_HOST directive
 - Redirector, 268
- REDIRECTOR_PORT directive
 - Redirector, 268
- referential integrity
 - during MobiLink synchronization, 35
 - resolving MobiLink RI violations, 599
- registering
 - applications with ActiveSync, 146
- remote databases
 - creating Adaptive Server Anywhere clients, 118
 - deploying, 148
 - extracting, 149
 - SQL scripts, 149
- remote DBA permissions
 - MobiLink synchronization, 138
- remote MobiLink databases
 - schema changes, 116
- removing
 - articles, 122
- replication
 - MobiLink synchronization subscriptions, 128
- report_error
 - connection event, 529
 - syntax, 75
- report_odbc_error
 - connection event, 531
- reporting errors during synchronization, 75
- reqtool
 - how to use, 307
- requests
 - routing, 264
- requirements
 - MobiLink consolidated databases, 12
- resolution
 - MobiLink conflict resolution, 104
- resolve_conflict
 - Contact sample, 375
 - table event, 533
- resolving
 - MobiLink conflicts, 104
- return values
 - .NET synchronization, 196
 - Java synchronization, 172
- reverse proxy
 - defined, 264
- role of digital certificates, 288
- ROLLBACK statement
 - event-hook procedures, 592
- routing requests
 - MobiLink synchronization, 264
- rows
 - partitioning, 91
- RSA certificates
 - generating, 621
 - reading, 620
- rsaserver.crt, 293
- running .NET synchronization logic, 191
- running outside the current session
 - MobiLink, 275

S

- s option
 - MobiLink [dbmlsrv8], 390
 - MobiLink [gencert], 621
- s7 option
 - MobiLink [mlxtract], 614
- sample application
 - CustDB application, 361
 - MobiLink database schema, 361
- sample database
 - CustDB application, 361
 - MobiLink database schema, 361
- sample domain configuration file
 - about, 192
- sample.crt, 293
- samples
 - .NET synchronization logic, 200
 - Contact MobiLink sample, 365
 - Contact MobiLink sample Contact table, 372
 - Contact MobiLink sample Customer table, 370
 - Contact MobiLink sample errors, 375
 - Contact MobiLink sample Product table, 374
 - Contact MobiLink sample SalesRep table, 370
 - Contact MobiLink sample statistics, 375
 - Contact MobiLink sample tables, 367
 - Contact MobiLink sample users, 369
 - Java synchronization logic, 177
 - JavaAuthentication MobiLink sample, 261
 - MobiLink Contact sample, 148
- scheduling
 - MobiLink synchronization, 162
- schema changes
 - remote MobiLink databases, 116
- schemas
 - relating consolidated tables to MobiLink remote tables, 13
- script parameters
 - about, 60
- script types, 55
- script versions
 - adding, 62
 - configuring at remote databases, 126
 - in MobiLink synchronization, 61
- scripts
 - about MobiLink, 30
 - adding and deleting .NET connection scripts, 588
 - adding and deleting .NET table scripts, 589
 - adding and deleting Java connection scripts, 589
 - adding and deleting Java table scripts, 590
 - adding and deleting SQL connection scripts, 586
 - adding and deleting SQL table scripts, 587
 - adding to the consolidated database, 63
 - automating MobiLink synchronization, 50
 - MobiLink events, 433
 - MobiLink synchronization, 18
 - parameters, 60
 - supported DBMS scripting strategies, 80
 - versions, 61
 - writing scripts to download rows, 70
 - writing scripts to upload rows, 66
- scripts and the synchronization process, 53
- ScriptVersion synchronization option
 - about, 414
- secure socket layer
 - with MobiLink synchronization, 284
- secure socket layers
 - obtaining certificates, 307
- security
 - changing passwords, 260
 - custom user authentication, 261
 - MobiLink, 284
 - MobiLink client architecture, 287
 - MobiLink custom user authentication, 446
 - MobiLink synchronization, 138
 - MobiLink user authentication, 251, 254
 - new MobiLink users, 258
 - user authentication passwords, 259
- security stream parameter
 - HTTP synchronization, 399
 - TCP/IP synchronization, 398
- security tips, 291
- SELECT statement
 - FOR UPDATE clause, 68
- self-signed certificates, 294
 - making, 295
 - using, 296
- SendDownloadAck synchronization option
 - about, 414

-
- SendTriggers synchronization option
 - about, 414
 - sequences
 - primary key uniqueness in MobiLink
 - synchronization, 81
 - server authentication
 - MobiLink, 291
 - server stored procedures
 - MobiLink, 586
 - ServerContext
 - MobiLink .NET API, 203
 - MobiLink Java API, 184
 - ServerException
 - MobiLink .NET API, 204
 - MobiLink Java API, 185
 - servers
 - about MobiLink synchronization, 18
 - service dependencies
 - MobiLink, 280
 - services
 - configuring, 277
 - dependencies, 280
 - removing, 277
 - running MobiLink, 275
 - running multiple, 280
 - Windows, 277
 - servlet Redirector
 - configuring, 273
 - servlets
 - installing, 273
 - session-wide variables
 - IBM DB2 in MobiLink synchronization, 81
 - Oracle packages, 80
 - setup
 - MobiLink .NET synchronization logic, 189
 - MobiLink Java synchronization logic, 167
 - shared assemblies
 - implementing in MobiLink, 191
 - ShutdownCallback
 - MobiLink .NET API, 204
 - ShutdownListener
 - MobiLink Java API, 185
 - signing
 - elliptic-curve and RSA certificates, 621
 - SiteScriptName synchronization option
 - about, 414
 - sl dnet option
 - MobiLink [dbmlsrv8], 390
 - user-defined start classes, 197
 - using -MLAutoLoadPath, 191
 - using -MLDomConfigFile, 191
 - sl java option
 - MobiLink [dbmlsrv8], 391
 - user-defined start classes, 174
 - SLEEP directive
 - Redirector, 268
 - snapshot synchronization
 - about, 88
 - Contact sample, 370
 - example, 89
 - sort order
 - characters and MobiLink synchronization, 42
 - sp_hook_dbmlsync_abort stored procedure
 - SQL syntax, 593
 - sp_hook_dbmlsync_begin stored procedure
 - SQL syntax, 594
 - sp_hook_dbmlsync_delay stored procedure
 - SQL syntax, 595
 - sp_hook_dbmlsync_download_begin stored procedure
 - SQL syntax, 596
 - sp_hook_dbmlsync_download_com_error stored procedure
 - SQL syntax, 596
 - sp_hook_dbmlsync_download_end stored procedure
 - SQL syntax, 597
 - sp_hook_dbmlsync_download_fatal_SQL_error stored procedure
 - SQL syntax, 598
 - sp_hook_dbmlsync_download_log_ri_violation stored procedure
 - SQL syntax, 599

- `sp_hook_dbmlsync_download_ri_violation` stored procedure
 - SQL syntax, 600
- `sp_hook_dbmlsync_download_sql_error` stored procedure
 - SQL syntax, 601
- `sp_hook_dbmlsync_download_table_begin` stored procedure
 - SQL syntax, 602
- `sp_hook_dbmlsync_download_table_end` stored procedure
 - SQL syntax, 602
- `sp_hook_dbmlsync_end` stored procedure
 - SQL syntax, 603
- `sp_hook_dbmlsync_logscan_begin` stored procedure
 - SQL syntax, 604
- `sp_hook_dbmlsync_logscan_end` stored procedure
 - SQL syntax, 605
- `sp_hook_dbmlsync_upload_begin` stored procedure
 - SQL syntax, 606
- `sp_hook_dbmlsync_upload_end` stored procedure
 - SQL syntax, 607
- `sp_hook_dbxtract_begin` procedure
 - unique primary keys, 98
 - using, 98
- SQL Anywhere Studio
 - documentation, xiv
- SQL Server
 - as MobiLink consolidated databases, 14
- SQL synchronization logic
 - alternatives, 38
 - MobiLink, 48
- SQL syntax
 - `ml_add_connection_script` stored procedure, 586
 - `ml_add_dnet_connection_script` stored procedure, 588
 - `ml_add_dnet_table_script` stored procedure, 589
 - `ml_add_java_connection_script` stored procedure, 589
 - `ml_add_java_table_script` stored procedure, 590
 - `ml_add_table_script` stored procedure, 587
 - MobiLink server [dbmlsrv8], 380
 - `sp_hook_dbmlsync_abort` stored procedure, 593

- `sp_hook_dbmlsync_begin` stored procedure, 594
- `sp_hook_dbmlsync_delay` stored procedure, 595
- `sp_hook_dbmlsync_download_begin` stored procedure, 596
- `sp_hook_dbmlsync_download_com_error` stored procedure, 596
- `sp_hook_dbmlsync_download_end` stored procedure, 597
- `sp_hook_dbmlsync_download_fatal_SQL_error` stored procedure, 598
- `sp_hook_dbmlsync_download_log_ri_violation` stored procedure, 599
- `sp_hook_dbmlsync_download_ri_violation` stored procedure, 600
- `sp_hook_dbmlsync_download_sql_error` stored procedure, 601
- `sp_hook_dbmlsync_download_table_begin` stored procedure, 602
- `sp_hook_dbmlsync_download_table_end` stored procedure, 602
- `sp_hook_dbmlsync_end` stored procedure, 603
- `sp_hook_dbmlsync_logscan_begin` stored procedure, 604
- `sp_hook_dbmlsync_logscan_end` stored procedure, 605
- `sp_hook_dbmlsync_upload_begin` stored procedure, 606
- `sp_hook_dbmlsync_upload_end` stored procedure, 607

SQLType

- MobiLink .NET API, 208

start classes

- .NET synchronization logic, 197
- DMLStartClasses option for Java, 391
- Java synchronization logic, 174
- MLStartClasses option for .NET, 390

starting

- MobiLink synchronization from UltraLite applications, 21
- MobiLink synchronization server, 18

statement-based scripts

- about, 66
- list, 56

statement-based uploads

- conflict detection, 105
- performance, 221

StaticCursorLongColBuffLen

- Adaptive Server Enterprise, 81

- statistical properties
 - MobiLink, 247
- stop
 - MobiLink synchronization server, 19
- STOP SYNCHRONIZATION DELETE
 - using, 156
- stop utility [dbmlstop]
 - syntax, 613
- stopping
 - MobiLink, 613
 - MobiLink synchronization server, 18, 19
 - upload of deletes using MobiLink, 156
- stored procedures
 - calling in MobiLink synchronization using
 - ODBC syntax, 81
 - ml_add_connection_script SQL syntax, 586
 - ml_add_dnet_connection_script SQL syntax, 588
 - ml_add_dnet_table_script SQL syntax, 589
 - ml_add_java_connection_script SQL syntax, 589
 - ml_add_java_table_script SQL syntax, 590
 - ml_add_table_script SQL syntax, 587
 - MobiLink, 585
 - MobiLink client procedures, 592
 - MobiLink server, 586
 - MobiLink stored procedure source code, 64
 - sp_hook_dbmlsync_abort SQL syntax, 593
 - sp_hook_dbmlsync_begin SQL syntax, 594
 - sp_hook_dbmlsync_delay SQL syntax, 595
 - sp_hook_dbmlsync_download_begin SQL syntax, 596
 - sp_hook_dbmlsync_download_com_error SQL syntax, 596
 - sp_hook_dbmlsync_download_end SQL syntax, 597
 - sp_hook_dbmlsync_download_fatal_SQL_error SQL syntax, 598
 - sp_hook_dbmlsync_download_log_ri_violation, 599
 - sp_hook_dbmlsync_download_ri_violation, 600
 - sp_hook_dbmlsync_download_sql_error SQL syntax, 601
 - sp_hook_dbmlsync_download_table_begin SQL syntax, 602
 - sp_hook_dbmlsync_download_table_end SQL syntax, 602
 - sp_hook_dbmlsync_end SQL syntax, 603
 - sp_hook_dbmlsync_logscan_begin SQL syntax, 604
 - sp_hook_dbmlsync_logscan_end SQL syntax, 605
 - sp_hook_dbmlsync_upload_begin SQL syntax, 606
 - sp_hook_dbmlsync_upload_end SQL syntax, 607
 - using to add or delete synchronization scripts, 64
 - using to download data, 113
- storing user name during conflict resolution, 108
- StreamCompression synchronization option
 - about, 414
- subscribing MobiLink synchronization users, 128
- subscriptions
 - MobiLink synchronization, 128
- support
 - newsgroups, xxi
- supported DBMS scripting strategies, 80
- switches
 - MobiLink ActiveSync provider [dbasinst], 610
 - MobiLink certificate generator [gencert], 621
 - MobiLink certificate reader [readcert], 620
 - MobiLink client [dbmlsync], 410
 - MobiLink client database extraction [mlxtract], 614
 - MobiLink server [dbmlsrv8], 380
 - MobiLink user authentication [dbmluser], 618
- switches:, 613
- Sybase Adaptive Server Enterprise
 - conversion of data types in MobiLink synchronization, 626
- syncasa.sql
 - about, 13
- syncase.sql
 - about, 13
- syncase125.sql
 - about, 13
- syncdb2.sql
 - about, 13
- synchronization
 - about MobiLink, 9, 283
 - architecture of the MobiLink system, 10
 - changing passwords, 260
 - conflict resolution, 104
 - conversion of data types in MobiLink, 625

- conversion to Adaptive Server Enterprise data types in MobiLink, 626
- conversion to IBM DB2 data types in MobiLink, 627
- conversion to Microsoft SQL Server data types in MobiLink, 630
- conversion to Oracle data types in MobiLink, 629
- custom user authentication, 261
- customizing, 592
- deleting rows, 72
- downloading rows, 70
- event hooks, 592
- initiating, 138
- many-to-many relationships, 92
- MobiLink character sets, 42
- MobiLink character-set translation under other platforms, 44
- MobiLink character-set translation under Windows, 42
- MobiLink fault recovery, 32
- MobiLink performance, 35
- MobiLink process overview, 24
- MobiLink scripts, 18
- MobiLink stored procedures, 586
- MobiLink synchronization server authentication, 293
- MobiLink system tables, 15
- MobiLink transactions, 32
- MobiLink tutorial, 315, 329, 347
- MobiLink utilities, 609
- ODBC data sources for MobiLink, 15
- options for writing synchronization logic, 38
- performance tips, 219
- process, 53
- running the MobiLink synchronization server, 275
- snapshot, 89
- techniques, 83
- timestamps in MobiLink, 33
- transactions, 592
- transport-layer security with MobiLink, 284
- using ActiveSync, 143
- writing MobiLink scripts in .NET, 187
- writing MobiLink scripts in Java, 165
- writing scripts, 47
- synchronization basics, 9
- synchronization definitions
 - differences from version 7, 132
 - rewriting for version 8, 136
 - writing, 134
- synchronization errors
 - handling MobiLink, 75
 - troubleshooting, 386
- synchronization event hook sequence, 157
- synchronization events
 - about MobiLink synchronization, 436
 - authenticate_user, 446
 - authenticate_user_hashed, 450
 - begin_connection, 452
 - begin_download, 454, 456
 - begin_download_deletes, 458
 - begin_download_rows, 460
 - begin_synchronization, 462, 464
 - begin_upload, 466, 468
 - begin_upload_deletes, 470
 - begin_upload_rows, 472
 - download_cursor, 474
 - download_delete_cursor, 477
 - download_statistics, 479, 482
 - end_connection, 485
 - end_download, 487, 489
 - end_download_deletes, 491
 - end_download_rows, 493
 - end_synchronization, 495, 497
 - end_upload, 499, 502
 - end_upload_deletes, 504
 - end_upload_rows, 506
 - example_upload_cursor, 508
 - example_upload_delete, 509
 - example_upload_insert, 510
 - example_upload_update, 511
 - handle_error, 512
 - handle_odbc_error, 515
 - MobiLink download, 444
 - MobiLink upload, 438
 - modify_last_download_timestamp, 517
 - modify_next_last_download_timestamp, 519
 - modify_user, 521
 - new_row_cursor, 523
 - old_row_cursor, 525
 - prepare_for_download, 527
 - report_error, 529
 - report_odbc_error, 531
 - resolve_conflict, 533
 - synchronization_statistics, 535, 537
 - time_statistics, 539, 541
 - upload_cursor, 543
 - upload_delete, 545
 - upload_fetch, 547
 - upload_insert, 549

- upload_new_row_insert, 551
- upload_old_row_insert, 553
- upload_statistics, 554, 557
- upload_update, 560
- synchronization logic
 - MobiLink, 48
 - options for writing, 38
- synchronization parameters
 - HTTP synchronization, 397
 - HTTPS synchronization, 397
 - TCP/IP synchronization, 397
- synchronization scripts
 - .NET, 187
 - .NET methods, 196
 - about, 48
 - adding and deleting, 63
 - adding or deleting with stored procedures, 64
 - adding with Sybase Central, 63
 - automatic generation, 50
 - connection scripts, 55, 58
 - cursor scripts, 55
 - DBMS dependencies, 80
 - download_cursor, 71
 - example, 49
 - example generation, 51
 - examples, 77
 - execution during, 53
 - handle_error event, 75
 - implementing for .NET, 189
 - implementing for Java, 167
 - Java, 165
 - Java methods, 172
 - MobiLink events, 433
 - parameters, 60
 - report_error, 75
 - statement-based scripts, 55, 56
 - supported DBMS scripting strategies, 80
 - table scripts, 55, 58
 - testing, 78
 - types, 55
 - versions in MobiLink, 61
 - writing, 47
 - writing scripts to download rows, 70
 - writing scripts to upload rows, 66
- synchronization server
 - about MobiLink, 18
- synchronization subscriptions
 - altering, 129
 - dropping, 130
 - MobiLink, 128
 - options, 129
- synchronization techniques
 - data entry, 110
 - deleting rows, 111
 - failed downloads, 112
 - MobiLink tutorial, 359
 - partitioning, 91
 - primary key pools, 100
 - snapshot-based synchronization, 88
 - stored procedures to download, 113
 - timestamp-based synchronization, 86
 - uploading rows, 66
- synchronization upload stream
 - MobiLink processing, 34
- synchronization users
 - adding to a remote database, 125
 - configuring properties at a remote database, 126
 - creating in remote databases, 125
 - dropping from a remote database, 127
 - multiple, 139
- synchronization_statistics
 - connection event, 535
 - table event, 537
- SynchronizationException
 - MobiLink .NET API, 205
 - MobiLink Java API, 186
- synchronizing
 - databases with MobiLink, 283
- syncmss.sql
 - about, 13
- syncora.sql
 - about, 13
 - using, 351
- syntax
 - MobiLink ActiveSync provider [dbasinst], 610
 - MobiLink certificate generator [gencert], 621
 - MobiLink certificate reader [readcert], 620
 - MobiLink client database extraction [mlxtract], 614
 - MobiLink scripts, 433
 - MobiLink stop utility [dbmlstop], 613
 - MobiLink stored procedures, 586

- MobiLink synchronization utilities, 609
- MobiLink user authentication [dbmluser], 618
- system tables
 - creating in MobiLink consolidated database, 13
 - MobiLink synchronization, 15
- T**
- t option
 - MobiLink [dbmlsrv8], 392
 - MobiLink [dbmlstop], 613
- table, 58
- table scripts
 - about, 58
 - adding .NET scripts, 589
 - adding Java scripts, 590
 - adding SQL scripts, 587
 - adding with Sybase Central, 63
 - defined, 49, 55
 - deleting .NET scripts, 589
 - deleting Java scripts, 590
 - deleting SQL scripts, 587
- TableOrder synchronization option
 - dbmlsync extended option, 414
- tables
 - column-wise partitioning, 120
 - partitioning, 91
 - publishing, 119
 - relating consolidated tables to MobiLink remote tables, 13
 - row-wise partitioning, 121
- TCP/IP
 - dbmlsrv8 -x command line option, 397
 - synchronization parameters, 397
- technical support
 - newsgroups, xxi
- temporarily stopping synchronization of deletes, 156
- testing
 - synchronization scripts, 78
- testing script syntax, 78
- time_statistics
 - connection event, 539
 - table event, 541
- timestamp-based synchronization
 - about, 86
 - Contact sample, 370, 372
 - download_cursor script, 87
 - download_delete_cursor script, 87
- tips
 - synchronization techniques, 85
- Tomcat
 - configuring servlet Redirector for MobiLink, 273
- transaction log
 - location for dbmlsync, 140
- transactions
 - during MobiLink synchronization, 32, 33
 - in MobiLink synchronization scripts, 171, 195
- translation
 - character-set by ODBC drivers, 44
- translation between character sets
 - MobiLink synchronization under other platforms, 44
 - MobiLink synchronization under Windows, 42
- transport-layer security
 - about, 283
 - invoking, 293
 - MobiLink, 284
 - MobiLink client architecture, 287
 - MobiLink security tips, 291
 - obtaining certificates, 307
- troubleshooting
 - handling failed downloads, 112
 - MobiLink, 282
 - MobiLink deployment, 154
 - MobiLink security, 293
 - MobiLink synchronization server log, 19
 - synchronization errors, 386
- tt option
 - MobiLink [dbmlsrv8], 393
- tutorials
 - MobiLink, 329
 - MobiLink sample applications, 359
 - MobiLink with Adaptive Server Anywhere
 - clients, 315
 - MobiLink with Oracle, 347
 - MobiLink with Sybase Central, 329

U

- u option
 - MobiLink [dbasinst], 610
 - MobiLink [dbmlsrv8], 393
 - MobiLink [dbmlsync], 429
 - MobiLink [dbmluser], 618
 - MobiLink [mlxtract], 614
- ud option
 - MobiLink [dbmlsrv8], 393
- ULSynchronize library function, 21
- UltraLite, 77
 - deploying, 715
 - MobiLink clients, 21
- UltraLite applications
 - as MobiLink clients, 21
- UltraLite clients
 - MobiLink, 21
- unique primary keys
 - generating using global autoincrement, 96
 - generating using key pools, 100
 - generating using UUIDs, 95
 - MobiLink installations, 95
- unknown_timeout stream parameter
 - HTTP synchronization, 400
 - HTTPS synchronization, 401
- unkown_timeout stream parameter
 - synchronizing across firewalls, 266
- upgrading applications
 - using multiple MobiLink script versions, 61
- upload cache size
 - MobiLink performance, 221
- upload events
 - about, 66
 - MobiLink synchronization, 438
- upload stream
 - defined, 24
 - events, 66
 - MobiLink transactions, 32
 - processing of MobiLink, 34
- upload_cursor
 - about, 55
 - conflict detection, 106
 - cursor event, 543
- upload_delete
 - about, 56
 - Contact sample, 372, 373
 - table event, 545
- upload_fetch
 - conflict detection, 105
 - Contact sample, 375
 - table event, 547
- upload_insert
 - about, 56
 - Contact sample, 371, 373
 - table event, 549
- upload_new_row_insert
 - about, 56
 - Contact sample, 375
 - table event, 551
- upload_old_row_insert
 - about, 56
 - Contact sample, 375
 - table event, 553
- upload_statistics
 - connection event, 554
 - table event, 557
- upload_update
 - about, 56
 - conflict detection, 105
 - Contact sample, 371, 373, 374
 - table event, 560
- uploading rows
 - MobiLink performance, 223
 - writing scripts, 66
- urc option
 - MobiLink [dbmlsync], 430
- url_suffix stream parameter
 - HTTP synchronization, 400
 - HTTPS synchronization, 402
 - synchronizing across firewalls, 266
- user authentication
 - changing passwords, 260
 - choosing a mechanism in MobiLink, 254
 - custom mechanism, 261

- Java synchronization logic, 261
- MobiLink architecture, 255
- MobiLink passwords, 257
- MobiLink security, 251
- new users, 258
- passwords, 259
- user authentication utility [dbmluser]
 - syntax, 618
- user names
 - MobiLink, 252
 - MobiLink client names, 22
 - MobiLink user authentication utility [dbmluser], 618
- user-defined start classes
 - MobiLink .NET, 197
 - MobiLink Java, 174
- users
 - about MobiLink, 252
- user-specific conflict resolution, 108
- Using a global certificate as a server certificate, 308
- using a globally-signed certificate as an enterprise certificate, 311
- using a self-signed certificate, 296
- using ActiveSync synchronization, 143
- using stored procedures to add or delete synchronization scripts, 64
- using the signed certificates, 303
- utilities
 - MobiLink ActiveSync provider [dbasinst], 610
 - MobiLink certificate generator [gencert], 621
 - MobiLink certificate reader [readcert], 620
 - MobiLink client database extraction [mlxtract], 614
 - MobiLink stop utility [dbmlstop], 613
 - MobiLink synchronization, 609
 - MobiLink user authentication [dbmluser], 618
- UUIDs
 - MobiLink synchronization application, 95

V

- v option
 - MobiLink [dbasinst], 610
 - MobiLink [dbmlsrv8], 393
 - MobiLink [dbmlsync], 430
 - MobiLink [dbmlsync] performance, 222
 - MobiLink [mlxtract], 614
- v+ option
 - MobiLink [dbmlsrv8], 393
 - MobiLink [dbmlsync], 430
- vc option
 - MobiLink [dbmlsrv8], 393
 - MobiLink [dbmlsync], 430
- Verbose
 - Upload about, 414
- Verbose synchronization option
 - about, 414
- VerboseHooks
 - about, 414
- VerboseMin
 - about, 414
- VerboseOptions
 - about, 414
- VerboseRowCounts
 - about, 414
- VerboseValues
 - about, 414
- verbosity option
 - MobiLink, 322
 - MobiLink [dbmlsync], 430
- verifying certificate fields, 310
- verifying fields in certificate chains, 311
- version stream parameter
 - HTTP synchronization, 400
 - HTTPS synchronization, 402
- versions
 - adding script versions, 62
 - of MobiLink synchronization scripts, 61
- vh option
 - MobiLink [dbmlsrv8], 393

Visual Basic
support in MobiLink .NET, 188

- vn option
 - MobiLink [dbmlsrv8], 393
 - MobiLink [dbmlsync], 430
- vo option
 - MobiLink [dbmlsync], 430
- vp option
 - MobiLink [dbmlsync], 430
- vr option
 - MobiLink [dbmlsrv8], 393
 - MobiLink [dbmlsync], 430
- vs option
 - MobiLink [dbmlsrv8], 393
 - MobiLink [dbmlsync], 430
- vt option
 - MobiLink [dbmlsrv8], 393
- vu option
 - MobiLink [dbmlsrv8], 393
 - MobiLink [dbmlsync], 430

W

- w option
 - MobiLink [dbmlsrv8], 394
 - MobiLink [dbmlstop], 613
- wc option
 - MobiLink [dbmlsync], 431

Web servers
 configuring, 264
 configuring for synchronization, 268
 configuring ISAPI Microsoft for synchronization,
 272
 configuring NSAPI Netscape for
 synchronization, 270
 MobiLink clients and, 266

WHERE clause
publications, 121

wizards
 add connection script, 63
 add service, 277
 add synchronized table, 63
 add synchronizing table script, 64, 335

add user, 257
 add version, 62, 335
 article creation, 123
 create database, 331
 MobiLink user creation, 125
 publication creation, 119

worker threads
 MobiLink, 225
 MobiLink performance, 220

writing
 .NET synchronization logic, 187
 Java synchronization logic, 165
 writing download_cursor scripts, 71
 writing scripts to download rows, 70
 writing scripts to handle errors, 75
 writing scripts to upload rows, 66
 writing SQL synchronization scripts, 47
 writing synchronization scripts
 supported DBMS scripting strategies, 80
 writing upload_cursor scripts, 68
 writing upload_delete scripts, 67
 writing upload_fetch scripts, 67
 writing upload_insert scripts, 67
 writing upload_update scripts, 67
 -wu option

- MobiLink [dbmlsrv8], 395

X

-x option

- MobiLink [dbmlsrv8], 396
- MobiLink [dbmlsrv8] -sl dnet, 390
- MobiLink [dbmlsrv8] -sl java, 391
- MobiLink [dbmlsync], 431
- MobiLink [mlxtract], 614

X509 certificates
 generating, 621
 reading, 620

-xf option

- MobiLink [mlxtract], 614

-xh option
MobiLink [mlxtract], 614

-xp option
MobiLink [mlxtract], 614

Xusage.txt
location, 390, 391

Y

-y option
MobiLink [mlxtract], 614

Z

-za option
MobiLink [dbmlsrv8], 402

-zac option
MobiLink [dbmlsrv8], 403

-zd option
MobiLink [dbmlsrv8], 403

-ze option
MobiLink [dbmlsrv8], 403

-zec option
MobiLink [dbmlsrv8], 404

-zp option
MobiLink [dbmlsrv8], 404

-zs option
MobiLink [dbmlsrv8], 405

-zt option
MobiLink [dbmlsrv8], 405

-zu option
MobiLink [dbmlsrv8], 405

-zw option
MobiLink [dbmlsrv8], 405

-zwd option
MobiLink [dbmlsrv8], 406

-zwe option
MobiLink [dbmlsrv8], 407