iAnywhere SOLUTIONS™
A SYBASE COMPANY

Mobile Link
管理ガイド

パート番号：DC03953-01-0902-01
改訂：2005 年 3 月

版権

Copyright © 2005 iAnywhere Solutions, Inc., Sybase, Inc. All rights reserved.

ここに記載されている内容を iAnywhere Solutions, Inc.、Sybase, Inc. またはその関連会社の書面による事前許可を得ずに電子的、機械的、手作業、光学的、またはその他のいかなる手段によっても複製、転載、翻訳することを禁じます。

Sybase、SYBASE のロゴ、Adaptive Server、AnswerBase、Anywhere、EIP、Embedded SQL、Enterprise Connect、Enterprise Portal、GainMomentum、iAnywhere、jConnect MASS DEPLOYMENT、Netimpact、ObjectConnect、ObjectCycle、OmniConnect、Open ClientConnect、Open ServerConnect、PowerBuilder、PowerDynamo、Powersoft、Quickstart Datamart、Replication Agent、Replication Driver、SQL Anywhere、SQL Central、SQL Remote、Support Plus、SWAT、Sybase IQ、Sybase System 11、Sybase WAREHOUSE、SyBooks、XA-Library は米国法人 Sybase, Inc. の登録商標です。Backup Server、Client-Library、jConnect for JDBC、MainframeConnect、Net-Gateway、Net-Library、Open Client、Open Client/Server、S-Designor、SQL Advantage、SQL Debug、SQL Server、SQL Server Manager、Sybase Central、Watcom、Web.SQL、XP Server は米国法人 Sybase, Inc. の商標です。

Certicom、MobileTrust および、SSL Plus は Certicom Corp. の商標です。Security Builder は Certicom Corp. の登録商標です。

ここに記載されている上記以外の社名および製品名は、各社の商標または登録商標の場合があります。

目次

はじめに xi
SQL Anywhere Studio のマニュアル xii
表記の規則 xvi
CustDB サンプル・データベース xix
詳細情報の検索／フィードバックの提供 xx

1 Mobile Link 同期について 3
Mobile Link 同期処理 4

2 同期の基本 9
同期システムの要素 10
統合データベース 12
Mobile Link 同期サーバ 13
Mobile Link クライアント 17
同期処理 18
アップロード専用の同期とダウンロード専用の同期 30
同期論理の作成オプション 31
セキュリティ 36

3 Mobile Link 統合データベース 39
概要 40
統合データベースの設定 42
DBMS 依存の同期スクリプト 43
Adaptive Server Anywhere 統合データベース 46
Sybase Adaptive Server Enterprise 統合データベース 47
Oracle 統合データベース 49
IBM DB2 統合データベース 51
Microsoft SQL Server 統合データベース 54

4 同期の方法 57
概要 58
開発のヒント 59
タイムスタンプベースの同期 60
スナップショットを使った同期 62
リモート・データベース間でローを分割する 65
ユニークなプライマリ・キーの管理 70
競合の解決 80
データ・エントリ 90
削除の処理 91
失敗したダウンロードの処理 92
ストアド・プロシージャ・コールからの結果セットのダウンロード 97
リモート・データベースでのスキーマの変更 100

5 ファイルベースのダウンロード 105
概要 106
ファイルベースのダウンロードの設定 107
検証チェック 112
例 117

6 Mobile Link のパフォーマンス 129
パフォーマンスに関するヒント 130
Mobile Link のパフォーマンスに影響を与える主要な要因 135
Mobile Link のパフォーマンスのモニタ 141

7 Mobile Link モニタ 143
概要 144
Mobile Link モニタの起動 145
Mobile Link モニタの使用 148
モニタのデータの保存 154
統計のカスタマイズ 156
Mobile Link の統計のプロパティ 158

8 リダイレクタによる Web サーバを経由した同期 161
概要 162
リダイレクタの設定 165
Mobile Link クライアントとサーバのリダイレクタ設定 166

リダイレクタのプロパティの設定 (すべてのバージョン共通).....169
Netscape/Sun Web サーバ用の NSAPI リダイレクタ 172
Microsoft Web サーバ用の ISAPI リダイレクタ 177
サーブレット・リダイレクタ 180
Apache リダイレクタ 186
M-Business Anywhere リダイレクタ 191

9 現在のセッション以外での Mobile Link の実行 195
デーモンとしての UNIX 版 Mobile Link サーバの稼働.....196
Windows 版 Mobile Link 同期サーバをサービスとして実行する.....197
Mobile Link サーバ起動時のトラブルシューティング 204

10 Mobile Link トランスポート・レイヤ・セキュリティ 205
概要 206
トランスポート・レイヤ・セキュリティの設定.....208
デジタル証明書の作成 209
トランスポート・レイヤ・セキュリティを使用する Mobile Link 同期サーバの起動 219
トランスポート・レイヤ・セキュリティを使用する Mobile Link クライアントの設定 222

11 Mobile Link 同期サーバのオプション 235
Mobile Link 同期サーバ 236

12 同期スクリプトの作成..... 285
同期スクリプトの概要 286
スクリプトと同期処理 293
スクリプトの種類 295
スクリプトのパラメータ.....297
スクリプト・バージョン.....298
統合データベースでのスクリプトの追加と削除.....301
ローをアップロードするスクリプトの作成.....305
ローをダウンロードするスクリプトの作成.....308
エラーを処理するスクリプトの作成.....316
スクリプト構文のテスト.....318

13 Java による同期スクリプトの作成 321
概要 322
Java 同期論理の設定 323
Java 同期論理の作成 326
Java 同期の例 337
Mobile Link Java API リファレンス 344

14 .NET での同期スクリプトの作成 353
概要 354
.NET 同期論理の設定 355
.NET 同期論理の作成 360
.NET の同期の方法 371
共有アセンブリのロード 373
.NET 同期のサンプル 377
Mobile Link .NET API リファレンス 381

15 同期イベント 401
Mobile Link イベントの概要 402
authenticate_parameters 接続イベント 417
authenticate_user 接続イベント 420
authenticate_user_hashed 接続イベント 425
begin_connection 接続イベント 429
begin_connection_autocommit 接続イベント 431
begin_download 接続イベント 432
begin_download テーブル・イベント 434
begin_download_deletes テーブル・イベント 437
begin_download_rows テーブル・イベント 440
begin_publication 接続イベント 443
begin_synchronization 接続イベント 447
begin_synchronization テーブル・イベント 449
begin_upload 接続イベント 452
begin_upload テーブル・イベント 455
begin_upload_deletes テーブル・イベント 458
begin_upload_rows テーブル・イベント 461
download_cursor テーブル・イベント 464
download_delete_cursor テーブル・イベント 468
download_statistics 接続イベント 472
download_statistics テーブル・イベント 476

end_connection 接続イベント 480
end_download 接続イベント.....482
end_download テーブル・イベント.....485
end_download_deletes テーブル・イベント 488
end_download_rows テーブル・イベント 491
end_publication 接続イベント.....494
end_synchronization 接続イベント.....497
end_synchronization テーブル・イベント 499
end_upload 接続イベント 502
end_upload テーブル・イベント 504
end_upload_deletes テーブル・イベント 508
end_upload_rows テーブル・イベント 511
example_upload_cursor テーブル・イベント 514
example_upload_delete テーブル・イベント.....515
example_upload_insert テーブル・イベント 516
example_upload_update テーブル・イベント.....517
handle_error 接続イベント 518
handle_odbc_error 接続イベント.....522
modify_error_message 接続イベント.....525
modify_last_download_timestamp 接続イベント 527
modify_next_last_download_timestamp 接続イベント 530
modify_user 接続イベント.....533
new_row_cursor テーブル・イベント (旧式) 536
old_row_cursor テーブル・イベント (旧式).....539
prepare_for_download 接続イベント 542
report_error 接続イベント 544
report_odbc_error 接続イベント.....547
resolve_conflict テーブル・イベント.....550
synchronization_statistics 接続イベント.....553
synchronization_statistics テーブル・イベント.....556
time_statistics 接続イベント 559
time_statistics テーブル・イベント 562
upload_cursor テーブル・イベント (旧式) 565
upload_delete テーブル・イベント.....568
upload_fetch テーブル・イベント 570
upload_insert テーブル・イベント 572
upload_new_row_insert テーブル・イベント 575
upload_old_row_insert テーブル・イベント.....577

upload_statistics 接続イベント....579
upload_statistics テーブル・イベント....583
upload_update テーブル・イベント....587

16 ストアド・プロシージャ 593
スクリプトを追加または削除するためのストアド・プロシージャ....594
プロパティを追加または削除するためのストアド・プロシージャ....602

17 ユーティリティ 605
Mobile Link 停止ユーティリティ....606
Mobile Link ユーザ認証ユーティリティ....608
証明書読み込みユーティリティ....611
証明書生成ユーティリティ....612

18 Mobile Link システム・テーブル 619
概要....620
ml_connection_script....621
ml_device....622
ml_device_address....624
ml_listening....626
ml_property....628
ml_qa_delivery....630
ml_qa_delivery_client....631
ml_qa_global_props....632
ml_qa_global_props_client....633
ml_qa_notifications....634
ml_qa_repository....635
ml_qa_repository_client....636
ml_qa_repository_content_client....638
ml_qa_repository_props....639
ml_qa_repository_props_client....640
ml_qa_repository_staging....641
ml_qa_status_staging....643
ml_script....644
ml_script_version....645
ml_scripts_modified....646
ml_subscription....647
ml_table....649

ml_table_script....650
ml_user....651

19 データ型の変換....653
Adaptive Server Enterprise データのマッピング....654
IBM DB2 データのマッピング....656
Oracle データのマッピング....658
Microsoft SQL Server データのマッピング....660

20 文字セットの考慮事項....663
文字セットの考慮事項....664

21 iAnywhere Solutions ODBC ドライバ....669
Mobile Link でサポートされる ODBC ドライバ....670

22 Mobile Link アプリケーションの配備....673
配備の概要....674
Mobile Link サーバの配備....675
Adaptive Server Anywhere Mobile Link クライアントの配備....682
Ultra Light Mobile Link クライアントの配備....685
QAnywhere アプリケーションの配備....686

索引....689

はじめに

このマニュアルの内容

このマニュアルでは、セッションベースのリレーショナル・データベース同期システムである Mobile Link について説明します。Mobile Link テクノロジは、双方向レプリケーションを可能にし、モバイル・コンピューティング環境に非常に適しています。

対象読者

このマニュアルは、使用している情報システムに同期またはレプリケーションを追加したいと考えている Adaptive Server Anywhere ユーザと他のリレーショナル・データベース・システムのユーザを対象としています。

始める前に

Mobile Link と他の同期／レプリケーション・テクノロジの比較については、『SQL Anywhere Studio の紹介』> 「レプリケーション・テクノロジの概要」を参照してください。

SQL Anywhere Studio のマニュアル

このマニュアルは、SQL Anywhere のマニュアル・セットの一部です。この項では、マニュアル・セットに含まれる各マニュアルと使用法について説明します。

SQL Anywhere Studio のマニュアル

SQL Anywhere Studio のマニュアルは、各マニュアルを 1 つの大きなヘルプ・ファイルにまとめたオンライン形式、マニュアル別の PDF ファイル、および有料の製本版マニュアルで提供されます。SQL Anywhere Studio のマニュアルは、次の分冊マニュアルで構成されています。

- **『SQL Anywhere Studio の紹介』** このマニュアルでは、SQL Anywhere Studio のデータベース管理と同期テクノロジの概要について説明します。また、SQL Anywhere Studio を構成する各部分について説明するチュートリアルも含まれています。
- **『SQL Anywhere Studio 新機能ガイド』** このマニュアルは、SQL Anywhere Studio のこれまでのリリースのユーザを対象としています。ここでは、製品の今回のリリースと以前のリリースで導入された新機能をリストし、アップグレード手順を説明しています。
- **『Adaptive Server Anywhere データベース管理ガイド』** このマニュアルでは、データベースおよびデータベース・サーバの実行、管理、設定について説明しています。
- **『Adaptive Server Anywhere SQL ユーザーズ・ガイド』** このマニュアルでは、データベースの設計と作成の方法、データのインポート・エクスポート・変更の方法、データの検索方法、ストアド・プロシージャとトリガの構築方法について説明します。
- **『Adaptive Server Anywhere SQL リファレンス・マニュアル』** このマニュアルは、Adaptive Server Anywhere で使用する SQL 言語の完全なリファレンスです。また、Adaptive Server Anywhere のシステム・テーブルとシステム・プロシージャについても説明しています。
- **『Adaptive Server Anywhere プログラミング・ガイド』** このマニュアルでは、C、C++、Java プログラミング言語を使用してデータベース・アプリケーションを構築、配備する方法につい

て説明します。Visual Basic や PowerBuilder などのツールのユーザは、それらのツールのプログラミング・インタフェースを使用できます。また、Adaptive Server Anywhere ADO.NET データ・プロバイダについても説明します。

- **『Adaptive Server Anywhere SNMP Extension Agent ユーザーズ・ガイド』** このマニュアルでは、Adaptive Server Anywhere SNMP Extension Agent を SNMP 管理アプリケーションとともに使用できるように設定して、Adaptive Server Anywhere データベースを管理できるようにする方法を説明します。

- **『Adaptive Server Anywhere エラー・メッセージ』** このマニュアルでは、Adaptive Server Anywhere エラー・メッセージの完全なリストを、その診断情報とともに説明します。

- **『SQL Anywhere Studio セキュリティ・ガイド』** このマニュアルでは、Adaptive Server Anywhere データベースのセキュリティ機能について説明します。Adaptive Server Anywhere 7.0 は、米国政府から TCSEC (Trusted Computer System Evaluation Criteria) の C2 セキュリティ評価を授与されています。このマニュアルには、Adaptive Server Anywhere の現在のバージョンを、C2 基準を満たした環境と同等の方法で実行することを望んでいるユーザにとって役に立つ情報が含まれています。

- **『Mobile Link 管理ガイド』** このマニュアルでは、モバイル・コンピューティング用の Mobile Link データ同期システムについてあらゆる角度から説明します。このシステムによって、Oracle、Sybase、Microsoft、IBM の単一データベースと、Adaptive Server Anywhere や Ultra Light の複数データベースの間でのデータ共有が可能になります。

- **『Mobile Link クライアント』** このマニュアルでは、Adaptive Server Anywhere リモート・データベースと Ultra Light リモート・データベースの設定を行い、これらを同期させる方法について説明します。

- **『Mobile Link サーバ起動同期ユーザーズ・ガイド』** このマニュアルでは、Mobile Link のサーバによって開始される同期について説明します。サーバによって開始される同期とは、統合データベースから同期の開始を可能にする Mobile Link の機能です。

- **『Mobile Link チュートリアル』** このマニュアルでは、Mobile Link アプリケーションの設定と実行の手順を示す各種のチュートリアルについて説明します。

- **『QAnywhere ユーザーズ・ガイド』** このマニュアルでは、Mobile Link QAnywhere について説明します。Mobile Link QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアントだけでなく、モバイル・クライアントや無線クライアント用のメッセージング・アプリケーションの開発と展開を可能にするメッセージング・プラットフォームです。

- **『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』** このマニュアルでは、Mobile Link 同期サーバから、または Adaptive Server Anywhere リモート・データ・アクセスによって、Adaptive Server Anywhere 以外の統合データベースにアクセスするための ODBC ドライバの設定方法について説明します。

- **『SQL Remote ユーザーズ・ガイド』** このマニュアルでは、モバイル・コンピューティング用の SQL Remote データ・レプリケーション・システムについて、あらゆる角度から説明します。このシステムによって、Adaptive Server Anywhere または Adaptive Server Enterprise の単一データベースと Adaptive Server Anywhere の複数データベースの間で、電子メールやファイル転送などの間接的リンクを使用したデータ共有が可能になります。

- **『SQL Anywhere Studio ヘルプ』** このマニュアルには、Sybase Central や Interactive SQL、その他のグラフィカル・ツールに関するコンテキスト別のヘルプが含まれています。これは、製本版マニュアル・セットには含まれていません。

- **『Ultra Light データベース・ユーザーズ・ガイド』** このマニュアルは、Ultra Light 開発者を対象としています。ここでは、Ultra Light データベース・システムの概要について説明します。また、すべての Ultra Light プログラミング・インタフェースに共通する情報を提供します。

- **Ultra Light のインタフェースに関するマニュアル** 各 Ultra Light プログラミング・インタフェースには、それぞれに対応するマニュアルを用意しています。これらのインタフェースは、RAD (

ラピッド・アプリケーション開発) 用の Ultra Light コンポーネントとして提供されているものと、C、C++、Java 開発用の静的インタフェースとして提供されているものがあります。

このマニュアル・セットの他に、PowerDesigner と InfoMaker には、独自のオンライン・マニュアル (英語版) がそれぞれ用意されています。

マニュアルの形式

SQL Anywhere Studio のマニュアルは、次の形式で提供されています。

- **オンライン・マニュアル**　オンライン・マニュアルには、SQL Anywhere Studio の完全なマニュアルがあり、SQL Anywhere ツールに関する印刷マニュアルとコンテキスト別のヘルプの両方が含まれています。オンライン・マニュアルは、製品のメンテナンス・リリースごとに更新されます。これは、最新の情報を含む最も完全なマニュアルです。

  Windows オペレーティング・システムでオンライン・マニュアルにアクセスするには、[スタート] － [プログラム] － [SQL Anywhere 9] － [オンライン・マニュアル] を選択します。オンライン・マニュアルをナビゲートするには、左ウィンドウ枠で HTML ヘルプの目次、索引、検索機能を使用し、右ウィンドウ枠でリンク情報とメニューを使用します。

  UNIX オペレーティング・システムでオンライン・マニュアルにアクセスするには、SQL Anywhere のインストール・ディレクトリに保存されている HTML マニュアルを参照してください。

- **PDF 版マニュアル**　SQL Anywhere の各マニュアルは、Adobe Acrobat Reader で表示できる PDF ファイルで提供されています。

  PDF 版マニュアルは、オンライン・マニュアルまたは Windows の [スタート] メニューから利用できます。

- **製本版マニュアル**　製本版マニュアルをご希望の方は、ご購入いただいた販売代理店または弊社営業担当までご連絡ください。

表記の規則

この項では、このマニュアルで使用されている書体およびグラフィック表現の規則について説明します。

SQL 構文の表記規則

SQL 構文の表記には、次の規則が適用されます。

- **キーワード**　SQL キーワードはすべて次の例に示す ALTER TABLE のように大文字で表記します。

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

- **プレースホルダ**　適切な識別子または式で置き換えられる項目は、次の例に示す *owner* や *table-name* のように表記します。

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

- **繰り返し項目**　繰り返し項目のリストは、次の例に示す *column-constraint* のように、リストの要素の後ろに省略記号 (ピリオド 3 つ …) を付けて表します。

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

  複数の要素を指定できます。複数の要素を指定する場合は、各要素間をカンマで区切る必要があります。

- **オプション部分**　文のオプション部分は角カッコで囲みます。

  RELEASE SAVEPOINT [*savepoint-name*]

  この例では、角カッコで囲まれた *savepoint-name* がオプション部分です。角カッコは入力しないでください。

- **オプション**　項目リストから 1 つだけ選択するか、何も選択しなくてもよい場合は、項目間を縦線で区切り、リスト全体を角カッコで囲みます。

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

  この例では、ASC と DESC のどちらか 1 つを選択しても、どちらも選択しなくてもかまいません。角カッコは入力しないでください。

- **選択肢**　オプションの中の 1 つを必ず選択しなければならない場合は、選択肢を中カッコで囲み、縦棒で区切ります。

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

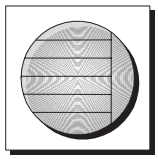
  QUOTES オプションを使用する場合は、ON または OFF のどちらかを選択する必要があります。角カッコと中カッコは入力しないでください。

グラフィック・アイコン

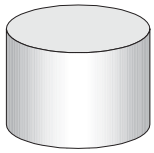
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

- Sybase Adaptive Server Anywhere などのデータベース・サーバ



- データベース。高度な図では、データベースとデータベースを管理するデータ・サーバの両方をこのアイコンで表します。



- レプリケーションまたは同期のミドルウェア。ソフトウェアのこれらの部分は、データベース間のデータ共有を支援します。たとえば、Mobile Link 同期サーバ、SQL Remote Message Agent などがあげられます。

- プログラミング・インタフェース

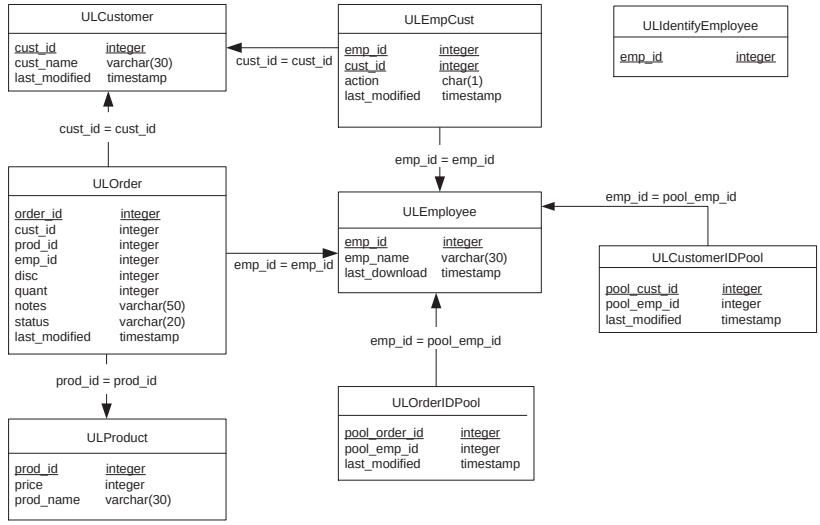
CustDB サンプル・データベース

Mobile Link と Ultra Light のマニュアルでは、多くの例で Ultra Light のサンプル・データベースが使用されています。

Ultra Light サンプル・データベースのリファレンス・データベースは、*custdb.db* という名前のファイルに保存され、SQL Anywhere ディレクトリのサブディレクトリ *Samples¥UltraLite¥CustDB* に置かれています。全面的にこのデータベースを使用して構築したアプリケーションも提供されています。

サンプル・データベースは、あるハードウェア販売会社の販売管理データベースです。データベースには、この販売会社の顧客、製品、営業戦力に関する情報が入っています。

次の図は、CustDB データベース内のテーブルと、各テーブル間の関係を示しています。



詳細情報の検索／フィードバックの提供

詳細情報の検索

詳しい情報やリソース (コード交換など) については、iAnywhere Developer Network (http://www.ianywhere.com/developer/) を参照してください。

ご質問がある場合や支援が必要な場合は、次に示す iAnywhere Solutions ニュースグループのいずれかにメッセージをお寄せください。

ニュースグループにメッセージをお送りいただく際には、ご使用の SQL Anywhere Studio バージョンのビルド番号を明記し、現在発生している問題について詳しくお知らせくださいますようお願いいたします。バージョン情報は、コマンド・プロンプトで **dbeng9 -v** と入力して確認できます。

ニュースグループは、ニュース・サーバ *forums.sybase.com* にあります (ニュースグループにおけるサービスは英語でのみの提供となります)。以下のニュースグループがあります。

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

ニュースグループに関するお断り
iAnywhere Solutions は、ニュースグループ上に解決策、情報、または意見を提供する義務を負うものではありません。また、システム・オペレータ以外のスタッフにこのサービスを監視させて、操作状況や可用性を保証する義務もありません。

iAnywhere Solutions のテクニカル・アドバイザとその他のスタッフは、時間のある場合にかぎりニュースグループでの支援を行います。こうした支援は基本的にボランティアで行われるため、解決策や情報を定期的に提供できるとはかぎりません。支援できるかどうかは、スタッフの仕事量に左右されます。

フィードバック

このマニュアルに関するご意見、ご提案、フィードバックをお寄せください。

マニュアルに関するご意見、ご提案は、SQL Anywhere ドキュメンテーション・チームの iasdoc@ianywhere.com 宛てに電子メールでお寄せください。このアドレスに送信された電子メールに返信はいたしませんが、お寄せいただいたご意見、ご提案は必ず読ませていただきます。

マニュアルまたはソフトウェアについてのフィードバックは、上記のニュースグループを通してお寄せいただいてもかまいません。

第 1 部　Mobile Link テクノロジの使用

第 1 部では、Mobile Link 同期テクノロジの概要と、Mobile Link 同期テクノロジを使用して 2 つ以上のデータベース間でデータをレプリケートする方法について説明します。

第 1 章

Mobile Link 同期について

この章の内容

この章では、Mobile Link 同期テクノロジについて説明します。ここでは、Mobile Link の目的と特性を説明します。

Mobile Link クライアントについては、『Mobile Link クライアント』>「Mobile Link クライアントの紹介」を参照してください。

Mobile Link テクノロジの詳細については、「同期の基本」9 ページを参照してください。

Mobile Link 同期処理

Mobile Link はセッションベースの同期システムです。したがって、統合データベースと呼ばれるメイン・データベースと多数のリモート・データベース間で双方向の同期を行うことができます。統合データベース (ODBC 準拠の任意のデータベース) には、すべてのデータのマスタ・コピーが保持されます。リモート・データベースには、Adaptive Server Anywhere データベースまたは Ultra Light データベースのどちらかを使用できます。

通常は、Mobile Link リモート・サイトで Mobile Link 同期サーバへの接続を開くと、同期が開始されます。同期中に、リモート・サイト側の Mobile Link クライアントは、前回の同期後にリモート・データベースに対して行われたデータベースの変更をアップロードします。Mobile Link 同期サーバは、このデータを受信すると、統合データベースを更新し、統合データベースの変更内容をリモート・データベースにダウンロードします。

Mobile Link の機能

Mobile Link 同期には高い適応性と柔軟性があります。以下に、主な機能の一部を示します。

- **データ調整**　Mobile Link によって、データの特定部分を同期対象として選択できます。また、Mobile Link 同期では、異なるデータベースで行われた変更内容の競合を解決できます。同期処理は、SQL、Java または .NET アプリケーションとして作成できる同期論理によって制御されます。この論理の各部は、「**スクリプト**」と呼ばれます。スクリプトを使用すると、たとえば、アップロードされたデータを統合データベースに適用する方法の指定やダウンロード内容を取得するデータベースの指定を行ったり、統合データベースとリモート・データベースとで異なるスキーマや名前を処理したりできます。

- **自動化**　Mobile Link には、多数の自動化機能があります。Mobile Link 同期サーバには、スナップショット同期に適したスクリプトや、サンプル同期スクリプトを生成するように指示で

きます。また、認証するユーザも自動的に追加できます。サーバが開始した同期では、データの更新をリモート・データベースに送信できます。

- **モニタとレポート**　Mobile Link には、同期をモニタする 2 つのメカニズムが用意されています。1 つは Mobile Link モニタで、もう 1 つは統計スクリプトです。スクリプト、スキーマの内容、ローカウント値、スクリプト名、変換されたスクリプトの内容、ローの値をモニタできます。

- **パフォーマンス・チューニング**　Mobile Link のパフォーマンスをチューニングするためのメカニズムが多数用意されています。たとえば、競合レベル、アップロードのキャッシュ・サイズ、データベース接続数、ワーカ・スレッド数、ロギングの冗長性、または BLOB のキャッシュ・サイズを調整できます。

- **双方向の同期**　すべてのロケーションでデータベースを変更できます。

- **アップロード専用の同期またはダウンロード専用の同期**　アップロード専用の同期またはダウンロード専用の同期を選択できます。

- **ファイルベースのダウンロード**　ダウンロード内容はファイルとして配布することが可能であり、同期の変更をオフラインで配布できます。このため、一度ファイルを作成すれば、広く配布することが可能です。

- **サーバが開始した同期**　Mobile Link 同期は、統合データベースから開始できます。これは、データの更新をリモート・データベースに送信できるほか、リモート・データベースが統合データベースにデータをアップロードできることを意味しています。

- **通信ストリームの選択**　TCP/IP、HTTP、または HTTPS を使用して同期を実行できます。Palm デバイスは HotSync を使用して同期できます。Windows CE デバイスは、ActiveSync を使用して同期できます。

- **リモートでの開始**　リモート・データベースと統合データベース間の同期を、リモート・データベース側から開始できます。

- **セッションベース**　すべての変更内容は、単一のトランザクションでアップロードし、単一のトランザクションでダウンロードできます。同期が成功するたびに、統合データベースとリモート・データベースが一貫した状態になります。

- **トランザクション単位の整合性**　トランザクション全体が同期されるか、トランザクション全体がまったく同期されないかのどちらかになります。これにより、各データベースでトランザクション単位の整合性が確保されます。

- **データの一貫性**　Mobile Link は、*緩やかな一貫性*方式を使用しています。つまり、変更内容はすべて一貫性が保たれるように各サイトで同期されますが、時間的にはわずかなズレがあるため、ある瞬間だけを見ると、各サイトに存在するデータのコピーが異なる場合もあります。

- **多様なハードウェアとソフトウェアのプラットフォーム**　広く使用されている Adaptive Server Anywhere、Adaptive Server Enterprise、Oracle、Microsoft SQL Server、または IBM DB2 などの多様なデータベース管理システムを、Mobile Link 統合データベースとして使用できます。リモート・データベースは、Adaptive Server Anywhere データベースでも Ultra Light データベースでもかまいません。Mobile Link 同期サーバは、Windows または UNIX プラットフォーム上で動作します。Adaptive Server Anywhere は、Windows、Windows CE、または UNIX マシン上で動作します。Ultra Light は、Palm、Windows CE、または Java ベースのデバイス上で動作します。

- **柔軟性**　Mobile Link 同期サーバは、SQL、Java または .NET スクリプトを使用して、データのアップロードとダウンロードを制御します。スクリプトは、各同期中にイベント・モデルに従って実行されます。イベントベースのスクリプト機能により、競合解決、エラー・レポート、ユーザ認証などの機能を含め、同期処理の設計がきわめて柔軟になります。

- **スケーラビリティとパフォーマンス**　Mobile Link 同期はスケーラビリティに優れています。単一のサーバが数千の同期を同時に処理したり、負荷分散を使用して複数の Mobile Link サーバを同時に稼働したりできます。Mobile Link 同期サーバはマルチス

レッド化されており、統合データベースで接続プールを使用します。Mobile Link には、広範囲のモニタ機能とレポート機能が用意されています。

- **起動が簡単**　シンプルな Mobile Link インストール環境をすばやく構成できます。より複雑な微調整は、運用作業全体で段階を追って実行できます。

第 2 章

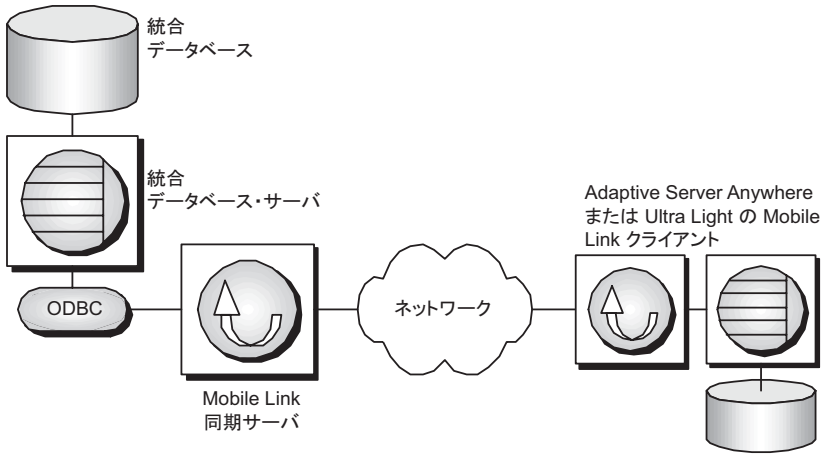
同期の基本

この章の内容

この章では、Mobile Link テクノロジの基本コンポーネントと、同期システムの設定方法について説明します。

同期システムの要素

次の図は、同期システムの主な要素を示しています。



- **統合データベース**　このデータベースには、同期システムのすべての情報の中核となるコピーが収められています。

  詳細については、「統合データベース」12 ページを参照してください。

- **統合データベース・サーバ**　統合データベースを管理するサーバまたは DBMS です。統合データベース・サーバには、Adaptive Server Anywhere または Adaptive Server Enterprise などの Sybase 製品か、サポートされている他社製造のシステムが使用されます。

  詳細については、「Mobile Link 統合データベース」39 ページを参照してください。

- **ODBC 接続**　Mobile Link 同期サーバと統合データベース間のすべてのやりとりは ODBC 接続を介して行われます。同期サーバは ODBC によって各種の統合データベース・システムを利用できます。

詳細については、『Mobile Link 管理ガイド』> 「iAnywhere Solutions ODBC ドライバ」を参照してください。

- **Mobile Link 同期サーバ**　同期処理を管理し、すべての Mobile Link クライアントと統合データベース・サーバ間のインタフェースを提供します。

  詳細については、「Mobile Link 同期サーバ」13 ページを参照してください。

- **ネットワーク**　Mobile Link 同期サーバ (dbmlsrv9) と Mobile Link クライアント (dbmlsync または Ultra Light) 間の接続では複数のプロトコルが使用できます。

  dbmlsync への接続の詳細については、『Mobile Link クライアント』> 「-x オプション」を参照してください。Ultra Light への接続の詳細については、『Mobile Link クライアント』> 「Ultra Light 同期クライアントのネットワーク・プロトコルのオプション」を参照してください。

- **Mobile Link クライアント**　クライアントは、Palm Pilot や PocketPC などのハンドヘルド・デバイス、サーバまたはデスクトップ・コンピュータ、または携帯電話や自動販売機などの組み込みデバイスにインストールできます。Ultra Light と Adaptive Server Anywhere データベースという 2 種類のクライアントがあります。単一の Mobile Link インストール環境では、これらのうちのどちらか 1 つまたは両方を使用できます。

  詳細については、「Mobile Link クライアント」17 ページを参照してください。

統合データベース

アプリケーションは、中央の統合データベースと同期します。このデータベースは同期システム内にある情報のマスタ・レポジトリです。

統合データベースとリモート・データベース間の関係を構築するには、多くの方法があります。そのうちの 2 つの例を次に示します。

リモート・データベースのスキーマを統合データベースのスキーマのサブセットにすることができます。たとえば、emp というテーブルはさまざまなリモート・サイトで繰り返し使用され、統合データベースは expense というテーブルにある emp.salary からのカラム・データを使用します。この場合、統合データベースとリモート・データベースのスキーマは、データを共有していても異なります。

また、統合データベースのスキーマとリモート・データベースのスキーマを構造上、並列にすることもできます。この場合、統合データベースのスキーマはリモート・データベースの参照になります。統合データベース内では、ユーザが各リモート・テーブルに対応するテーブルをすでに持っている場合があります。この場合、統合データベースとリモート・データベースのスキーマは事実上同じであり、リモート・データベースのデータは統合データベースのデータのサブセットにすぎません。

リモート・データベースの各テーブルに対して「**同期スクリプト**」を記述して、これらのスクリプトを統合データベースで保存してください。リモート・データベースと統合データベース間でデータを移動させるときには、統合データベースの中央のロケーションにあるこれらのスクリプトから同期サーバに指示が出ます。特定のリモート・テーブルの 1 つのスクリプトは、同期サーバに対して、統合データベースのリモート・テーブルからアップロードされたデータの格納場所を通知します。別のスクリプトは、同期サーバに対して、同じリモート・テーブルにどのデータをダウンロードするかを通知します。

統合データベースの詳細については、「Mobile Link 統合データベース」39 ページを参照してください。

Mobile Link 同期サーバ

すべての Mobile Link クライアントは、Mobile Link 同期サーバを介して同期します。データベース・サーバには、直接接続できません。Mobile Link 同期サーバを起動してから、Mobile Link クライアントに同期を要求します。

Mobile Link 同期サーバの実行

Mobile Link 同期サーバは、統合データベース・サーバとの接続を ODBC を介して開きます。その後、リモート・アプリケーションからの接続を受け入れ、同期処理を制御します。

❖ **Mobile Link 同期サーバを起動するには、次の手順に従います。**

- dbmlsrv9 を実行します。-c オプションを使用して統合データベースの ODBC 接続パラメータを指定します。

接続パラメータの詳細については、「-c オプション」245 ページを参照してください。

接続パラメータの指定は必須です。他のオプションは、必要に応じて使用します。これらのオプションを使用すると、サーバの動作方法を指定できます。たとえば、ワーカ・スレッドの最大数、キャッシュ・サイズ、ログ・オプションを指定することが可能です。

dbmlsrv9 オプションの詳細については、「Mobile Link 同期サーバのオプション」235 ページを参照してください。

*注意：*dbmlsrv9 のオプションを指定すると、Mobile Link 同期サーバの動作方法を指定できます。サーバの動作を制御するには、同期イベントで呼び出されるスクリプトを定義します。

詳細については、「Mobile Link イベント」20 ページを参照してください。

例

次のコマンドは、Mobile Link 同期サーバを起動し、ODBC データ・ソース *UltraLite 9.0 Sample* を統合データベースとして特定します。コマンド全体を 1 行に入力してください。

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

この例では、-zs オプションでサーバ名を指定しています。−o オプションは、ログ・ファイル名を *mlsrv.log* と指定します。−vcr オプションが指定されているので、*mlsrv.log* の内容は冗長です。−x オプションで、Mobile Link クライアントが TCP/IP を介する接続を許可されるように指定しています。

Mobile Link 同期サーバを Windows サービスまたは UNIX デーモンとして起動することもできます。詳細については、「現在のセッション以外での Mobile Link の実行」195 ページを参照してください。

Mobile Link 同期サーバの停止

Mobile Link 同期サーバは、サーバを起動したコンピュータから停止できます。次の方法で、Mobile Link サーバを停止できます。

- Mobile Link サーバのウィンドウで [シャットダウン] をクリックする
- システム・トレイのコンテキスト・メニューで [終了] をクリックする
- dbmlstop ユーティリティを使用する

  詳細については、『Mobile Link 管理ガイド』> 「Mobile Link 停止ユーティリティ」を参照してください。

Mobile Link 同期サーバの動作のロギング

サーバの動作をロギングすると、開発プロセスとトラブルシューティングのときに特に役立ちます。パフォーマンスが低下するため、運用環境の通常の操作には冗長出力を使用しないでください。

ファイルへのロギング結果の出力

ロギング結果は、[Mobile Link 同期サーバ] ウィンドウに送信されます。また、-o オプションを使用して結果をログ・ファイルにも送信できます。次のコマンドでは、結果を *mlsrv.log* という名前のログ・ファイルに送ります。

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

詳細については、「-o オプション」254 ページを参照してください。

ログ・ファイルのサイズの制御

ログファイルのサイズを制御したり、ファイルが最大サイズに達したときの処理を指定したりできます。

- -on オプションを付けてサイズを指定すると、そのサイズに達したときに拡張子 .old を付けてログ・ファイルの名前が変更され、元の名前の新しいファイルが使用されます。
- -os オプションを付けてサイズを指定すると、そのサイズに達したときに、日付と連番に基づいた新しい名前の新しいログ・ファイルが使用されます。
- -ot オプションでは、メッセージが送信される前に、ログ・ファイルの内容が削除されます。

詳細については、次の項目を参照してください。

- 『Mobile Link 管理ガイド』> 「-on オプション」
- 『Mobile Link 管理ガイド』> 「-os オプション」
- 『Mobile Link 管理ガイド』> 「-ot オプション」

ロギング結果の出力容量の制御

ロギングされる結果の出力容量を制御するには、-v オプションを使用します。

詳細については、「-v オプション」265 ページを参照してください。

レポートされるエラーの制御

レポートされる警告メッセージを制御することもできます。

詳細については、次の項目を参照してください。

- 『Mobile Link 管理ガイド』> 「-zw オプション」
- 『Mobile Link 管理ガイド』> 「-zwd オプション」
- 『Mobile Link 管理ガイド』> 「-zwe オプション」

Mobile Link クライアント

各リモート・データベースとそのアプリケーションを、「**Mobile Link クライアント**」といいます。次の 2 種類の Mobile Link クライアントがサポートされています。

- Adaptive Server Anywhere
- Ultra Light

詳細については、『Mobile Link クライアント』> 「Mobile Link クライアントの紹介」を参照してください。

同期処理

「**同期**」とは、Mobile Link クライアントと同期サーバの間で行われる双方向のデータ交換処理です。この処理の最中、クライアントは同期サーバへの接続を確立して維持します。同期に成功した場合、セッションによってリモート・データベースと統合データベースは互いに一貫した状態に保たれます。

同期処理は通常クライアントが開始します。この処理は、Mobile Link 同期サーバとの接続を確立することから始まります。

アップロード・ストリームとダウンロード・ストリーム

ローをアップロードするために、Mobile Link クライアントが「**アップロード・ストリーム**」を準備し送信します。このアップロード・ストリームは、Mobile Link クライアント上で前回の同期以後に更新、挿入、または削除されたローすべてのリストを含みます。同様に、ローをダウンロードするために、挿入、更新、削除のリストを含む「**ダウンロード・ストリーム**」を Mobile Link 同期サーバが準備し送信します。

1. **アップロード・ストリーム**　Mobile Link クライアントは、前回成功した同期以後にリモート・データベースで挿入、更新、または削除されたローを自動的に追跡します。接続が確立すると、Mobile Link クライアントはこれらのすべての変更を記載したリストを同期サーバにアップロードします。

   アップロード・ストリームは、リモート・データベースで変更された各ローに対する新旧のロー値のセットで構成されます。ローが更新されたり削除されたりしていれば、前回成功した同期直後に存在していた値が古い値になります。ローが挿入または更新されていれば、現在のローの値が新しい値です。現在の状態に至るまでローが数回変更されていても、その中間値は送信されません。

   Mobile Link 同期サーバは、アップロード・ストリームを受信して統合データベースに変更を適用します。通常は、1 回のトランザクションですべての変更が適用されます。処理が完了すると、Mobile Link 同期サーバはトランザクションをコミットします。

注意

Mobile Link は、統合データベースのデフォルトの独立性レベルとして ODBC 独立性レベル SQL_TXN_READ_COMMITTED を使用して動作します。これは、競合を検出するには、読み出しを繰り返し可能にする必要があるためです。競合の検出スクリプトがない場合、または独立性を必要に応じたレベルに設定する場合は、begin_connection スクリプトにレベルを設定します。

2. **ダウンロード・ストリーム**　Mobile Link 同期サーバは、ユーザが作成した同期論理を使用して、Mobile Link クライアント側で挿入、更新、または削除されるローのリストを収集します。これらのローを Mobile Link クライアントにダウンロードします。このリストを収集するために、Mobile Link 同期サーバは統合データベースで新しいトランザクションを開きます。

   Mobile Link クライアントは、ダウンロード・ストリームを受信します。Mobile Link クライアントは、ダウンロード・ストリームの着信を、アップロードしたすべての変更内容が統合データベースで正常に適用されたことの確認と見なします。確認後、Mobile Link クライアントはこれらの変更内容が統合データベースに再送されないようにします。

   次に、Mobile Link クライアントは、ダウンロード・ストリームを自動的に処理して、古いローの削除、新しいローの挿入、変更されたローの更新を行います。これらの変更はすべて、リモート・データベース内の 1 つのトランザクションで適用されます。終了すると、トランザクションをコミットします。

3. **ダウンロード確認のオプション**　Mobile Link クライアントは、必要に応じて Mobile Link 同期サーバに短い確認メッセージを送信します。

   Mobile Link 同期サーバは確認メッセージを受信します。このメッセージによって、ダウンロードされたすべての変更をクライアントが受信して処理したことが同期サーバに伝えられます。これに応答して、手順 2 で開始されたダウンロード・トランザクションがコミットされます。

SendDownloadAck 拡張オプションの詳細については、『Mobile Link クライアント』> 「SendDownloadACK (sa) 拡張オプション」および 『Mobile Link クライアント』> 「Send Download Acknowledgement 同期パラメータ」を参照してください。

Mobile Link 同期中に情報が明確に交換されることはほとんどありません。クライアントは完全なアップロード・ストリームを構築してアップロードします。これに応答して、同期サーバは完全なダウンロード・ストリームを構築してダウンロードします。電話回線または公共無線ネットワークを使用しているために通信が低速で遅延時間が長い場合は、プロトコルのチャティネスの制限が特に重要になります。

Mobile Link イベント

Mobile Link クライアントが同期を開始すると、複数の同期イベントが発生します。イベントが発生すると、Mobile Link はその同期イベントに適したスクリプトを探します。このスクリプトには、実行する作業の概要を示す指示が含まれています。基本的な順序は次のとおりです。

イベントの発生 → *スクリプトの呼び出し* (存在する場合)

同期イベントおよびスクリプトの詳細については、次の項目を参照してください。

- 「同期イベント」401 ページ
- 「同期スクリプトの作成」285 ページ

Mobile Link スクリプト

イベントが発生すると、関連するスクリプトを作成していれば、Mobile Link 同期サーバがそのスクリプトを実行します。スクリプトが存在しなければ、次の順位のイベントが発生します。

テーブルに対する一般的な同期スクリプトを以下に示します。

イベント	スクリプト
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 (?,?)`
upload_fetch	`SELECT id, name, size, quantity, unit_price FROM Product WHERE id=?`

最初のイベント upload_insert は、emp_id と emp_name を emp テーブルに挿入する upload_insert スクリプトの実行をトリガします。これと同じように、upload_delete と upload_update テーブルは、同じ emp テーブルでの削除と更新アクションに対して同様の機能を実行します。

ダウンロード・スクリプトはカーソルを使用します。次に、download_cursor スクリプトの例を示します。

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

download_cursor は、指定された emp_name のデータを emp テーブルから取得します。

スクリプト内に COMMIT 文または ROLLBACK 文があると、同期手順のトランザクションの性質が変化してしまいます。これらの文を使用すると、障害が発生した場合にデータの整合性を保証できません。

同期スクリプト、または同期スクリプトから呼び出されるプロシージャやトリガで、暗黙的または明示的なコミットまたはロールバックを実行しないでください。

SQL、Java、または .NET でスクリプトを作成可能

統合データベースのネイティブ SQL ダイアレクトを使用するか、Java または .NET の同期論理を使用して、スクリプトを記述できます。Java と .NET 同期論理を使用すると、Mobile Link 同期サーバによって呼び出されるコードを記述して、データベースへの接続、変数の操作、Mobile Link やサポートされているリレーショナル・データベースで機能するユーザ定義のプロシージャの作成が可能です。同期の要件に適したルーチンを持つ Mobile Link Java API と Mobile Link .NET API があります。

同期論理のプログラミングの詳細については、「同期論理の作成オプション」31 ページを参照してください。

Oracle、MS SQL Server、IBM DB2、または Adaptive Server Enterprise のデータベースのスクリプト記述など、DBMS 依存のスクリプト記述については、「DBMS 依存の同期スクリプト」43 ページを参照してください。

スクリプトの格納

SQL スクリプトは統合データベースのシステム・テーブルに格納されます。Java と .NET の場合は、他のロケーションへのポインタが統合データベースに格納されます。統合データベースにスクリプトを追加する方法は 2 つあります。

- 統合データベースの作成時に Mobile Link システム・テーブルとともにインストールされるストアド・プロシージャを使用する。
- Sybase Central を使用する。

詳細については、「統合データベースでのスクリプトの追加と削除」301 ページを参照してください。

ストアド・プロシージャ

Mobile Link ストアド・プロシージャは、プログラム上の競合解決、スクリプトの追加、ユーザ認証、その他のカスタマイズ・プロシージャに使用されます。

カスタマイズのために Mobile Link ストアド・プロシージャを使用する方法については、「ストアド・プロシージャ」593 ページを参照してください。

通常、定義済みの手続き型言語を備えていないデータベースでは、手続き型の制御を取得するために別の手段が使用されます。たとえば、IBM の DB2 などのユーザ定義プロシージャが許可されていないデータベースでは、Java プロシージャが使用され、Mobile Link ストアド・プロシージャとして機能します。

Java または .NET 同期論理を使用してスクリプトを記述する方法の詳細については、「Java による同期スクリプトの作成」321 ページおよび「.NET での同期スクリプトの作成」353 ページを参照してください。

Adaptive Server Anywhere クライアントの場合は、「**クライアント・イベント・フック・プロシージャ**」というストアド・プロシージャを使用できます。このプロシージャは、リモート・データベースに格納されています。Mobile Link 同期処理に独自の論理を挿入する場合、各種のイベント・フック・プロシージャを使用できます。

詳細については、『Mobile Link クライアント』> 「dbmlsync クライアント・イベント・フック」を参照してください。

同期処理のトランザクション

Mobile Link 同期サーバは、各 Mobile Link クライアントからアップロードされた変更を 1 回のトランザクションで統合データベースに組み込みます。Mobile Link 同期サーバは、新しいローの挿入、古いローの削除、更新の実行、競合の解決が完了した後で、これらの変更をコミットします。

Mobile Link 同期サーバは、別のトランザクションを使用して、ダウンロード・ストリームを準備します。このダウンロード・ストリームには、すべての削除、挿入、更新処理が含まれます。SendDownloadAck の設定に応じて、Mobile Link 同期サーバは、Mobile Link クライアントから肯定の確認を受け取るまでこのトランザクションをコミットしません。クライアントがダウンロードの成功を確認すると、Mobile Link 同期サーバがダウンロード・トランザク

ションをコミットします。アプリケーションで問題が発生したり返信できなかったりする場合は、Mobile Link 同期サーバが代わりにダウンロード・トランザクションをロールバックします。

スクリプト内ではトランザクションをコミットしたりロールバックしたりしないでください
スクリプト内に COMMIT 文または ROLLBACK 文があると、同期手順のトランザクションの性質が変化してしまいます。これらの文を使用すると、障害が発生した場合にデータの整合性を保証できません。同期スクリプト、または同期スクリプトから呼び出されるプロシージャやトリガで、暗黙的または明示的なコミットまたはロールバックを実行しないでください。

ダウンロードした情報の追跡

ダウンロード・トランザクションの主な役割は、統合データベースのローを選択することです。ダウンロード情報のリモートへの送信に失敗しても、リモートが同じタイムスタンプを何度もアップロードするため、データが失われることはありません。

Mobile Link 同期サーバは、他にトランザクションを 2 つ使用します。1 つは同期の開始時に、もう 1 つは同期の終了時に使用します。これらのトランザクションは、各同期とその処理時間に関する情報を記録します。したがって、試行された同期、成功した同期、同期にかかった時間についての統計を記録できます。データは処理のさまざまな時点でコミットされるので、これらのトランザクションによって、データを失敗した同期の分析に役立てられるようにコミットできます。

同様に、Mobile Link クライアントはダウンロード・ストリームの情報を *1 つのトランザクション*内で処理します。ローを挿入、更新、削除して、リモート・データベースを統合データベースの最新の状態にします。

同期の障害処理の方法

Mobile Link は、フォールト・トレラントになっています。たとえば、同期中に通信リンクに障害が起きた場合は、リモート・データベースと統合データベースの両方が同じ状態のままになります。

クライアントでは、障害はリターン・コードで示されます。たとえば、Embedded SQL Ultra Light アプリケーションでは、ULSynchronize が戻ったときに SQLCode が SQLE_COMMUNICATION_ERROR に設定されます。

ただし、以下の 3 つのケースでは処理が異なります。

- **アップロード中の障害**　アップロード・ストリームの構築中や適用中に障害が起きた場合は、リモート・データベースは同期の起動時とまったく同じ状態のままになります。サーバでは、適用されたアップロード・ストリームのすべての部分がロールバックされます。

- **アップロードとダウンロード間の障害**　アップロード・ストリームの完了後、Mobile Link クライアントがダウンロード・ストリームを受信する前に障害が発生した場合、クライアントはアップロードした変更が統合データベースに適切に適用されたかどうかを確認できません。アップロード・ストリームが完全に適用されコミットされているか、サーバがアップロード・ストリーム全体を適用する前に障害が起きています。Mobile Link 同期サーバは、統合データベースにある不完全なトランザクションを自動的にロールバックします。

  アップロードされたすべての変更を再送する必要が生じた場合に備え、Mobile Link クライアントはそれらの記録を維持します。Mobile Link クライアントは、次に同期したときに前回のアップロード・ストリームの状態を要求してから、新しいアップロード・ストリームを構築します。前回のアップロードがコミットされていない場合は、新しいアップロード・ストリームに前回のアップロード・ストリームからの変更をすべて含めます。

- **ダウンロード中の障害**　ダウンロード・ストリームの適用中にリモート・デバイスで障害が起きた場合は、適用されたダウンロードはすべての部分がロールバックされ、リモート・データベースはダウンロード前と同じ状態のままになります。Mobile Link 同期サーバは、統合データベースのダウンロード・トランザクションを自動的にロールバックします。

どのような障害が発生しても、データは失われません。Mobile Link サーバと Mobile Link クライアントが障害時のデータ管理を行います。開発者やユーザは、アプリケーション内のデータが一貫性を保持しているかどうか心配する必要はありません。

アップロード・ストリームの処理方法

Mobile Link 同期サーバが Mobile Link クライアントからアップロード・ストリームを受信すると、同期が完了するまでアップロード・ストリーム全体が格納されます。これは 3 つの目的で行われます。

- **デッドロック**　アップロード・ストリームを統合データベースに適用すると、他のトランザクションとの同時実行性が原因でデッドロックが発生することがあります。そのトランザクションは、他の Mobile Link 同期サーバのデータベース接続からのアップロード・トランザクションの場合や、統合データベースを使用している他のアプリケーションからのトランザクションの場合があります。アップロード・トランザクションがデッドロックされている場合は、そのトランザクションはロールバックされ、Mobile Link 同期サーバが自動的にアップロード・ストリームを最初から適用し始めます。

パフォーマンスのヒント
競合をできるだけ避けるように同期スクリプトを書くことが重要です。複数のユーザが同時に同期しているときに競合が起きると、パフォーマンスに大きな影響があります。

- **ダウンロード・ローのフィルタ**　ダウンロードするローを決定する方法で最も一般的なのは、前回のダウンロード以後に修正されたローをダウンロードすることです。同期中は、ダウンロードよりアップロードが優先されます。アップロード中に挿入または更新されたローが、前回のダウンロード以後に修正されたローになります。

  アップロードの一部として送られたダウンロード・ストリームのローから除外する download_cursor スクリプトを書くのは困難です。このため、Mobile Link 同期サーバがダウンロード・ストリームからこのようなローを自動的に取り除きます。ローがダウンロード・ストリームに追加されていると、Mobile Link 同期サーバがそのローをアップロード・ストリーム内で探し、同じローを見つけたらダウンロード・ストリームから除外します。

- **挿入と更新後の削除の処理**　アップロード・ストリームは、参照整合性違反が起きないような順番で統合データベースに適用されます。アップロード・ストリームは、1 つのテーブルに対する

すべての操作 (挿入、更新、削除) がグループ分けされるような形でフォーマットされます。アップロード・ストリーム内のテーブルは、外部キー関係に基づいて並べられます。リモート・データベース内で別のテーブルに参照されているすべてのテーブルは、参照元テーブルより先にアップロード・ストリームに入ります。

たとえば、テーブル A とテーブル C の両方がテーブル B のプライマリ・キー・カラムを参照する外部キーを持っている場合は、テーブル B のローが先にアップロードされます。

アップロード・ストリームを統合データベースに適用すると、挿入と更新はアップロード・ストリーム内で現れる順に適用されます。挿入または更新されたローが新規挿入されたローを参照すると、これにより参照先ローが参照元ローより先に挿入されます。削除は、すべての挿入と更新が適用されてから逆の順序で適用されます。削除されるローが削除される別のローを参照すると、この操作の順序では、参照元ローが参照先ローより先に削除されることになります。

参照整合性と同期

すべての Mobile Link クライアントは、ダウンロード・ストリームをリモート・データベースに組み込むときに参照整合性を確保します。

参照整合性に違反するローがあった場合、Mobile Link クライアントはダウンロード・トランザクションを失敗させずに、参照整合性に違反するすべてのローを自動的に削除します。

この機能には、次のような利点があります。

- 同期スクリプトの間違いから保護します。スクリプトに柔軟性があると、リモート・データベースの整合性をそこなうローを誤ってダウンロードしてしまうことがあります。Mobile Link クライアントは、介入を要求せずに参照整合性を自動的に管理します。
- この参照整合性のメカニズムを使用して、リモート・データベースから情報を効率的に削除できます。親レコードに削除データを送信するだけで、Mobile Link クライアントはすべての

子レコードを自動的に削除します。これにより、Mobile Link がリモート・データベースに送信するトラフィックの量を大幅に減らすことができます。

Mobile Link クライアントは、参照整合性を維持するためにローを明示的に削除する必要がある場合は、通知を行います。

- Adaptive Server Anywhere クライアントの場合は、dbmlsync によってログにエントリが書き込まれます。
- Ultra Light クライアントの場合は、SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY 警告が発生します。この警告には、テーブル名のパラメータが含まれます。警告は、参照整合性を維持するために削除されるすべてのローで発生します。同期をそのまま進める場合は、警告を無視してかまいません。警告を明示的に処理する場合は、エラー・コールバック関数を使用して、たとえば、削除されたローをカウントしてその数を取得します。

  警告が発生したら同期を破棄する場合は、同期 observer を実装して、observer への次の呼び出しのときに同期を破棄するように、エラー・コールバック関数から observer に (たとえばグローバル変数を使用して) 通知する必要があります。

トランザクション終了時にチェックされる参照整合性

Mobile Link クライアントは、1 つのトランザクション内のダウンロード・ストリームから変更を組み込みます。柔軟性をより向上させるために、参照整合性のチェックはこのトランザクションの終了時に行われます。チェックが遅いため、データベースはローが挿入、更新、削除されたときに参照整合性に違反する状態を一時的にパスすることがあります。しかし、参照整合性に違反するローは、ダウンロードがコミットされる前に自動的に削除されます。

エラーの回避

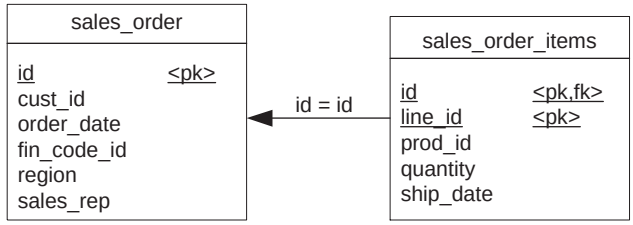
Mobile Link クライアントは、参照整合性に対する違反を自動的に解析します。この機能は、管理の要件を最小限に抑えます。また、同期スクリプト内のエラーが Mobile Link クライアントを無効にするのを防ぎます。Ultra Light クライアントは、参照整合性を維持するためにローが削除されると、警告を発生します。アプリケーションは、この警告を、処理することも無視することもできます。

ローを削除する効率的な方法

Mobile Link クライアントの自動参照整合性メカニズムを利用すると、大量の情報を非常に効率的に削除できます。使用している Mobile Link クライアントにプライマリ・ローとそのローを参照する他のローが含まれている場合は、プライマリ・ローの削除を同期するだけで、すべての参照元ローを削除できます。

例

Ultra Light 販売アプリケーションに、他のテーブルとともに次の 2 つのテーブルが含まれているとします。1 つのテーブルには、販売注文が含まれています。もう 1 つのテーブルには、各注文で販売された商品情報が含まれています。この 2 つのテーブルは、次のような関係です。



1 つの注文を削除するために sales_order テーブルに download_delete_cursor を使用すると、削除された受注を示す sales_order_items テーブルのすべてのローが、自動参照整合性メカニズムによって自動的に削除されます。

この方法には、次のような利点があります。

- sales_order_items テーブルからのローは自動的に削除されるので、sales_order_items スクリプトは必要ありません。
- 同期の効率が向上します。ダウンロード・ローを sales_order_item テーブルから削除する必要がありません。各販売注文に項目がたくさんある場合は、ダウンロード・ストリームが小さくなるのでパフォーマンスが向上します。この方法は、速度の遅い通信方法を使用しているときに特に役立ちます。

アップロード専用の同期とダウンロード専用の同期

Adaptive Server Anywhere と Ultra Light の両方のリモート・データベースでは、完全な同期を実行するか、アップロード専用の同期またはダウンロード専用の同期を実行するかを選択できます。

Adaptive Server Anywhere リモート・データベース

Adaptive Server Anywhere リモート・データベースでは、dbmlsync オプション -ds または拡張オプション DownloadOnly を使用してアップロード専用の同期を実行します。

詳細については、『Mobile Link クライアント』> 「-ds オプション」および『Mobile Link クライアント』> 「DownloadOnly (ds) 拡張オプション」を参照してください。

Adaptive Server Anywhere リモート・データベースでは、dbmlsync オプション -uo または拡張オプション UploadOnly を使用してダウンロード専用の同期を実行します。

詳細については、『Mobile Link クライアント』> 「-uo オプション」または『Mobile Link クライアント』> 「UploadOnly (uo) 拡張オプション」を参照してください。

Ultra Light リモート・データベース

Ultra Light リモート・データベースでは、DownloadOnly 同期パラメータを使用してダウンロード専用の同期を実行します。

詳細については、『Mobile Link クライアント』> 「Download Only 同期パラメータ」を参照してください。

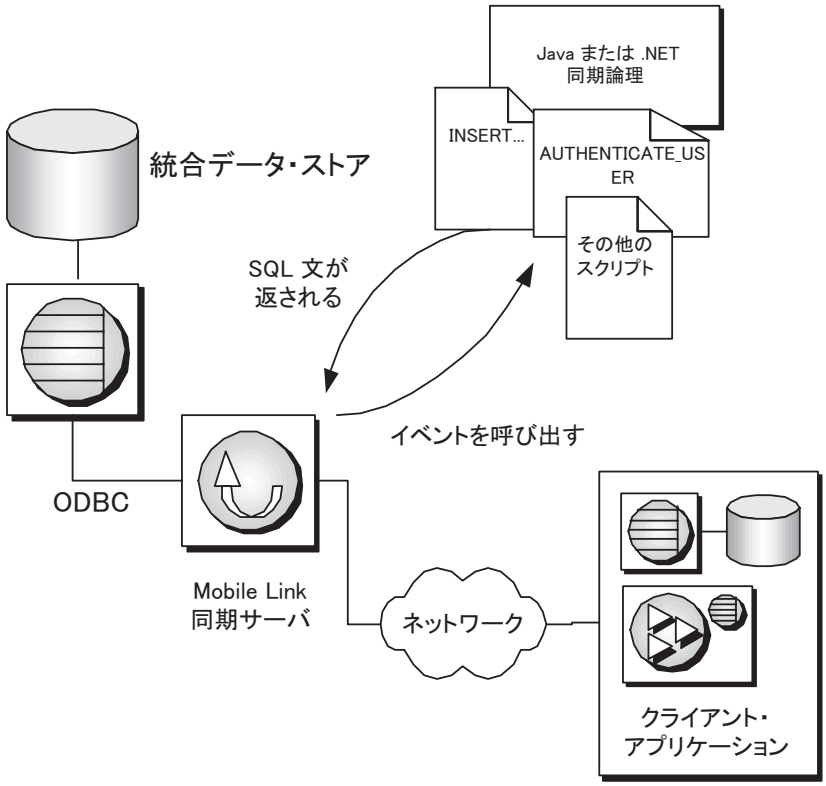
Ultra Light リモート・データベースでは、UploadOnly 同期パラメータを使用してアップロード専用の同期を実行します。

詳細については、『Mobile Link クライアント』> 「Upload Only 同期パラメータ」を参照してください。

同期論理の作成オプション

Mobile Link 同期スクリプトは、SQL、Java、.NET プログラミング言語で記述できます。SQL 言語の制限事項や、データベース管理システムの機能によって設計が制限されている場合、または DBMS 依存の同期論理を使用したい場合は、Java や .NET を使用することをおすすめします。

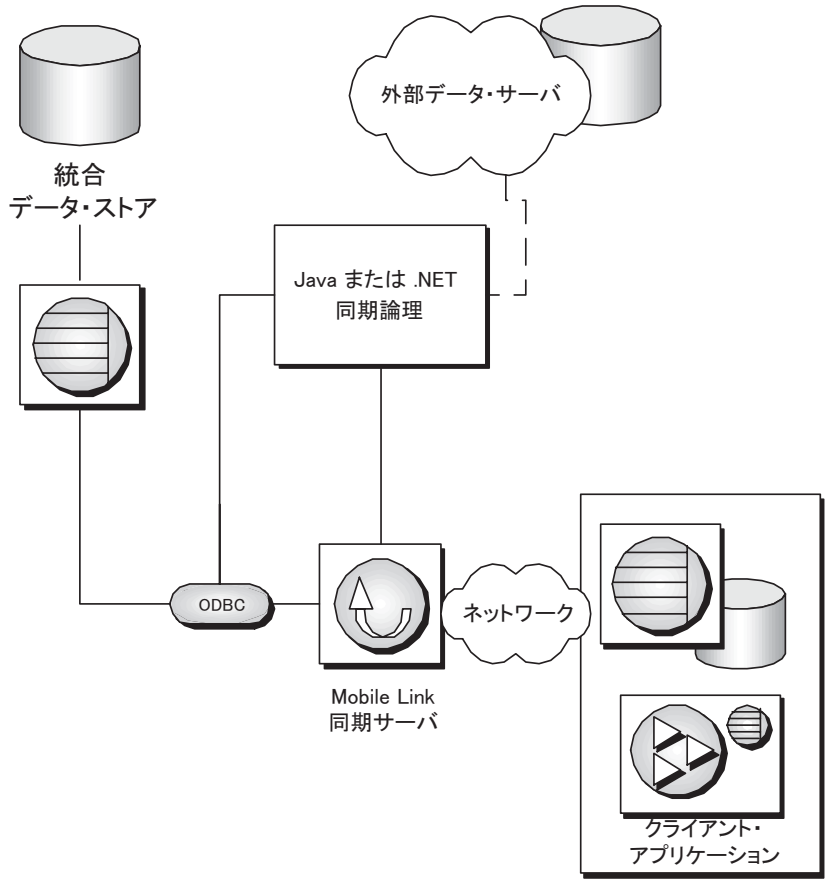
プログラム同期論理には、次の図に示すように、SQL 論理と同じ機能を持たせることができます。Mobile Link 同期サーバは、Mobile Link イベントの発生時に SQL スクリプトにアクセスするのと同様に、Java メソッドや .NET メソッドを呼び出すことができます。ただし、SQL 文字列を Mobile Link に返す必要がある場合は、Java または .NET の同期論理からアップロード・ストリームとダウンロード・ストリームに直接アクセスすることはできません。



SQL 同期論理は、統合データベースの手続き型言語機能に制限されています。Java や .NET が持つプログラミング論理をすべて備えている SQL 言語はほとんどありません。SQL 論理が制限されているとき、複数のデータベース・プラットフォームで操作を実行する必要があるとき、RDBMS とオペレーティング・システムの間の移植性を必要とするときに、Java または .NET の同期論理を使用できます。Java または .NET でのスクリプトの作成を検討する場合のシナリオを以下に示します。

- ユーザ認証プロシージャを Java や .NET で作成し、Mobile Link ユーザ ID を監査目的で統合データベースのテーブルに挿入できます。

- データベースに変数を処理する機能がない場合は、接続または同期を通して持続する変数を Java や .NET で作成できます。
- データベースにユーザ定義のストアド・プロシージャを作成する機能がない場合は、必要な機能を実行できるメソッドを Java や .NET で作成できます。
- プログラムが同期イベント中に外部サーバへの接続を必要とする場合は、Java または .NET 同期論理を使用して、同期イベントによりトリガされるアクションを実行できます。Java または .NET 同期論理は、複数の接続間で共有できます。
- Java または .NET 同期論理を使用して、Mobile Link でアプリケーション・サーバ、Web サーバ、ファイルのデータにアクセスできます。同期論理に JDBC や iAnywhere のクラスを使用すると、統合データベース以外のリレーショナル・データベース内のデータにアクセスできます。たとえば、外部サーバを使用して、ユーザ ID とパスワードを検証できます。次の図は、Java または .NET 同期論理と、統合データベースと 2 番目のデータ・サーバの両方とのリンクを示します。



Mobile Link API

Java と .NET 同期論理を使用すると、Mobile Link API にアクセスできます。Mobile Link API は、Mobile Link 同期用のクラスとインタフェースのセットです。Mobile Link API には、Java と .NET の 2 つがあります。

Mobile Link Java API には、次の利点があります。

- 既存の ODBC 接続に JDBC 接続としてアクセスできます。
- 新しい JDBC 接続を作成し、現在の同期接続の外部でコミットまたは接続を実行できます。たとえば、これをエラー・ログに使用できます。

- Java コードを作成してデバッグしてから、Mobile Link サーバで実行できます。多くのデータベース管理システムの SQL 開発環境は、Java アプリケーションが使用可能な環境に比べると初歩的です。
- Java 仮想マシン内でコードが実行されるため、すべての Java ライブラリと Java ネイティブ・インタフェース呼び出しを使用できます。

詳細については、「Mobile Link Java API リファレンス」344 ページを参照してください。

Mobile Link .NET API には、次の利点があります。

- .NET から ODBC を呼び出す iAnywhere クラスを使用して既存の ODBC 接続にアクセスできます。
- .NET Common Language Runtime (CLR) 内でコードが実行されるため、すべての .NET ライブラリと管理されていない呼び出しを使用できます。

詳細については、「Mobile Link .NET API リファレンス」381 ページを参照してください。

詳細情報

同期スクリプトの作成オプションの詳細については、以下を参照してください。

- 「同期スクリプトの作成」285 ページ
- 「同期の方法」57 ページ
- 「Java による同期スクリプトの作成」321 ページ
- 「.NET での同期スクリプトの作成」353 ページ

セキュリティ

Mobile Link のインストール環境のように、広範囲の分散システム全体のデータの安全を確保するには、いくつかの要素があります。

- **統合データベースのデータ保護**　統合データベース内のデータは、DBMS のユーザ認証システムとその他のセキュリティ機能で保護できます。

  詳細については、使用している DBMS のマニュアルを参照してください。Adaptive Server Anywhere 統合データベースを使用している場合は、『SQL Anywhere Studio セキュリティ・ガイド』> 「安全なデータの管理」を参照してください。

- **リモート・データベースのデータ保護**　Adaptive Server Anywhere リモート・データベースを使用している場合、Adaptive Server Anywhere のセキュリティ機能を使用してデータを保護できます。このセキュリティ機能は、デフォルトではクライアント／サーバ通信での不正なアクセスを防止するように設計されていますが、データベース・ファイルから直接情報を抽出するという悪質な攻撃に対する保証にはなりません。

  クライアント上のファイルは、クライアントのオペレーティング・システムのセキュリティ機能によって保護されます。

  Adaptive Server Anywhere リモート・データベースを使用している場合は、『SQL Anywhere Studio セキュリティ・ガイド』> 「安全なデータの管理」を参照してください。

- **同期中のデータ保護**　Mobile Link クライアントから Mobile Link 同期サーバへの通信は、Mobile Link トランスポート・レイヤ・セキュリティ機能で保護できます。

  詳細については、「Mobile Link トランスポート・レイヤ・セキュリティ」205 ページを参照してください。

- **権限のないユーザからの同期システムの保護**　Mobile Link 同期は、パスワードによるユーザ認証システムで保護できます。このメカニズムにより、権限のないユーザはデータを同期できなくなります。

詳細については、『Mobile Link クライアント』> 「Mobile Link ユーザの認証」を参照してください。

第 3 章

Mobile Link 統合データベース

この章の内容

この章では、統合データベースの設定方法および使用方法について説明します。

概要

統合データベースには、ODBC 互換データベースである Adaptive Server Anywhere、Adaptive Server Enterprise、Oracle、IBM DB2、Microsoft SQL Server のいずれかを使用できます。同期スクリプトを使って特定の統合サーバの機能を利用できます。

同期スクリプトは、統合データベースに関連付けられています。SQL スクリプトは統合データベースに格納されており、Java スクリプトと .NET スクリプトが参照されます。

各タイプのデータベースを統合データベースとして設定する方法については、「統合データベースの設定」42 ページを参照してください。

特定の統合データベース用の同期スクリプトを記述する方法については、「DBMS 依存の同期スクリプト」43 ページを参照してください。

リモート・テーブルと統合データベースの関係

同期は、リモート・データベースと統合データベースにあるテーブルとローの間のマッピングを指定できるように設計されています。

許可されている任意の関係

リモート・データベースにあるテーブルは、統合データベースのテーブルと同じである必要はありません。1 つのリモート・アプリケーション・テーブルで同期されたデータは、異なるテーブルのカラムに配布できます。また、異なる統合データベースのテーブル間でも配布できます。これらの関係を指定するには、同期スクリプトを使用します。

シンプルな直接関係

統合データベースのテーブル構造のサブセットであるリモート・データベースのテーブル構造を使用すると、設計を簡素化することができます。この方法を使うと、リモート・データベースの各テーブルが統合データベースに存在するようになります。対応するテーブルの構造体と外部キーの関係が、統合データベースのものと同じになります。

統合データベースのテーブルには、同期されていない余分なカラムが含まれていることがよくあります。実際のところ、このようなカラムは同期を支援します。たとえば、タイムスタンプ・カラムから統合

データベース内で新規作成または更新されたローを識別できます。他にも、統合データベース内の余分なカラムやテーブルにリモート・サイトでは必要ない情報が保存されています。

統合データベースの設定

設定スクリプト

Mobile Link 統合データベースとして使用できるようにデータベースを設定するには、Mobile Link システム・テーブルとストアド・プロシージャをインストールする「**設定スクリプト**」を実行する必要があります。例外は、Adaptive Server Anywhere データベースです。これは適切なシステム・テーブルとストアド・プロシージャを使用して事前に設定されています。設定スクリプトの実行方法については、サポート対象の各 RDBMS についての以下の項を参照してください。

Mobile Link システム・テーブルの詳細については、『Mobile Link 管理ガイド』> 「Mobile Link システム・テーブル」を参照してください。

ストアド・プロシージャの詳細については、「ストアド・プロシージャ」593 ページを参照してください。

ODBC 接続

さらに、Mobile Link 同期サーバに統合データベースへの ODBC 接続が必要です。使用しているサーバ用の適切な ODBC ドライバを設定して、Mobile Link 同期サーバを実行しているコンピュータのデータベース用に ODBC データ・ソースを作成してください。

サポートされている ODBC ドライバの概要については、『Mobile Link 管理ガイド』> 「iAnywhere Solutions ODBC ドライバ」を参照してください。

最新の情報と完全な機能仕様については、http://www.ianywhere.com/developer/technotes/odbc_mobilink.html (英文) を参照してください。

Mobile Link 統合データベース用の ODBC ドライバの設定については、『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』> 「iAnywhere Solutions ODBC ドライバの概要」を参照してください。

各タイプの統合データベースの固有の情報については、該当の項を参照してください。

- 「Adaptive Server Anywhere 統合データベース」46 ページ
- 「Sybase Adaptive Server Enterprise 統合データベース」47 ページ
- 「Oracle 統合データベース」49 ページ
- 「IBM DB2 統合データベース」51 ページ
- 「Microsoft SQL Server 統合データベース」54 ページ

DBMS 依存の同期スクリプト

Mobile Link では、データを同期するときに使用する規則に柔軟性を提供するために、同期スクリプトを使用しています。スクリプトでは次の内容を定義します。

- リモート・データベースからアップロードしたデータを、統合データベースに適用する方法。
- 統合データベースからダウンロードするデータ。

同期スクリプト作成の詳細については、「同期スクリプトの作成」285 ページを参照してください。

スクリプトに記述できるイベントの一覧については、「同期イベント」401 ページを参照してください。

スクリプトには、使用する DBMS に依存する側面があります。ODBC 準拠データベースとの同期に必要なスクリプトの種類は、さまざまな要素によって決定されます。要素には次のようなものがありますが、これだけに限定されません。

- ユーザ定義プロシージャ
- セッション全体の接続変数

  セッション全体の変数は、同期全体をとおして保持される値を格納するための効果的な手段です。たとえば、begin_synchronization スクリプトは、UserName というセッション全体の変数にユーザ名を格納します。後続のスクリプトでは、何度でも UserName を参照できます。

- オートインクリメント・メソッド

  これがサポートされている場合は、オートインクリメントを使用して、Mobile Link 環境内でユニークなプライマリ・キー値を管理できます。詳細については、「ユニークなプライマリ・キーの管理」70 ページを参照してください。

各タイプの統合データベースの固有の情報については、該当の項を参照してください。

- 「Adaptive Server Anywhere 統合データベース」46 ページ
- 「Sybase Adaptive Server Enterprise 統合データベース」47 ページ
- 「Oracle 統合データベース」49 ページ
- 「IBM DB2 統合データベース」51 ページ
- 「Microsoft SQL Server 統合データベース」54 ページ

.NET 同期スクリプトと Java 同期スクリプト

上記のようなデータベースで Mobile Link を使用する場合、テーブル・スクリプトと同期論理を、SQL 言語の DBMS バージョンで記述するのも 1 つの方法です。また、Java や .NET の同期論理を使う方法もあります。Java または .NET の同期論理を使用する場合は、セッション全体の変数を保持し、ユーザ定義のプロシージャを作成できます。

Java 同期論理については、「Java 同期論理の作成」326 ページを参照してください。

.NET 同期論理については、「.NET での同期スクリプトの作成」353 ページを参照してください。

スクリプトからのプロシージャの呼び出し

Microsoft SQL Server などのいくつかのデータベースでは、パラメータを持つプロシージャ・コールは、ODBC の構文を使用して記述する必要があります。

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

プロシージャ定義の中でパラメータを OUT または INOUT として定義することで、戻り値を渡すことができます。

CHAR カラム

他の多くの DBMS では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモートデータベース (Adaptive Server Anywhere または Ultra Light) では、CHAR は VARCHAR と同じで、固定幅に合わせて値にブランクが埋め込まれることはありません。dbmlsrv9 -b コマンド・ライン・オプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

詳細については、『Mobile Link 管理ガイド』> 「-b オプション」を参照してください。

データ変換

Mobile Link 同期サーバが、Adaptive Server Anywhere 以外で作成された統合データベースと通信するときに行われるデータ変換については、「データ型の変換」653 ページを参照してください。

Adaptive Server Anywhere 統合データベース

Adaptive Server Anywhere データベースは、設定スクリプトを実行しなくても Mobile Link 統合データベースとして使用できるように自動的に設定されます。

Adaptive Server Anywhere データベースの設定スクリプトは、ソース・コードを調べる必要がある場合に備えて用意されています。たとえば、ml_add_connection_script ストアド・プロシージャのソース・コードが含まれています。この設定スクリプトは *syncasa.sql* というファイルで、SQL Anywhere インストール環境の *scripts* サブディレクトリにあります。

ODBC ドライバの設定

Adaptive Server Anywhere 統合データベースには、ODBC DSN を設定する必要があります。Adaptive Server Anywhere 用の ODBC ドライバは、SQL Anywhere Studio とともにインストールされます。

Adaptive Server Anywhere ODBC ドライバについては、『ASA データベース管理ガイド』> 「ODBC データ・ソースの使用」を参照してください。

Sybase Adaptive Server Enterprise 統合データベース

Mobile Link 統合データベースとして機能するように Adaptive Server Enterprise バージョン 12.5 以降を設定するには、SQL Anywhere インストール環境の *MobiLink¥setup* サブディレクトリにある *syncase125.SQL* 設定スクリプトを実行します。バージョン 12.5 より前の場合は、同じロケーションの *syncase.SQL* を実行します。

Mobile Link 管理者が (スクリプトやその他の追加で必要な) Mobile Link システム・テーブルを変更するための適切なパーミッションを持つようにするには、これらのテーブルをアクセスする必要のあるユーザが設定スクリプトを実行するようにします。

ODBC ドライバ

Adaptive Server Enterprise 統合データベースには、ODBC DSN を設定する必要があります。SQL Anywhere Studio には、Adaptive Server Enterprise 用の iAnywhere Solutions ODBC ドライバがあります。このドライバを設定して、Mobile Link を使用できるようにする必要があります。

詳細については、『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』> 「Sybase Adaptive Server Enterprise 対応の iAnywhere Solutions ODBC ドライバ」を参照してください。

Adaptive Server Enterprise の問題

- **カラム・サイズ** BLOB データを Adaptive Server Enterprise 統合データベースからダウンロードするには、カラム・サイズを 4096 バイトより大きくできるように ODBC ドライバ接続オプションを設定してください。このためには、StaticCursorLongColBuffLen という ODBC ドライバ接続オプションを使用します。次に例を示します。

  ```
  dbmlsrv9 -c "...;StaticCursorLongColBuffLen=number"
  ```

  number はバイト単位で、予想される最大 BLOB より大きくします。

  このオプションを使用すると、Mobile Link 同期サーバが稼働しているコンピュータのディスクが大幅に消費されることに注意してください。

- **numeric 型のプライマリ・キー値** また、Mobile Link 同期サーバでは、numeric または decimal 型のプライマリ・キー値を Adaptive Server Enterprise で使用する型に明示的に変換する必要があります。

  スクリプトには、次の例で示すように、数値パラメータに明示的に変換するコードを追加します。

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

  上記の文では、1 つ目のパラメータを NUMERIC データ型に明示的に変換しています。

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

  上記の文では、1 つ目のパラメータを DECIMAL (10,8) データ型に明示的に変換しています。

- **CHAR カラム** Adaptive Server Enterprise では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモート データベース (Adaptive Server Anywhere または Ultra Light) では、CHAR は VARCHAR と同じで、固定幅に合わせて値にブランクが埋め込まれることはありません。dbmlsrv9 -b コマンド・ライン・オプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

  詳細については、『Mobile Link 管理ガイド』> 「-b オプション」を参照してください。

- **データ型マッピング** Adaptive Server Anywhere データ型と Adaptive Server Enterprise データ型のマッピングの詳細については、「Adaptive Server Enterprise データのマッピング」654 ページを参照してください。

Oracle 統合データベース

Mobile Link 統合データベースとして機能するように Oracle を設定するには、SQL Anywhere インストール環境の *MobiLink¥setup* サブディレクトリにある *syncora.sql* 設定スクリプトを実行します。

Mobile Link 管理者が (スクリプトやその他の追加で必要な) Mobile Link システム・テーブルを変更するための適切なパーミッションを持つようにするには、これらのテーブルをアクセスする必要のあるユーザが設定スクリプトを実行するようにします。

ODBC ドライバ

Oracle 統合データベースには、ODBC DSN を設定する必要があります。SQL Anywhere Studio には、Oracle 用の iAnywhere Solutions ODBC ドライバがあります。このドライバを設定して、Mobile Link を使用できるようにする必要があります。

詳細については、『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』> 「Oracle 対応の iAnywhere Solutions ODBC ドライバ」を参照してください。

Oracle の問題

- **セッション全体の変数** Oracle ではセッション全体の変数は提供されていません。セッション全体の情報を保持するには、Oracle パッケージ内の変数を使用します。Oracle パッケージでは、変数の作成、修正、破棄が可能です。これらの変数は Oracle パッケージが現在のパッケージであるかぎり有効です。

- **オートインクリメント・メソッド** プライマリ・キーの一意性を維持するには、Oracle シーケンスを使用して、オートインクリメント・フィールドのキーのリストに似た、キーのリストを生成します。CustDB サンプル・データベースの *Samples¥MobiLink¥CustDB¥custora.sql* に、サンプル・コーディングが用意されています。ただし、オートインクリメントとは異なり、シーケンスを明示的に参照する必要があります。オートインクリメントは、INSERT 文でカラムが参照されていない場合は、自動的にカラム値を挿入します。

  Oracle シーケンスの使用例については、『Mobile Link チュートリアル』> 「チュートリアル : Oracle 8i 統合データベースでの Mobile Link の使用」を参照してください。

- **CHAR カラム** Oracle では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモート データベース (Adaptive Server Anywhere または Ultra Light) では、CHAR は VARCHAR と同じで、固定幅に合わせて値にブランクが埋め込まれることはありません。dbmlsrv9 -b コマンド・ライン・オプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

  詳細については、『Mobile Link 管理ガイド』> 「-b オプション」を参照してください。

- **データ型マッピング** Adaptive Server Anywhere データ型と Oracle データ型のマッピングの詳細については、「Oracle データのマッピング」658 ページを参照してください。

IBM DB2 統合データベース

Mobile Link 統合データベースとして機能するように IBM DB2 UDB を設定するには、SQL Anywhere インストール環境の *MobiLink¥setup* サブディレクトリにある *syncdb2long.sql* 設定スクリプトを実行します。ただし、次の手順を実行してから、スクリプトを実行してください。

設定スクリプトを使用して Mobile Link システム・テーブルをインストールするには、IBM DB2 テーブル領域は最低でも 8 KB ページを使用します。テーブル領域が 8 KB ページを使用しない場合は、次の手順を実行します。

- 1 つ以上のバッファ・プールに 8 KB ページがあることを確認します。ない場合は、8 KB ページのバッファ・プールを作成します。
- 8 KB ページのバッファ・プールを使用する、新しいテーブル領域とテンポラリ・テーブル領域を作成します。詳細については、DB2 のマニュアルを参照してください。

Mobile Link 管理者が (スクリプトやその他の追加で必要な) Mobile Link システム・テーブルを変更するための適切なパーミッションを持つようにするには、これらのテーブルをアクセスする必要のあるユーザが設定スクリプトを実行するようにします。

syncdb2long.sql スクリプトには、デフォルトの接続文 `connect to DB2Database` が含まれています。スクリプトのコピーを作成して、各自のインストール環境に適合するようにこの行を変更してください。この行には、次の構文を使用してください。

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

ここでは、*DB2Database*、*userid*、*password* に適切な名前を指定します。

syncdb2long.sql スクリプトでは、チルド (~) をコマンド・デリミタとして使用します。次のように入力すると、このスクリプトを実行できます。

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

DB2 が、*syncdb2long.sql* で定義されているストアド・プロシージャを使用するためには、SQL Anywhere インストール環境の *MobiLink¥setup* サブディレクトリにある、*syncdb2long_version* Java およびクラス・ファイルを、DB2 インストール環境の *FUNCTION* サブディレクトリにコピーします。

ODBC ドライバ

DB2 統合データベースに ODBC DSN を設定するには、iAnywhere Solutions DB2 Wire Protocol ドライバを使用します。

詳細については、『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』> 「DB2 対応の iAnywhere Solutions ODBC ドライバ」を参照してください。

DB2 の問題

- **テーブル領域の容量** DB2 データベースを統合データベースとして使用する場合のテーブル領域およびテンポラリ・テーブル領域は、8 KB ページを使用します。

  また、LONG テーブル領域が必要なカラムもあります。次の例のように、デフォルトの LONG テーブル領域がない場合は、これらのカラムを含むテーブルの作成文を正しく設定します。

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

  サンプル・アプリケーションの使用例については、『Mobile Link チュートリアル』> 「CustDB サンプル・アプリケーション」を参照してください。

- **セッション全体の変数** DB2 の以前のバージョン (バージョン 8 より以前) では、セッション全体の変数はサポートされていません。これを解決する便利な方法としては、Mobile Link ユーザ名と他のセッション・データ用のカラムがあるベース・テーブルを使用します。ベース・テーブルには、同時同期を表すローが含まれます。

- **ユーザ定義のプロシージャ** DB2 が SQL プロシージャを実行可能ライブラリ (DLL など) にコンパイルするには、C コンパイラが必要です。作成された DLL/ 共有ライブラリは、サー

バ上の特別なディレクトリにコピーする必要があります。C/C++ や Java など、さまざまな言語を使用してストアド・プロシージャを記述できることに注意してください。

DB2 用手続き型言語としての Java の例については、ファイル *Samples¥MobiLink¥CustDB¥custdbq.sql* と *Samples¥MobiLink¥CustDB¥custdbq.java* にある CustDB スクリプトを参照してください。

Java と .NET の同期スクリプトの詳細については、以下の項目を参照してください。

- 「同期論理の作成オプション」31 ページ
- 「Java による同期スクリプトの作成」321 ページ
- 「.NET での同期スクリプトの作成」353 ページ

- **CHAR カラム** IBM DB2 では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモート データベース (Adaptive Server Anywhere または Ultra Light) では、CHAR は VARCHAR と同じで、固定幅に合わせて値にブランクが埋め込まれることはありません。dbmlsrv9 -b コマンド・ライン・オプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

  詳細については、『Mobile Link 管理ガイド』> 「-b オプション」を参照してください。

- **データ型マッピング** Adaptive Server Anywhere データ型と DB2 データ型のマッピングの詳細については、「IBM DB2 データのマッピング」656 ページを参照してください。

Microsoft SQL Server 統合データベース

Mobile Link 統合データベースとして機能するように Microsoft SQL Server を設定するには、SQL Anywhere インストール環境の *MobiLink¥setup* サブディレクトリにある *syncmss.sql* 設定スクリプトを実行します。

Mobile Link 管理者が (スクリプトやその他の追加で必要な) Mobile Link システム・テーブルを変更するための適切なパーミッションを持つようにするには、これらのテーブルをアクセスする必要のあるユーザが設定スクリプトを実行するようにします。

ODBC ドライバ

Microsoft SQL Server 統合データベースには、ODBC DSN を設定する必要があります。他の統合データベースとは異なり、アイエニウェア・ソリューションズでは Microsoft SQL Server 用の ODBC ドライバを提供していません。この理由は、Microsoft SQL Server のドライバが自由にダウンロードできるためです。

最新情報については、http://www.ianywhere.com/developer/technotes/odbc_mobilink.html (英文) を参照してください。

SQL Server の問題

- **プロシージャ・コール** Microsoft SQL Server では、パラメータを持つプロシージャ・コールは、ODBC の構文を使用して記述する必要があります。

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

- **CHAR カラム** Microsoft SQL Server では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモート データベース (Adaptive Server Anywhere または Ultra Light) では、CHAR は VARCHAR と同じで、固定幅に合わせて値にブランクが埋め込まれることはありません。dbmlsrv9 -b コマンド・ライン・オプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

  詳細については、『Mobile Link 管理ガイド』> 「-b オプション」を参照してください。

- **データ型マッピング** Adaptive Server Anywhere データ型と SQL Server データ型のマッピングの詳細については、「Microsoft SQL Server データのマッピング」660 ページを参照してください。

第 4 章

同期の方法

この章の内容

この章では、Mobile Link のインストール環境で一般的な同期タスクを実行する場合に使用できるさまざまな方法について説明します。

この章で説明する方法の例を示すサンプル・アプリケーションがあります。詳細については、『Mobile Link チュートリアル』> 「Contact サンプル・アプリケーション」および『Mobile Link チュートリアル』> 「CustDB サンプル・アプリケーション」を参照してください。

この章では、各方法について、SQL スクリプトを使用して説明します。多くの方法は Java または .NET 同期論理でも実装できます。詳細については、次の項目を参照してください。

- 「Java による同期スクリプトの作成」321 ページ
- 「.NET での同期スクリプトの作成」353 ページ

概要

簡単な同期スクリプトを作成する方法、作成した同期スクリプトをデータベースに格納する方法、構文エラーがないかどうかをテストする方法については、「同期スクリプトの作成」285 ページの章を参照してください。

便利な同期機能の多くは、1 つのスクリプトだけでなく、いくつかのスクリプトをまとめて実行する必要があります。この章では、いくつかの一般的な同期方法を実装する方法について説明します。どの例でも、SQL 同期スクリプトについて説明します。Java または .NET 同期論理を使用することもできますが、アップロード・イベントとダウンロード・イベントには SQL スクリプトの知識が必要です。

例

Contact サンプル・アプリケーションで Customer テーブルをタイムスタンプベースで同期する場合は、以下のスクリプトを使用します。

- **upload_insert** スクリプト。リモート・データベース側で追加された新しいローを、統合データベース側で処理する。
- **upload_update** スクリプト。リモート・データベース側で行われた修正を、統合データベース側で処理する。
- **upload_delete** スクリプト。リモート・データベースから削除されたローを、統合データベース側で処理する。
- **download_cursor** スクリプト。新しいローと更新されたローをリモート・データベースにダウンロードする。
- **download_delete_cursor** スクリプト。リモート・データベースから削除されるローをダウンロードする。

開発のヒント

同期機能をアプリケーションに追加すると、複雑なアプリケーションを作成できます。以下のヒントを参考にしてください。

- **慌てずにテストしてみる**　同期機能をプロトタイプ・アプリケーションに追加する場合、トラブルの原因となるコンポーネントを推測するのは困難です。Ultra Light アプリケーションの場合は、データベースとアプリケーションが一緒にコンパイルされるので推測は特に困難です。プロトタイプの開発時に、アプリケーションの内部に INSERT 文を一時的にハード・エンコードして、テストやデモンストレーションのためのデータを作成してください。プロトタイプが正常に動作するようになったら、同期を有効にして、一時的に使用した INSERT 文を削除します。

- **1 ステップずつ実行してみる**　簡単な同期の方法から始めてください。簡単なアップロードまたはダウンロードを行う場合、スクリプトは 1 つか 2 つしか必要ありません。スクリプトが正しく動作していれば、タイムスタンプ、プライマリ・キー・プール、競合解決などのより高度な方法を導入できます。

Mobile Link 同期アプリケーションの基本的な規則の一部を以下に示します。

- 同期される各テーブルには、プライマリ・キーが存在する必要がある。

- プライマリ・キーの値は更新しない。

- プライマリ・キーは、同期されるすべてのデータベース間でユニークでなければならない。

- 各リモート・データベースを識別する Mobile Link ユーザ名はユニークでなければならない。

タイムスタンプベースの同期

タイムスタンプによる方法は、効率よく同期するために最も便利な一般的な手法です。この方法では、各ユーザが最後に同期を行った時間が追跡され、この情報をもとに各リモート・データベースにダウンロードするローが制御されます。

Mobile Link は、各 Mobile Link ユーザが最後にデータをダウンロードした日時を示すタイムスタンプ値を管理します。この値は、「**最終ダウンロード・タイムスタンプ**」と呼ばれます。最終ダウンロード・タイムスタンプは、多くのイベントにパラメータとして指定され、同期スクリプトで使用できます。

Adaptive Server Anywhere 統合データベースを使用していて、最終変更情報を保持しているカラムが DEFAULT TIMESTAMP 型の場合は、カラムを同期する必要があります。リモート・データベースがこのようなカラムを要求する場合は、別のカラム名を使用してください。そうしないと、デフォルトのタイムスタンプ値が、アップロードされた値で上書きされ、統合データベースでローが最後に変更された時刻が保持されません。

❖ テーブル用にタイムスタンプベースの同期を実装するには、次の手順に従います。

1. 統合データベースで、ローの最終修正時刻を保持するカラムを追加します。通常、このカラムは次のように宣言されます。

DBMS	最終変更カラム
Adaptive Server Anywhere	`timestamp DEFAULT timestamp`
Adaptive Server Enterprise	`datetime`
Microsoft SQL Server	`datetime`
Oracle	`date`
IBM DB2	`timestamp`

2　download_cursor イベントと download_delete_cursor イベントのスクリプト内で、最初のパラメータを timestamp カラムの値と比較します。

例

次のテーブル宣言とスクリプトによって、Contact サンプルの Customer テーブルに対するタイムスタンプベースの同期が実装されます。

- テーブル定義

```
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 スクリプト

```
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 スクリプト

```
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
```

詳細については、『Mobile Link チュートリアル』> 「同期論理のソース・コード」および『Mobile Link チュートリアル』> 「Contact サンプルの顧客窓口の同期」を参照してください。

スナップショットを使った同期

ほとんどの同期には、タイムスタンプベースの同期が適しています。ただし、スナップショットによってデータの更新をしたい場合も考えられます。

スナップショットを使ってテーブルを同期する場合、テーブルのローのうちで関係するローすべてを完全にダウンロードします。すでにダウンロード済みのローもダウンロードされます。この方法が最も簡単ですが、不必要に大量のデータ・セットが交換されるため、パフォーマンスが悪くなります。

スナップショットを使った同期によって、テーブルのすべてのローをダウンロードできます。また、「リモート・データベース間でローを分割する」65 ページで説明しているローの分割方法と組み合わせて実行することもできます。

スナップショットを使った同期をいつ行うか

通常、次の特徴を両方満たすテーブルに対してスナップショットを使用すると最も有効です。

- **ロー数が比較的少ない**　ローの数が少ない場合は、ローを全部ダウンロードしても大きなオーバヘッドにはなりません。
- **頻繁にローが変更される**　テーブルのほとんどのローが頻繁に変更される場合は、前回の同期の後で変更されていないローを明示的に除外してもあまり効果はありません。

テーブルの内容が為替レートのリストになっている場合は、通貨の種類はそれほど多くないので、この方法が適しています。また、ほとんどのレートは頻繁に更新されます。ビジネスの性質によって、価格、利率のリスト、または最新ニュース項目といった内容を含むテーブルが考えられます。

❖ スナップショットベースの同期を実装するには、次の手順に従います。

1. リモート・ユーザが値を更新しない場合は、アップロード・スクリプトを未定義のままにしておきます。
2. テーブルのローを削除する場合は、リモート・テーブルのローをすべて削除する download_delete_cursor スクリプトを作成するか、少なくともすべてのローがもう必要ないことを定

義します。統合データベースからローを削除しないで、削除のマークを付けてください。ローの値を知らないと、リモート・データベースからローを削除できません。

詳細については、「download_delete_cursor スクリプトの作成」311 ページを参照してください。

3 download_cursor スクリプトを作成し、リモート・テーブルに登録するローをすべて選択します。

ローの削除

統合データベースからローを削除しないで、削除のマークを付けてください。ローの値を知らないと、リモート・データベースからローを削除できません。download_cursor スクリプトの場合はマークなしのローだけを、download_delete_cursor スクリプトの場合はマーク付きのローだけを選択します。

download_delete_cursor スクリプトは、download_cursor スクリプトより先に実行されます。ダウンロード・ストリームにローが含まれる場合は、同じプライマリ・キーを持つローを削除リスト内に含む必要はありません。ダウンロードしたローをリモート・ロケーションで取得するときに、同じプライマリ・キーを持つ既存のローは置き換えられます。

詳細については、「ローをダウンロードするスクリプトの作成」308 ページを参照してください。

別の削除方法

リモート・データベースからローを削除する場合、download_cursor スクリプトを使わなくても、リモート・アプリケーションを使ってローを削除できます。たとえば、同期のすぐ後に、アプリケーションで SQL 文を実行して不要なローを削除できます。

アプリケーションによって削除されたローは、通常、次回の同期で Mobile Link 同期サーバにアップロードされますが、STOP SYNCHRONIZATION DELETE 文を使って、アップロードしないようにできます。次に例を示します。

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

ローの削除の詳細については、「download_delete_cursor スクリプトの作成」311 ページを参照してください。

スナップショットの例

サンプル・アプリケーションの ULProduct テーブルは、スナップショットを使った同期によって管理されます。テーブルに入っているローの数は比較的少ないため、スナップショットを使った同期でのオーバヘッドがわずかです。

1. アップロード・スクリプトがありません。これは、製品情報はリモート・データベースでは追加できないという業務意思を反映しています。

2. download_delete_cursor スクリプトがないため、リストから製品を削除しないものと見なします。

3. download_cursor スクリプトによって、現在のすべての製品に関して、製品 ID、価格、製品名が選択されます。既存の製品の場合は、リモート・テーブル中のその製品の価格が更新されます。新しい製品の場合は、リモート・テーブルにローが挿入されます。

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

ローの数が極めて少ないテーブルでのスナップショットを使った同期の別の例については、『Mobile Link チュートリアル』> 「Contact サンプルの営業担当者の同期」を参照してください。

リモート・データベース間でローを分割する

各 Mobile Link リモート・データベースは、異なるデータのサブセットを統合データベース内に持つことができます。別の言い方をすれば、自分専用のスクリプトを作成して、リモート・データベース間でデータを「**分割**」できる、ということです。

共通部分がないように切断分割にすることも、重複を持たせて分割することもできます。たとえば、従業員ごとに独自の顧客セットを持っていて、かつ顧客を共有していない場合は、「**切断**」分割になります。複数のリモート・データベースに存在するように顧客を共有している場合、分割は「**重複**」を含みます。

分割は、テーブル用のスクリプトである download_cursor と download_delete_cursor で実行されます。これらのスクリプトによって、リモート・データベースにローをダウンロードするように定義します。各スクリプトは、パラメータとして Mobile Link ユーザ名を使用します。スクリプトでこのパラメータを WHERE 句に指定して、ユーザごとに適切なローを取得します。

切断分割

download_cursor スクリプトと download_delete_cursor スクリプトによって、同期で使用するテーブルごとに分割を制御します。このスクリプトでは、最後のダウンロードのタイムスタンプと、同期を呼び出すときに指定する Mobile Link ユーザ名という 2 つのパラメータを使用します。

❖ リモート・データベース間でテーブルを分割するには、次の手順に従います。

1. テーブル定義でカラムを指定し、そこに統合データベースの同期ユーザ名を持たせます。このカラムをリモート・データベースにダウンロードする必要はありません。
2. このカラムがスクリプトのパラメータと一致するように、download_cursor スクリプトと downloade_delete_cursor スクリプトの WHERE 句に条件文を指定します。

スクリプト中で疑問符 (?) が付いているところがパラメータです。ユーザ名は、download_cursor スクリプトの 2 番目のパラメータです。次の例では、download_cursor スクリプトによって、テーブル Contact を従業員 ID で分割します。

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

詳細については、『Mobile Link 管理ガイド』> 「download_cursor テーブル・イベント」および『Mobile Link 管理ガイド』> 「download_delete_cursor テーブル・イベント」を参照してください。

例

CustDB サンプル・アプリケーション内のプライマリ・キー・プール・テーブルを使って、リモート・データベースごとに独自のプライマリ・キー値のセットを指定します。この方法は、プライマリ・キーの重複を避けるために使用します。詳細については、「キー・プールを使用したユニークなプライマリ・キーの管理」76 ページを参照してください。

この方法では、プライマリ・キー・プール・テーブルがリモート・データベース間で切断分割されるようにしてください。

キー・プール・テーブル ULCustomerIDPool にあるプライマリ・キー値は、各ユーザが顧客を追加するときに使用します。ULCustomerIDPool テーブルには次の 3 つのカラムがあります。

- **pool_cust_id**　ULCustomer テーブルで使用するプライマリ・キー値。このカラムだけがリモート・データベースにダウンロードされます。
- **pool_emp_id**　このプライマリ・キーの所有者である従業員。
- **last_modified**　このテーブルは、last_modified カラムに基づいたタイムスタンプを使って管理されます。

  タイムスタンプの同期については、「タイムスタンプベースの同期」60 ページを参照してください。

このテーブルの download_cursor スクリプトを次に示します。

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

変数を使用しない場合は、プレースホルダの "**?**" を含んだ、ジョインまたはサブ選択を使用してください。

詳細については、『Mobile Link チュートリアル』> 「Contact サンプルの顧客の同期」および 『Mobile Link チュートリアル』> 「Contact サンプルの顧客窓口の同期」を参照してください。

重複のある分割

統合データベースの一部のテーブルは、複数のリモート・データベースに属するローを持ちます。各リモート・データベースは統合データベース内にローのサブセットを持ち、さらにそのサブセットは他のリモート・データベースと重複しています。顧客テーブルの場合は、こうしたことがよく起こります。この場合、顧客テーブルと複数のリモート・データベース間で多対多の関係があり、通常、この関係を表すテーブルが存在します。download_cursor イベントと download_delete_cursor イベントのスクリプトでは、関係を表すテーブルにダウンロードされるテーブルをジョインする必要があります。

例

CustDB サンプル・アプリケーションでは、この方法を ULOrder テーブルに使用します。ULEmpCust テーブル上での ULCustomer と ULEmployee の関係は、多対多の関係です。

各リモート・データベースは、ULOrder テーブルから、emp_id カラムの値が Mobile Link ユーザ名と一致するローのみを受信します。

CustDB アプリケーションの ULOrder テーブルで、Adaptive Server Anywhere バージョンの download_cursor スクリプトを使用した例を次に示します。

```
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 )
```

このスクリプトは非常に複雑です。スクリプトを見ると、リモート・データベースのテーブルを定義するクエリには、統合データベースのテーブルを複数指定できることがわかります。このスクリプトは、次のすべての条件に一致する ULOrder のローをすべてダウンロードします。

- ULOrder の cust_id カラムと ULEmpCust の cust_id カラムが一致する。
- ULEmpCust の emp_id カラムが同期ユーザ名と一致する。
- 注文、または従業員と顧客の関係に対する最終更新日がどちらも、同期ユーザ用の最新の同期時間よりも新しい。
- ステータスは "**Approved**" 以外である。

ULEmpCust のアクション・カラムを使用して、削除するカラムにマークを付けます。その目的は、現在説明している内容には関係ありません。

download_delete_cursor スクリプトを次に示します。

```
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 )
```

リモート・データベースから "Approved" のローがすべて削除されます。

子テーブルの分割

「重複のある分割」67 ページの例では、他のテーブルの基準に従ってテーブルを分割する方法を説明しています。

リモート・データベースの一部のテーブルには、切断のサブセットか重複したサブセットがありますが、サブセットを決定するカラムはありません。子テーブルには、通常、別のテーブルを参照する 1 つの外部キー (または一連の外部キー) があります。参照先のテーブルにはカラムがあり、これによって適切なサブセットが決定されます。

この場合、download_cursor スクリプトと download_delete_cursor スクリプトでは、参照先のテーブルをジョインしたり、WHERE 句を使用してローを適切なサブセットに制限したりする必要があります。

例については、『Mobile Link チュートリアル』> 「Contact サンプルの顧客窓口の同期」を参照してください。

ユニークなプライマリ・キーの管理

同期される各テーブルには、プライマリ・キーが必要です。また、プライマリ・キーは同期対象のすべてのデータベース間でユニークでなければなりません。プライマリ・キーの値は更新しないようにしてください。

多くの場合、テーブルのプライマリ・キーとして単一のカラムを使用すると便利です。たとえば、顧客にはそれぞれユニークな ID 値を割り当ててください。営業担当者全員がデータベース接続を直接管理できる環境で作業する場合は、ID 番号の割り当ては簡単に実施できます。顧客テーブルに新しい顧客が挿入されると、テーブルに最後に追加された値よりも大きな値を持つ新規のプライマリ・キーが自動的に追加されます。

接続が切断された環境では、新しいローの挿入時に、プライマリ・キーにユニークな値を割り当てるのは簡単ではありません。営業担当者が新しい顧客を追加する場合は、顧客テーブルのリモート・コピーにも同様な操作を行います。自分以外の営業担当者が、顧客テーブルの自分以外のコピーに対して操作を行っている場合、同じ顧客 ID 値を使わせないようにします。

この項では、ユニークなプライマリ・キーの生成に関する問題を解決する次の方法について説明します。

- ユニバーサル・ユニーク ID (UUID) の使用
- グローバル・オートインクリメント値の使用
- プライマリ・キー・プールの使用

UUID を使用したユニークなプライマリ・キーの管理

newid( ) 関数を使用してプライマリ・キーに対して完全にユニークな値を作成することによって、Adaptive Server Anywhere データベースのプライマリ・キーをユニークにすることができます。作成された UUID は、uuidtostr( ) 関数を使用して文字列に変換できます。また、strtouuid( ) 関数を使用してバイナリに戻すことができます。

UUID はすべてのコンピュータを通じてユニークです。ただし、この値は完全にランダムのため、値が追加された日時や値の順序を判別することはできません。また、UUID の値は他の方法 (グローバル・オートインクリメントを含む) で必要な値よりかなり大きく、プライマリ・キー・テーブルと外部キー・テーブルの両方でより多くのテーブル領域を必要とします。UUID を使用するテーブルのインデックスも効率性に劣ります。

詳細については、次の項目を参照してください。

- 『ASA SQL ユーザーズ・ガイド』> 「NEWID デフォルト」
- 『ASA SQL リファレンス・マニュアル』> 「NEWID 関数 [その他]」
- 『ASA SQL リファレンス・マニュアル』> 「UNIQUEIDENTIFIER データ型 [バイナリ]」

例

次の CREATE TABLE 文は、完全にユニークなプライマリ・キーを作成します。

```
CREATE TABLE customer (
    cust_key UNIQUEIDENTIFIER NOT NULL
             DEFAULT NEWID( ),
    rep_key VARCHAR(5),
    PRIMARY KEY(cust_key))
```

グローバル・オートインクリメントを使用したユニークなプライマリ・キーの管理

Adaptive Server Anywhere と Ultra Light のデータベースでは、デフォルトのカラム値を GLOBAL AUTOINCREMENT に設定できます。このデフォルト設定は、ユニークな値を管理するカラムのすべてに適用できますが、特にプライマリ・キーの場合に役立ちます。

❖ グローバル・オートインクリメント・カラムを使用するには、次の手順に従います。

1. カラムをグローバル・オートインクリメント・カラムとして宣言します。

このデフォルトのグローバル・オートインクリメントを指定すると、そのカラムの値のドメインが分割されます。各分割には同じ数の値が含まれます。たとえば、データベース内の整数カラムの分割サイズを 1000 に設定した場合、1 つの分割が 1001 から 2000 まで拡大します。また、2 つ目の分割は 2001 から 3000 まで拡大し、以降、同じように拡大していきます。

「デフォルトのグローバル・オートインクリメントの宣言」73 ページを参照してください。

2 GLOBAL_DATABASE_ID 値を設定します。

Adaptive Server Anywhere では、データベースのデフォルト値は、そのデータベース番号でユニークに識別された分割からのみ設定されます。たとえば、上の例のデータベースに ID 番号 10 を割り当てた場合、このデータベースのデフォルト値は 10001 ～ 11000 の範囲から選択されます。このデータベースの別のコピーで、ID 番号 11 が割り当てられたデータベースからは、11001 ～ 12000 の範囲にある同一カラムのデフォルト値が指定されます。

「GLOBAL_DATABASE_ID 値の設定」73 ページを参照してください。

関連項目

この項では、Adaptive Server Anywhere リモート・データベース内でグローバル・オートインクリメント・カラムを使用する方法について説明します。Ultra Light データベース内でグローバル・オートインクリメント・カラムを使用する方法については、『Mobile Link クライアント』> 「デフォルトのグローバル・オートインクリメント・カラムの宣言」を参照してください。

Adaptive Server Anywhere データベースにおけるグローバル・オートインクリメント・カラムの動作については、「デフォルト値の選択方法」74 ページを参照してください。

デフォルトのグローバル・オートインクリメントの宣言

Sybase Central でカラムのプロパティを選択するか、CREATE TABLE 文または ALTER TABLE 文に DEFAULT GLOBAL AUTOINCREMENT フレーズを組みこむことで、作業データベースにデフォルト値を設定できます。

オプションで、AUTOINCREMENT キーワードの直後にカッコで分割サイズを指定できます。この分割サイズには任意の正の整数を設定できますが、通常、分割サイズは、サイズの値がすべての分割で不足しないように選択されています。

INT または UNSIGNED INT 型のカラムの場合、デフォルトの分割サイズは 2^{16} = 65536 です。他の型のカラムの場合、デフォルトの分割サイズは 2^{32} = 4294967296 です。特にカラムが INT または BIGINT 型以外のときは、これらのデフォルトが適切ではない場合があるため、分割サイズを明示的に指定してください。

たとえば、次の文では 2 つのカラム (顧客 ID 番号を保持する整数カラム、顧客名を保持する文字列カラム) を持つ簡単なテーブルが作成されます。

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

上の例では、選択された分割サイズは 5000 です。

GLOBAL AUTOINCREMENT の詳細については、『ASA SQL リファレンス・マニュアル』> 「CREATE TABLE 文」を参照してください。

GLOBAL_DATABASE_ID 値の設定

アプリケーションを配備するときには、各データベースに対して必ず異なる ID 番号を割り当てます。ID 番号の作成と配布はさまざまな方法で実行できます。テーブルに値を設定し、ユーザ名など、ユニークなプロパティに基づいて、各データベースに適切なローをダウンロードするのも 1 つの方法です。

❖ **グローバル・データベース ID 番号を設定するには、次の手順に従います。**

- パブリック・オプション GLOBAL_DATABASE_ID の値を設定して、データベースの ID 番号を設定します。ID 番号は正の整数にします。

例

たとえば、次の文ではデータベースの ID 番号が 20 に設定されます。

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

特定カラムの分割サイズが 5000 の場合、このデータベースのデフォルト値は 100001 ～ 105000 の範囲から選択されます。

デフォルト値の選択方法

各データベース内のパブリック・オプション GLOBAL_DATABASE_ID は、ユニークな正の整数に設定してください。特定のデータベースのデフォルト値の範囲は、$pn + 1$ ～ $p(n + 1)$ です。ここで、p は分割サイズ、n はパブリック・オプション GLOBAL_DATABASE_ID の値を表します。たとえば、分割サイズを 1000、GLOBAL_DATABASE_ID を 3 に設定すると、範囲は 3001 ～ 4000 になります。

GLOBAL_DATABASE_ID が正の整数に設定されている場合、Adaptive Server Anywhere は次のルールを適用してデフォルト値を選択します。

- カラムに現在の分割の値が含まれていない場合、最初のデフォルト値は $pn + 1$ である。
- カラムに現在の分割の値が含まれていても、そのすべてが $p(n + 1)$ 未満であれば、この範囲内でこれまで使用した最大値より 1 大きい値が次のデフォルト値になる。
- デフォルトのカラム値は、現在の分割以外のカラムの値の影響を受けない。つまり、$pn+1$ より小さいか $p(n + 1)$ より大きい数には影響されない。Mobile Link 同期を介して別のデータベースからレプリケートされた場合に、このような値が存在する可能性があります。

public オプション GLOBAL_DATABASE_ID がデフォルト値の 2147483647 に設定されると、null 値がカラムに挿入されます。null 値が許可されていない場合に、ローの挿入を試みるとエラーが発生します。たとえば、テーブルのプライマリ・キーにカラムが含まれている場合に、この状況が発生します。

public オプション GLOBAL_DATABASE_ID は、負の値に設定できないため、選択された値は常に正になります。ID 番号の最大値を制限するのは、カラムのデータ型と分割サイズだけです。

デフォルトの Null 値は、分割で値が不足したときにも生成されます。この場合には、別の分割からデフォルト値を選択できるように、データベースに GLOBAL_DATABASE_ID の新しい値を割り当ててください。カラムで null が許可されていない場合、null 値を挿入しようとするとエラーが発生します。未使用の値が残り少ないことを検出し、このような状態を処理するには、**GlobalAutoincrement** タイプのイベントを作成します。

特定の分割で値が不足する場合は、新しいデータベース ID をそのデータベースに割り当てることができます。方法が適切なものであれば、新しいデータベース ID 番号を割り当てることができます。未使用のデータベース ID 値のプールを管理する方法も、その 1 つです。このプールは、プライマリ・キー・プールと同じ方法で管理されます。

分割で値が不足しそうな場合に、自動的にデータベース管理者へ通知する (またはその他のアクションを実行する) ようにイベント・ハンドラを設定できます。詳細については、『ASA データベース管理ガイド』> 「イベントのトリガ条件の定義」を参照してください。

詳細については、「GLOBAL_DATABASE_ID 値の設定」73 ページおよび『ASA データベース管理ガイド』> 「GLOBAL_DATABASE_ID オプション [データベース]」を参照してください。

キー・プールを使用したユニークなプライマリ・キーの管理

この問題を解決する効果的な方法の 1 つは、データベースの各ユーザに、必要に応じて、プライマリ・キー値のプールを割り当てることです。たとえば、営業担当者ごとに 100 個の新しい ID 値を割り当てます。各営業担当者は、自分のプール内の値を、新しい顧客に自由に割り当てることができます。

❖ プライマリ・キー・プールを実装するには、次の手順に従います。

1. 統合データベースと各リモート・データベースに新しいテーブルを追加して、新しいプライマリ・キー・プールを格納します。ユニークな値を格納するカラムとは別に、これらのテーブルにはユーザ名を格納するカラムが必要です。このユーザ名のカラムによって、値を割り当てる権限を持つ者を識別できます。

2. ストアド・プロシージャを作成し、十分な数の新しい ID 値が各ユーザに確実に割り当てられるようにします。新しいエントリを多数挿入する、または同期をあまり行わないリモート・ユーザには、特に多く、新しい値を割り当てます。

3. download_cursor スクリプトを作成して、各ユーザに割り当てられた新しい値を選択し、それをリモート・データベースにダウンロードします。

4. リモート・データベースを使用するアプリケーションを変更し、ユーザが新しいローを挿入するときに、プールに入っている値をアプリケーションが使用するようにします。アプリケーションは、その値をプールから削除して、値の再使用を防ぎます。

5. アップロード・スクリプトを作成します。ユーザがリモート・データベースの自分専用の値プールから削除した値と対応するローが、Mobile Link 同期サーバによって、統合データベースの値のプールから削除されます。

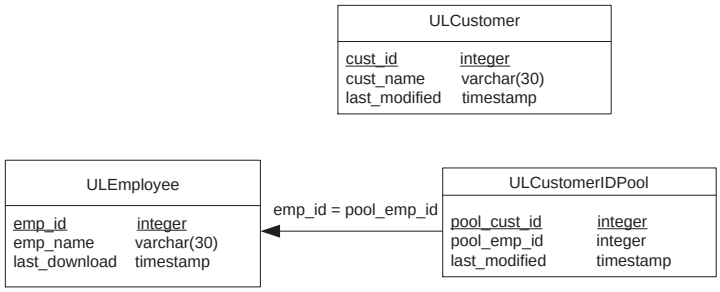
6. end_upload スクリプトを作成し、値のプールを管理するストアド・プロシージャを呼び出します。これで、ユーザのプールに対してさらに多くの値が追加され、削除済みの値がアップロード中に置き換わります。

例

リモート・ユーザは、サンプル・アプリケーションを使って顧客を追加できます。新しいローにはそれぞれユニークなプライマリ・キー値があることが必要です。ただし、データ・エントリ中は、まだ各リモート・データベースは切断された状態です。

ULCustomerIDPool にはプライマリ・キー値のリストがあり、この値を各リモート・データベースで使用できます。また、値を使い切ってしまうと、ULCustomerIDPool_maintain ストアド・プロシージャによってプール内の値が完全に補充されます。管理プロシージャは、テーブルレベルの end_upload スクリプトによって呼び出されます。各リモート・データベースのプールは、upload_cursor スクリプトと download_cursor スクリプトによって管理されます。

1. 統合データベースの ULCustomerIDPool テーブルには、新しい顧客 ID 番号のプールが格納されます。ULCustomer テーブルとは直接のリンク関係はありません。



2. ULCustomerIDPool_maintain プロシージャによって、統合データベースの ULCustomerIDPool テーブルが更新されます。Adaptive Server Anywhere 統合データベース用のサンプル・コードを次に示します。

```
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
```

このプロシージャは、現在のユーザに今割り当てられている番号をカウントします。また、新しいローを挿入して、このユーザが十分な数の顧客 ID 番号を使用できるようにします。

このプロシージャは、ULCustomerIDPool テーブル用の end_upload テーブル・スクリプトによって、アップロード・ストリームの最後に呼び出されます。スクリプトを次に示します。

```
CALL ULCustomerIDPool_maintain( ? )
```

3. ULCustomerIDPool テーブル用の download_cursor スクリプトは、リモート・データベースに新しい番号をダウンロードします。

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

4. 新しい顧客を挿入するには、リモート・データベースを使用しているアプリケーションで、プール中の未使用の ID 番号を選択して、その番号をプールから削除してから、この ID 番号による新しい顧客情報を挿入します。次に示す Ultra Light アプリケーション用の Embedded SQL 関数は、プールから新しい顧客番号を取り出します。

```
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;
}
```

競合の解決

警告
Mobile Link 環境では、プライマリ・キーを更新しないでください。プライマリ・キーは、競合を検出する目的でローをユニークに識別します。

競合は、統合データベースにローをアップロードしているときに発生します。異なるリモート・データベースで 2 人のユーザが同じローを修正した場合、Mobile Link 同期サーバに 2 つめのローが到着したときに競合が検出されます。競合が起こる可能性がある場合は、適切な値を計算するプロセスを定義するか、最低でも競合のログを取ってください。

同期のダウンロード処理中は、リモート・データベースでは競合は発生しません。ダウンロードしたローに新しいプライマリ・キーが含まれている場合は、その値は新しいローに挿入されます。新しいプライマリ・キーが既存のローのプライマリ・キーと一致する場合は、そのローの値が更新されます。

競合はエラーとは異なります。優れたアプリケーションを設計するには、競合の解決は不可欠です。

デフォルト動作

競合は、ローの更新中にのみ検出されます。競合とは、最後の同期以降に更新または削除されたローを更新しようとすることです。デフォルトでは、次の動作が行われます。

- ローを挿入しようとしたときに、そのローが挿入済みであることが検出されると、エラーが生成されます。
- ローを削除しようとしたときに、そのローが削除済みであることが検出されると、削除の試行は無視されます。

この処理は、利便性のために定義された組み込みの動作です。異なる動作を定義する場合は、1 つまたは複数のアップロード・イベントを使用して独自に実装してください。

同期の競合の対処方法については、『Mobile Link チュートリアル』>「チュートリアル：Mobile Link スクリプトの作成と同期のモニタリング」を参照してください。

競合の検出

Mobile Link クライアントが更新済みのローを Mobile Link 同期サーバに送信するときは、新しい値 (更新後イメージ) だけでなく、古いローの値 (更新前イメージ) のコピーも含まれています。更新前イメージが統合データベースの現在の値と一致しないと、競合が検出されます。

競合を検出するには、次の 2 つの方法があります。

- upload_fetch スクリプトを定義する
- WHERE 句に指定されたカラムがすべて非プライマリ・キー・カラムである upload_update スクリプトを定義する

同じテーブルにこれら両方のスクリプトを定義すると、upload_fetch スクリプトのみが使用されます。

upload_fetch または適切な upload_update スクリプトが適用されている場合にのみ、Mobile Link 同期サーバが競合を検出します。upload_fetch スクリプトを指定すると、Mobile Link 同期サーバは、アップロードの更新前イメージを、upload_fetch スクリプトから返される、同じプライマリ・キー値を持つローの値と比較します。更新前イメージの値が統合データベースの現在の値と一致しない場合、Mobile Link 同期サーバは競合を検出します。

upload_fetch

通常、upload_fetch スクリプトは、更新対象となるローに対応する統合データベースのテーブルから、データの単一のローを選択します。通常の upload_fetch スクリプトの構文は次のとおりです。

SELECT *col1*, *col2*, …
　FROM *table-name*
　WHERE *pk1* **= ? AND** *pk2* **= ?** …

詳細については、『Mobile Link 管理ガイド』> 「upload_fetch テーブル・イベント」を参照してください。

upload_update

upload_update スクリプトは、ローの各カラムのパラメータを提供します。

upload_update イベントのパラメータは、次の構文を持つ文でローが適切に更新されるように配置されています。

UPDATE *table-name*
  SET *col1* **= ?,** *col2* **= ?,** …
  WHERE *pk1* **= ? AND** *pk2* **= ?** …

この文では、col1 や col2 はプライマリ・キー・カラムではありませんが、pk1 や pk2 はプライマリ・キー・カラムです。

競合を検出するには、次の構文を使用します。

UPDATE *table-name*
  SET *col1* **= ?,** *col2* **= ?,** …
  WHERE *pk1* **= ? AND** *pk2* **= ?** …
  AND *col1* **= ? AND** *col2* **= ?** …

WHERE 句は、リモート・データベースから更新された古い値と、統合データベースの現在の値を比較します。これらの値が一致しないと更新は無視されるので、すでに統合データベースにあった値は保持されます。

詳細については、『Mobile Link 管理ガイド』> 「upload_update テーブル・イベント」を参照してください。

競合の解決

競合を解決するために、いくつかのオプションがあります。

- 競合が発生した場合に、テンポラリ・テーブルまたは永久テーブルと resolve_conflict スクリプトを使用して解決する。

  「resolve_conflict スクリプトによる競合の解決」83 ページを参照してください。

- 競合が発生した場合に、upload_update スクリプトを使用して解決する。

  「upload_update スクリプトによる競合の解決」86 ページを参照してください。

- テーブル用の end_upload イベントを使用して、すべての競合を一度に解決する。

『Mobile Link 管理ガイド』> 「end_upload テーブル・イベント」を参照してください。

文ベースのアップロードを使用した競合解決の例については、『Mobile Link チュートリアル』> 「Contact サンプルの製品の同期」を参照してください。

例

たとえば、User1 が最初に 10 個の在庫を売り出し、そのうち 3 個を販売して、Remote1 にある在庫の値を 7 個に更新します。User2 は 4 個販売し、Remote2 にある在庫を 6 に更新します。Remote1 が同期を実行すると、統合データベースは 7 に更新されます。Remote2 が同期を実行すると、在庫の値が 10 ではなくなっているため、競合が検出されます。この競合をプログラムで解決するには、次のような 3 つのロー値が必要となります。

1. 統合データベースにある現在の値。
2. Remote2 がアップロードした新しいローの値。
3. Remote2 が最後の同期中に取得した古いローの値。

この場合、ビジネス論理は新しい在庫値を計算し、競合を解決するために次の方法を使用します。

```
current consolidated - (old remote - new remote)
-> 7 - (10-6) = 3
```

resolve_conflict スクリプトによる競合の解決

Mobile Link 同期サーバが upload_fetch スクリプトを使用して競合を検出すると、次のイベントが発生します。

- Mobile Link 同期サーバは、リモート・データベースからアップロードされた古いロー値を upload_old_row_insert スクリプトの定義に従って挿入します。通常、古い値はテンポラリ・テーブルに挿入されます。

  詳細については、『Mobile Link 管理ガイド』> 「upload_old_row_insert テーブル・イベント」を参照してください。

- Mobile Link 同期サーバは、リモート・データベースからアップロードされた新しいロー値を upload_new_row_insert スクリプトの定義に従って挿入します。通常、新しい値はテンポラリ・テーブルに挿入されます。

  詳細については、『Mobile Link 管理ガイド』> 「upload_new_row_insert テーブル・イベント」を参照してください。

- Mobile Link 同期サーバは、resolve_conflict スクリプトを実行します。このスクリプトでは、ストアド・プロシージャを呼び出したり、実行手順の順序を定義したりすることによって、新しいロー値と古いロー値を使用して競合を解決できます。

詳細については、『Mobile Link 管理ガイド』> 「resolve_conflict テーブル・イベント」を参照してください。

例

次の例では、6 つのイベントに対してスクリプトを作成し、次にストアド・プロシージャを作成します。

- begin_synchronization スクリプトでは、contact_new および contact_old という 2 つのテンポラリ・テーブルを作成します。この操作は、begin_connection ではなく begin_synchronization で行います。これは、同期が失敗しても、対応する "end_" スクリプトが実行して、テンポラリ・テーブルを削除できるようにするためです。

- upload_old_row_insert スクリプトと upload_new_row_insert スクリプトでは、リモート・データベースからアップロードした新しいデータと古いデータを使用して、2 つのテンポラリ・テーブルを設定します。

- upload_fetch スクリプトは、競合を検出します。

- resolve_conflict スクリプトは、MLResolveConflict ストアド・プロシージャを呼び出して、競合を解決します。

イベント	スクリプト

begin_synchronization	`CREATE TABLE #contact_new(`
	`id  INTEGER,`
	`location CHAR(36),`
	`contact_date DATE);`
	`CREATE TABLE #contact_old(`
	`id  INTEGER,`
	`location CHAR(36),`
	`contact_date DATE)`

upload_fetch	`SELECT id, location, contact_date`
	`FROM contact`
	`WHERE id = ?`

upload_old_row_insert	`INSERT INTO #contact_new( id, location, contact_date )` `VALUES ( ?, ?, ? )`
upload_new_row_insert	`INSERT INTO #contact_old( id, location, contact_date )` `VALUES ( ?, ?, ? )`
resolve_conflict	`CALL MLResolveConflict( )`
end_synchronization	`DROP TABLE #contact_new;` `DROP TABLE #contact_old`

MLResolveConflict ストアド・プロシージャは次のとおりです。

```
CREATE PROCEDURE MLResolveConflict( )
 BEGIN
   --update the consolidated database only if the new
contact date
   --is later than the existing contact date
   UPDATE contact c
     SET c.contact_date = cn.contact_date
       FROM #contact_new cn
       WHERE c.id = cn.id
         AND cn.contact_date > c.contact_date;
   --cleanup
   DELETE FROM #contact_new;
   DELETE FROM #contact_old;
 END
```

upload_update スクリプトによる競合の解決

競合を解決するのに resolve_conflict スクリプトを使用する代わりに、upload_update スクリプトでストアド・プロシージャを呼び出すこともできます。この方法では、プログラムで競合の検出と解決の両方を行う必要があります。

たとえば、次のストアド・プロシージャ sp_update_customer には、競合を検出し解決するための論理が定義されています。この論理では、パラメータに新しいカラム値、プライマリ・キー、古いカラム値を許可しています。

```
call ml_add_table_script( 'version_1', 'customer',
'upload_update', '
 CALL sp_update_customer (?,?,?,?,?,?)
 )
 CREATE PROCEDURE sp_update_customer(
      IN @new_first_name    varchar(30)
     ,IN @new_last_name     varchar(30)
     ,IN @new_nullable_col  varchar(30)
     ,IN @new_last_modified timestamp
     ,IN @cust_1st_pk       integer
     ,IN @cust_2nd_pk       integer
     ,IN @old_first_name    varchar(30)
     ,IN @old_last_name     varchar(30)
     ,IN @old_nullable_col  varchar(30)
     ,IN @old_last_modified timestamp
 )
```

競合を検出するために、次の更新操作によって影響を受けるローの数をチェックできます。WHERE 句は、リモート・データベースから更新された古い値と、統合データベースの現在の値を比較します。値が一致すると、更新が成功し、競合は発生しません。

```
UPDATE customer
SET first_name                  = @new_first_name,
 last_name                         = @new_last_name,
 nullable_col                   = @new_nullable_col,
 last_modified                  = @new_last_modified

WHERE cust_1st_pk               = @cust_1st_pk
 AND cust_2nd_pk                = @cust_2nd_pk
 AND first_name                 = @old_first_name
```

```
  AND last_name                    = @old_last_name
  AND COALESCE(nullable_col, '') =
COALESCE(@old_nullable_col, '')
  AND last_modified                = @old_last_modified;
 ...
```

注意：
COALESCE 関数はリストから最初の非 NULL 式を返しますが、この場合は、NULL 入力可能なカラムの値を比較するために使用されます。

詳細については、『ASA SQL リファレンス・マニュアル』>「COALESCE 関数 [その他]」を参照してください。

統合データベースでメカニズムを使用することによって、上記で説明した更新文の結果を調べることができます。たとえば、Adaptive Server Anywhere を使用して、更新によって影響を受けたローの数を調べるために @@rowcount グローバル変数をチェックできます。@@rowcount = 0 の場合は競合が発生しているので、次に示すようにプログラムで競合を解決できます。この例では、最新の更新を実行したデータベースが競合に勝ちます。統合データベースが競合に勝った場合は、その現在の値が保持され、何のアクションも実行されません。

```
IF( @@rowcount = 0 ) THEN
 -- A conflict has been detected. To resolve, you:
 -- b) Use business logic to determine which values to
use
 -- c) Update the consolidated database with the final
values

  IF( @new_last_modified > @current_last_modified )
THEN
  -- The remote has won the conflict
  -- use the values it uploaded.

   UPDATE customer
    SET first_name    = @new_first_name,
        last_name     = @new_last_name,
        nullable_col  = @new_nullable_col,
        last_modified = @new_last_modified
```

```
            WHERE cust_1st_pk  = @cust_1st_pk
             AND cust_2nd_pk   = @cust_2nd_pk;

     END IF;
    END IF;
```

強制的な競合解決

強制的な競合解決は、アップロードされたすべてのローに競合があるものとして強制的に処理する特別な方法です。

upload_insert、upload_update、upload_delete スクリプトがすべて未定義の場合、Mobile Link 同期サーバは強制的な競合解決を使用します。この操作モードでは、Mobile Link 同期サーバは、そのテーブルからアップロードされたすべてのローを、upload_old_row_insert スクリプトと upload_new_row_insert スクリプトによって定義された文を使って挿入しようとします。基本的には、アップロードされたすべてのローは競合として処理されます。ストアド・プロシージャまたはスクリプトを作成し、アップロードした値を目的に応じて処理できます。

upload_insert、upload_update、または upload_delete スクリプトがない場合は、通常の競合解決処理は実行されません。この方法の主な使い方は、次の 2 とおりです。

- **任意の競合の検出と解決**　自動メカニズムでは、ローの更新時にエラーを検出するだけです。また、それは古い値が統合データベースの現在の値と一致しない場合にだけ行われます。

  upload_old_row_insert スクリプトと upload_new_row_insert スクリプトを使って、アップロードしたロー・データを取得し、そのデータが最適になるように処理できます。

- **パフォーマンス**　upload_insert、upload_update、upload_delete が定義されていない場合、Mobile Link 同期サーバが通常行う競合検出タスクは実行されません。このタスクには、1 回につき 1 つのローについての統合データベースへの問い合わせが含まれます。これらのスクリプトを定義しない場合は、upload_old_row_insert スクリプトと upload_new_row_insert スクリプトで定義した文を使って、アップロードしたばかりの情報の挿入だけは行ってください。挿入だけが関連するため、Mobile Link 同期サーバはより効率的に処理を行います。

ユーザ名の格納

upload_old_row_insert スクリプトまたは upload_new_row_insert スクリプトを作成するときに、INSERT 文の中に追加のパラメータを含めることができます。その場合、Mobile Link 同期サーバは、最初のパラメータにユーザ名を自動的に挿入し、その後、通常どおり残りのパラメータを使用します。データベース管理システムによっては、現在のユーザに関する識別情報を格納できないため、この便利なメカニズムが備わっています。

この機能を使って、各ローにどのユーザが挿入されたのかを簡単に識別できます。また、この機能は、同時に複数の異なるユーザからの新旧アップロード値を格納するために 1 つまたは 2 つのテーブルが使用されている場合に便利です。この情報により、resolve_conflict スクリプト中にユーザ固有のロジックを指定できます。

たとえば、通常の upload_old_row_insert スクリプトの形式は、次のとおりです。select リストの項目は、リモート・テーブルのカラムに対応しています。

```
INSERT INTO MyTable (c1, c2, . . . , cN) VALUES
(?,?,...,?)
```

ただし、次の構文も有効です。

```
INSERT INTO MyTableWithUser (user_name, c1, c2, . . . ,
cN) VALUES (?,?,...,?)
```

通常、select リストのカラムは、順序と型の両方をリモート・テーブルのカラムに一致させますが、この場合は例外です。値リストにパラメータを 1 つ追加する場合は、たとえば VARCHAR(128) などのように、ユーザ名の格納に適した型にしてください。値リスト内のユーザ名の後のパラメータは、通常のように順序と型の両方をリモート・テーブルのカラムに一致させます。1 つでも余分なパラメータがある場合は、エラーになります。

詳細については、『Mobile Link 管理ガイド』> 「upload_old_row_insert テーブル・イベント」および 『Mobile Link 管理ガイド』> 「upload_new_row_insert テーブル・イベント」を参照してください。

データ・エントリ

一部のデータベースには、データ・エントリ専用のテーブルがあります。この種類のテーブルを処理するには、同期時に挿入されたローをすべてアップロードし、そのローをダウンロード・ストリームでリモート・データベースから削除する方法があります。同期後に、リモート・テーブルは再び空になり、別のデータ・バッチに使用できます。

このモデルを使用するには、end_upload テーブル・スクリプトを使用して、ローをテンポラリ・テーブルにアップロードしてから、ベース・テーブルに挿入してください。テンポラリ・テーブルを download_delete_cursor スクリプトの中で使用し、同期が正常に完了した後で、リモート・データベースからローを削除できます。

また、STOP SYNCHRONIZATION DELETE 文 (削除内容を次回の同期中にアップロードしないようにする) を使って、クライアント・アプリケーションでもローを削除できます。

詳細については、『ASA SQL リファレンス・マニュアル』> 「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」を参照してください。

削除の処理

ローを統合データベースから削除する場合、そのローを持つすべてのリモート・データベースからも削除できるように、ローの記録が必要です。

1 つの方法として、ローそのものを削除しない方法があります。不要になったデータについては、ローのステータス・カラムを変更して、"inactive (非アクティブ)" のマークを付けることができます。これを論理削除と言い、download_cursor と download_delete_cursor の WHERE 句で、ローのステータスを参照できます。この方法は、CustDB サンプル・アプリケーションの ULEmpCust テーブルで使用されており、テーブルのアクション・カラムには削除を示す "D" が入っています。スクリプトでは、この値を使用してリモート・データベースからレコードを削除し、さらに、同期処理の最後に統合データベースからもレコードを削除します。CustDB ではこの方法を ULOrder テーブルに使用し、Contact サンプルではこの方法を Customer、Contact、Product テーブルに使用しています。

2 つ目の方法は、シャドー・テーブルを作成し、削除したローのプライマリ・キー値をそこに格納することです。ローが削除されると、1 つのトリガによってシャドー・テーブルに移植されます。download_delete_cursor スクリプトは、シャドー・テーブルを使用して、リモート・データベースからローを削除します。シャドー・テーブルには、実際のテーブルのプライマリ・キー・カラムだけが必要です。

詳細については、「download_delete_cursor スクリプトの作成」311 ページを参照してください。

失敗したダウンロードの処理

ダウンロード・トランザクションで、ダウンロード内容の書き換え情報を保持します。この情報は、リモート・データベースに適用されたダウンロードの内容を基に、自動的に更新されます。

エラーが発生してダウンロード・ストリーム全体をリモート・データベースに適用できない場合、SendDownloadAck を ON に変更しても、Mobile Link 同期サーバはダウンロードの確認を取得できないため、ダウンロード・トランザクションをロールバックします。トランケーション・ポイントの書き換え情報は、ダウンロード・トランザクションの一部であるため、ロールバックされます。次回ダウンロード・ストリームが作成される場合、元のトランケーション・ポイントの書き換え情報を使用します。

詳細については、『Mobile Link クライアント』> 「SendDownloadACK (sa) 拡張オプション」または 『Mobile Link クライアント』> 「Send Download Acknowledgement 同期パラメータ」を参照してください。

同期スクリプトをテストした時にエラーが発生した場合は、end_download スクリプトを論理に追加してください。これで、失敗に終わったダウンロードを、スクリプトを使って確実に処理できます。

失敗したダウンロードの再開

ダウンロードの失敗は、ダウンロード中の通信エラーまたはユーザによるダウンロードのアボートによって発生します。Mobile Link は、ダウンロードの失敗からのリカバリを支援する機能を備えています。この機能を使用すると、ダウンロード全体の再送を防ぐこともできます。この機能は、Adaptive Server Anywhere と Ultra Light リモート・データベースでそれぞれ別に実装されています。

Adaptive Server Anywhere リモート・データベース

ダウンロード中に同期が失敗すると、いずれのダウンロードもリモート・データベースには適用されません。ただし、正常に送信されたダウンロードの部分は、リモート・デバイスのテンポラリ・ファイルに格納されます。このファイルには直接アクセスできませんが、dbmlsync はこのファイルを利用するための機能を提供しています。この機能を使用すると、長時間のデータ再送を防ぐことができます。また、自動的にダウンロードの失敗をリカバリすることも可能です。

注意
SendDownloadACK 拡張オプションが ON (デフォルトは OFF) に設定されている場合、または DownloadBufferSize 拡張オプションが 0 (これもデフォルトではありません) に設定されている場合、ダウンロードを再開することはできません。

この機能を実装するには 3 つの方法があります。どの方法の場合も、アップロードする新しいデータが存在する場合は再開されたダウンロードは失敗し、dbmlsync はアボートされます。

- **-dc**　ダウンロードが失敗した後、次回 dbmlsync を起動するときに、-dc を使用してダウンロードを再開します。失敗したダウンロードの一部が送信されている場合、Mobile Link 同期サーバは、ダウンロードの残りだけを送信します。

  詳細については、『Mobile Link クライアント』> 「-dc オプション」を参照してください。

- **ContinueDownload (cd) 拡張オプション**　dbmlsync コマンド・ラインで使用した場合、cd 拡張オプションは -dc オプションと同じように動作します。このオプションは、データベース内に格納したり、sp_hook_dbmlsync_set_extended_options を使用して 1 つの同期内に設定することもできます。

  詳細については、『Mobile Link クライアント』> 「ContinueDownload (cd) 拡張オプション」および 『Mobile Link クライアント』> 「sp_hook_dbmlsync_set_extended_options」を参照してください。

- **sp_hook_dbmlsync_end hook**　**restart** パラメータを使用して、ダウンロードを再開できます。**restartable download** パラメータが true に設定されている場合は、ダウンロードを再開できます。ダウンロード・ファイルが存在し、一定のサイズの場合は、フック内に論理を作成してダウンロードを再開することもできます。

  詳細については、『Mobile Link クライアント』> 「sp_hook_dbmlsync_end」を参照してください。

Ultra Light リモート・データベース

次に示すように、ダウンロードが失敗した後の Ultra Light アプリケーションの動作を制御できます。

- 同期時に Keep Partial Download 同期パラメータを true に設定している場合、ダウンロードが完了する前に失敗すると、Ultra Light はダウンロードされた変更の部分を適用します。また Ultra Light は、Partial Download Retained パラメータを true に設定します。

  この時点では、Ultra Light データベースは一貫性のない状態になります。アプリケーションの仕様に応じて、同期を正常に完了するか、データの変更を許可する前にロールバックしてください。

  詳細については、『Mobile Link クライアント』> 「Keep Partial Download 同期パラメータ」および 『Mobile Link クライアント』> 「Partial Download Retained 同期パラメータ」を参照してください。

- ダウンロードを再起動するには、Resume Partial Download 同期パラメータを true に設定し、再度同期を実行します。

  詳細については、『Mobile Link クライアント』> 「Resume Partial Download 同期パラメータ」を参照してください。

  再起動可能な同期はアップロードを実行せず、失敗したダウンロードによってダウンロードされた変更のみをダウンロードします。つまり、失敗したダウンロードは完了しますが、前回の実行以降に行われた変更は同期しません。これらの変更を取得するには、一度失敗したダウンロードが完了してから、再度同期を実行する必要があります。または、Rollback Partial Download を呼び出して、false に設定されている Resume Partial Download と同期します。

  ダウンロードを再起動すると、失敗した同期から多数の同期パラメータがもう一度自動的に使用されます。たとえば、パブリケーション・パラメータは無視されます。同期は、最初のダウンロードで要求されたパブリケーションをダウンロードします。唯一設定が必要なパラメータは、Resume Partial Download パラメータ (true に設定) と User Name パラメータです。また、以下のパラメータ (設定されている場合) の設定も有効です。

  - Keep Partial Download (今後の中断に備える)

- DisableConcurrency
- Observr
- User Data

- 失敗したダウンロードからの変更を、同期を再開せずにロールバックするには、変更をロールバックする関数を呼び出します。この関数は、embedded SQL の ULRollbackPartialDownload 関数です。Ultra Light コンポーネントの場合は、Connection オブジェクトのメソッドです。

  - **MobileVB**『Ultra Light for MobileVB ユーザーズ・ガイド』> 「RollbackPartialDownload メソッド」

  - **Ultra Light.NET**『Ultra Light.NET ユーザーズ・ガイド』> 「RollbackPartialDownload メソッド」(iAnywhere.Data.UltraLite ネームスペース) および『Ultra Light.NET ユーザーズ・ガイド』> 「RollbackPartialDownload メソッド」(iAnywhere.UltraLite ネームスペース) を参照してください。

  - **Native Ultra Light for Java**「API リファレンス」の **ianywhere.native_ultralite.Connection.rollbackPartialDownload** メソッド

  - **Ultra Light C++ コンポーネント**『Ultra Light C/C++ ユーザーズ・ガイド』> 「RollbackPartialDownload メソッド」

  - **Ultra Light ActiveX**『Ultra Light ActiveX ユーザーズ・ガイド』> 「RollbackPartialDownload メソッド」

  - **Embedded SQL**『Ultra Light C/C++ ユーザーズ・ガイド』> 「ULRollbackPartialDownload 関数」

  - **静的型 C++ API**『Ultra Light C/C++ ユーザーズ・ガイド』> 「RollbackPartialDownload メソッド」

  - **静的型 Java API** この機能は静的型 Java API には使用できません。

サーバやネットワークが使用できないなどの理由で同期が完了できない場合や、エンド・ユーザがアプリケーションで作業を実行中もデータの一貫性を維持したい場合に、失敗したダウンロードから変更をロールバックできます。

通信エラーの詳細については、『ASA エラー・メッセージ』> 「ASA エラー・メッセージ」を参照してください。

注意

send_download_ack 同期パラメータが true (デフォルトではありません) に設定されている場合、再開されたダウンロードでは設定は無視されます。

ストアド・プロシージャ・コールからの結果セットのダウンロード

ストアド・プロシージャ・コールから結果セットをダウンロードできます。たとえば、次のテーブルに対する download_cursor があるとします。

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

download_cursor テーブル・スクリプトは次のようになります。

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

MyTable へのダウンロードに、より高度なビジネス論理を使用する場合は、次のようにスクリプトを作成できます。DownloadMyTable は、2 つのパラメータ (最終ダウンロード・タイムスタンプと Mobile Link ユーザ名) を取り、結果セットを返すストアド・プロシージャです (この例では、移植性のため ODBC 呼び出し規則を使用しています)。

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

次に、サポートされる統合データベースごとの簡単な例を示します。詳細については、使用している統合データベースのマニュアルを参照してください。

次の例は、Adaptive Server Anywhere、Adaptive Server Enterprise、Microsoft SQL Server に適用されます。

```
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
```

次の例は、Oracle に適用されます。Oracle では、パッケージが定義されている必要があります。定義するパッケージには、結果セットのレコード・タイプと、レコード・タイプを返すカーソル・タイプが含まれていなければなりません。

注意

この例では、Oracle が結果セットを返す必要があります。[ODBC Oracle ドライバ設定] ダイアログで [Procedure Returns Results] オプションを選択する必要があります。または、接続文字列で ProcedureRetResults=1 と設定する必要があります。Oracle ODBC ドライバの設定の詳細については、『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』> 「Oracle 対応の iAnywhere Solutions ODBC ドライバ」を参照してください。

```
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;
```

次に、Oracle では、パッケージのカーソル・タイプが最初のパラメータに指定されているストアド・プロシージャが必要です。download_cursor スクリプトは、3 つではなく 2 つのパラメータのみを渡すことに注意してください。Oracle で結果セットを返すストアド・プロシージャの場合、ストアド・プロシージャ定義でパラメータとして宣言されているカーソル・タイプによって結果セットの構造が定義されますが、真のパラメータ自体は定義されません。

```
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;
```

次の例は、IBM DB2 UDB に適用されます。

```
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
```

次に示すのは、ストアド・プロシージャを DLMyTable.java に Java で実装した例です。結果セットを返すには、メソッドが戻るときに、結果セットを開いたままにします。

```
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 + "'" );
    }
 }
```

リモート・データベースでのスキーマの変更

要件の変化に応じて、配備したリモート・データベースのスキーマを変更しなければならない場合があります。最も一般的なスキーマの変更とは、新しいカラムを既存のテーブルに追加する、または新しいテーブルをデータベースに追加することです。

Adaptive Server Anywhere リモート・データベース

Adaptive Server Anywhere リモート・データベースを配備した後に、そのスキーマを変更することができます。

❖ Adaptive Server Anywhere リモート・データベースにテーブルを追加するには、次の手順に従います。

1. 関連するテーブル・スクリプトを統合データベースに追加します。

   新しいテーブルのないリモート・データベースと、新しいテーブルのあるリモート・データベースには、同じスクリプト・バージョンを使用できます。ただし、新しいテーブルが存在することによって既存のテーブルの同期方法が変更される場合は、新しいスクリプト・バージョンを作成し、そのスクリプト・バージョンで同期されるすべてのテーブルに対して新しいスクリプトを作成する必要があります。

2. 通常の同期を実行します。

   この手順は必須ではありませんが、スキーマを変更する前に実行しておくことをおすすめします。

3. ALTER PUBLICATION 文を使用して、テーブルを追加します。次に例を示します。

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

   詳細については、『ASA SQL リファレンス・マニュアル』>「ALTER PUBLICATION 文」を参照してください。

4　同期を実行します。必要な場合は、新しいスクリプト・バージョンを使用します。

リモート・データベースのテーブル定義の変更

既存テーブルのカラムの数か型を変更する場合は、注意してください。Mobile Link クライアントが新しいスキーマで同期する場合、upload_update や download_cursor などのスクリプトが必要です。これらのスクリプトには、リモート・テーブルの全カラム用のパラメータがあります。古いリモート・データベースの場合は、元のカラムだけを保持するスクリプトが必要です。

❖ 配備された Adaptive Server Anywhere リモート・データベースのパブリッシュ済みのテーブルを変更するには、次の手順に従います。

1　統合データベースで、新しいスクリプト・バージョンを作成します。

詳細については、「スクリプト・バージョン」298 ページを参照してください。

2　新しいスクリプト・バージョンには、古いスクリプト・バージョンで同期された変更対象のテーブルを含むパブリケーションのすべてのテーブルに対してスクリプトを作成します。

3　リモート・データベースで、古いスクリプト・バージョンを使用して通常の同期を実行します。

4　リモート・データベースで、ALTER PUBLICATION 文を使用して、テーブルをパブリケーションから一時的に削除します。次に例を示します。

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

詳細については、『ASA SQL リファレンス・マニュアル』>「ALTER PUBLICATION 文」を参照してください。

5　リモート・データベースで、ALTER TABLE 文を使用してテーブルを変更します。

詳細については、『ASA SQL リファレンス・マニュアル』> 「ALTER TABLE 文」を参照してください。

6 リモート・データベースで、ALTER PUBLICATION 文を使用して、テーブルをパブリケーションに戻します。

詳細については、『ASA SQL リファレンス・マニュアル』> 「ALTER PUBLICATION 文」を参照してください。

7 新しいスクリプト・バージョンを使用して同期します。

*注意：*手順 4 ～ 6 は、sp_hook_dbmlsync_schema_upgrade ストアド・プロシージャで実行することも可能です。詳細については、『Mobile Link クライアント』> 「sp_hook_dbmlsync_schema_upgrade」を参照してください。

Adaptive Server Anywhere リモート・データベースのスキーマの変更方法については、『Mobile Link クライアント』> 「sp_hook_dbmlsync_schema_upgrade」を参照してください。

Ultra Light リモート・データベース

Ultra Light リモート・データベースのスキーマは、新しいアプリケーションを配備するか、スキーマをアップグレードすることで変更できます。

- スキーマをアップグレードせずに新しいアプリケーションを配備する場合は、Mobile Link 同期サーバとの同期を行って Ultra Light データベースにデータを再移植する必要があります。
- スキーマをアップグレードする場合、データは保持されます。全ユーザが同時にアプリケーションを新しいバージョンにアップグレードすることは、通常は難しいからです。

新旧 2 つのバージョンを使用できるようにして、単一の統合データベースとこれらを同期する必要があります。2 種類以上の同期スクリプトを作成し、それらを統合データベースに格納して、Mobile Link 同期サーバの動作を制御できます。次に、使用するアプリケーションのバージョンごとに、同期の開始時に正しいバージョン名を指定することによって、適切な同期スクリプトのセットを選択できます。

Ultra Light のスキーマの詳細については、『Ultra Light データベース・ユーザーズ・ガイド』> 「Ultra Light データベースとスキーマの作成」を参照してください。

第 5 章

ファイルベースのダウンロード

この章の内容

この章では、Adaptive Server Anywhere リモート・データベースにデータをダウンロードする、もう 1 つの方法について説明します。ダウンロードの内容はファイルとして配布でき、同期の変更をオフラインで配布可能です。このため、ファイルを一度作成すれば、多数のリモート・データベースにこのファイルを配布できます。

この章では、ファイルベースのダウンロードの使用方法について詳しく説明し、その設定方法を示す 2 つの詳細な例を示します。

概要

ファイルベースのダウンロードでは、ダウンロードした同期の変更内容をファイルに保存し、ファイルを転送可能なあらゆる方法を使用して Adaptive Server Anywhere リモート・データベースに転送できます。次に例を示します。

- 衛星マルチキャストでデータをブロードキャストする。
- Manage Anywhere Studio を使用して更新を適用する。
- ファイルを電子メールまたは FTP でユーザに送信する。

ファイルを送信するユーザを選択します。ファイルベースのダウンロードでは、競合の検出と解決を含め、同期の整合性は完全に保護されます。このファイルにサードパーティの暗号化を適用することにより、ファイルの安全性を保証できます。

使用する場合

ファイルベースのダウンロードは、統合データベースで大量のデータが変更されたときには便利ですが、リモート・データベースではデータの更新の頻度が低いか、更新がまったく行われません。たとえば、価格リスト、製品リスト、コードのテーブルなどです。

ファイルベースのダウンロードは、ダウンロードされたデータがリモート・データベースで頻繁に更新される場合、またはアップロード専用の同期を頻繁に実行している場合には適していません。このような状況では、ダウンロード・ファイルの適用時に実行される整合性のチェックが原因で、リモート・サイトはダウンロード・ファイルを適用できないことがあります。

リモート・データベースを更新する手段として、ファイルベースのダウンロードのみを使用することはできません。完全な同期またはアップロード専用の同期を定期的に実行する必要もあります。完全な同期またはアップロード専用の同期は、ログ・オフセットを進めたり、ログ・ファイルを保守したりするために必要です。こうしないと、ログ・ファイルのサイズが大きくなり、同期処理に時間がかかるようになります。また、完全な同期ではエラーからのリカバリが必要になることもあります。

現時点では、ファイルベースのダウンロードは Adaptive Server Anywhere リモート・データベースでのみ使用可能です。

ファイルベースのダウンロードの設定

ファイルベースのダウンロードを設定するには、次の手順に従います。

1. ファイル定義データベースを作成します。
2. 統合データベースで、新しいスクリプト・バージョンを使用してスクリプトを作成します。
3. ダウンロード・ファイルを作成します。
4. ダウンロード・ファイルを適用します。

上記の手順について、以下に詳しく説明します。

ファイル定義データベースの作成

ファイルベースのダウンロードを設定するには、「**ファイル定義データベース**」を作成します。これは、リモート・データベースと同じ同期テーブルと同期パブリケーションを持つ Adaptive Server Anywhere データベースです。配置する場所は、どこでもかまいません。このデータベースには、データまたはステータスの情報はありません。バックアップまたは保守を行う必要がなく、必要に応じて削除したり、再作成できます。

ファイル定義データベースには、次のものが含まれている必要があります。

- リモート・データベースと同じパブリケーション、そのパブリケーションで使用されるテーブルとカラム、これらのテーブルとカラムの外部キー関係と制約、これらの外部キー関係に必要なテーブル。
- ダウンロード・ファイルを適用するリモート・データベースのグループを識別する Mobile Link ユーザ名。リモート・データベースのグループを識別するには、このグループの Mobile Link ユーザ名を同期スクリプトで使用します。

統合データベースでの変更

統合データベースでは、ファイルベースのダウンロード用のスクリプト・バージョンを新規に作成し、既存の同期システムに必要なスクリプトを実装します。アップロード・スクリプトは必要ありません。このスクリプト・バージョンは、ファイルベースのダウンロードにのみ使用されます。このスクリプト・バージョンの場合、パラメータとして Mobile Link ユーザ名を利用するすべてのスクリプトは、リモート・データベースのグループを示す Mobile Link ユーザ名をパラメータとして利用します。これは、ファイル定義データベースで定義されているユーザ名です。

定義した各スクリプト・バージョンには、begin_publication スクリプトを実装します。

タイムスタンプベースのダウンロードの場合は、各スクリプト・バージョンに modify_last_download_timestamp スクリプトを実装します。このスクリプトの実装方法は、各ダウンロード・ファイルで送信するデータ量によって異なります。たとえば、グループのユーザによる前回の正常なダウンロードの時刻の中で最も早いものを使用する方法があります。このスクリプトに渡される ml_username パラメータは、実際にはグループ名です。

詳細については、「modify_last_download_timestamp 接続イベント」527 ページを参照してください。

ダウンロード・ファイルの作成

ダウンロード・ファイルには、同期されるデータが格納されています。ダウンロード・ファイルを作成するには、上記の説明のようにファイル定義データベースと統合データベースを設定します。-bc オプションを使用し、拡張子が .df のファイル名を指定して dbmlsync を実行します。次に例を示します。

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

必要に応じて、ダウンロード・ファイルの作成時にオプションを指定できます。

- **-be オプション** -be オプションを使用すると、sp_hook_dbmlsync_validate_download_file ストアド・プロシージャを使用してリモート・データベースでアクセス可能なダウンロード・ファイルに文字列を追加できます。

  詳細については、『Mobile Link クライアント』> 「-be オプション」および 『Mobile Link クライアント』> 「sp_hook_dbmlsync_validate_download_file」を参照してください。

- **-bg オプション** -bg オプションを使用すると、一度も同期されていないリモート・データベースによって使用可能なダウンロード・ファイルを作成できます。

新しいリモートの同期

Mobile Link を使用して同期されたことのないリモート・データベースにダウンロード・ファイルを適用する場合は、リモート・データベースで通常の同期を実行してからダウンロード・ファイルを適用するか、ダウンロード・ファイルの作成時に dbmlsync -bg オプションを使用する必要があります。

タイムスタンプベースの同期の場合は、この 2 つの方法のいずれかを行うと、データの初期のスナップショットがダウンロードされます。タイムスタンプベースとスナップショットベースの両方の同期では、この手順によって、統合データベースの begin_publication スクリプトが生成する値に世代番号が設定されます。

通常の同期の実行

ダウンロード・ファイルを使用しない同期を実行することによって、ダウンロード・ファイルを受信するためのリモート・データベースを準備します。

-bg オプションの使用

別の方法として、まだ同期されていないリモート・データベースで使用するために、-bg オプションを使用してダウンロード・ファイルを作成できます。この初期ダウンロード・ファイルを適用して、ファイルベースの同期に使用されるリモート・データベースを準備します。

- **スナップショットのダウンロード** スナップショットのダウンロードを実行している場合は、初期ダウンロード・ファイルに世代番号を設定する必要があります。このファイルにデータの初期スナップショットを含めることは可能ですが、各スナップショットのダウンロードにはすべてのデータが含まれ、前のダウンロードに依存しないため必須ではありません。

  スナップショットのダウンロードは、-bg オプションを使用すると簡単です。ダウンロード・ファイルを作成するときに、dbmlsync コマンドで -bg を指定するだけです。同じスクリプト・バージョンを使用して、以降のダウンロード・ファイルに使用する初期ダウンロード・ファイルを作成できます。

- **タイムスタンプベースのダウンロード** タイムスタンプベースのダウンロードを実行している場合は、初期ダウンロードでリモート・データベースの世代番号を設定し、データのスナップショットを含める必要があります。タイムスタンプベースのダウンロードでは、各ダウンロードは前のダウンロードに基づいています。ダウンロード・ファイルには、それぞれ最終ダウンロード・タイムスタンプが格納されています。ファイルの最終ダウンロード・タイムスタンプの後に統合データベースで変更されたローは、すべてこのファイルに格納されています。ファイルを適用するには、ファイルの最終ダウンロード・タイムスタンプの前に発生したすべての変更をリモート・データベースが受信している必要があります。この確認は、ファイルの最終ダウンロード・タイムスタンプが、リモート・データベースの最終ダウンロード・タイムスタンプ (リモート・データベースが統合データベースからすべての変更を受信するまでの時刻) 以降であることをチェックして行われます。

  リモート・データベースでは、最初の通常のダウンロード・ファイルを適用する前に、ファイルの最終ダウンロード・タイムスタンプより前に変更され、しかも 1900 年 1 月 1 日より後に変更されたすべてのデータを受信している必要があります。このデータを選択する最も簡単な方法は、通常のファイルベースの同期スクリプト・バージョンと同じ download_cursor を使用していても、modify_last_download_timestamp スクリプトは含まない別のスクリプト・バージョンを作成することです。no modify_last_download_timestamp スクリプトが定義されていな

い場合、ファイルベースのダウンロードの最終ダウンロード・タイムスタンプは、デフォルトで 1900 年 1 月 1 日に設定されます。

-bg オプションを使用して構築されたダウンロード・ファイルを同期済みのリモート・データベースに適用すると、この -bg オプションにより、ダウンロード・ファイルが作成されたときの統合データベースの値を使用してリモート・データベースで世代番号が更新されます。このため、世代番号は無効になります。世代番号は、消失または破損した統合データベースをリカバリする場合にアップロードが実行されるまで、ファイルベースのダウンロードをそれ以上適用しないようにするためのものです。

世代番号の詳細については、「Mobile Link の世代番号」115 ページを参照してください。

検証チェック

dbmlsync は、同期が有効であることを確認するためにいくつかの処理を行ってから、ダウンロード・ファイルをリモート・データベースに適用します。

- dbmlsync は、ダウンロード・ファイルの作成に使用されたファイル定義データベースに次のものが含まれていることを確認するため、このダウンロード・ファイルをチェックします。
  - リモート・データベースと同じパブリケーション
  - そのパブリケーションで使用される同じテーブルとカラム
  - それらのテーブルとカラムと同じ外部キー関係
- dbmlsync は、リモート・データベースからアップロードされていないデータがパブリケーションに存在するかどうかをチェックします。データが存在する場合は、ダウンロード・ファイルは適用されません。これは、ダウンロード・ファイルを適用すると、保留中のアップロードが消失することがあるためです。
- dbmlsync は、最終ダウンロード・タイムスタンプ、次の最終ダウンロード・タイムスタンプ、ダウンロード・ファイルの作成時刻をチェックして、次のことを確認します。
  - リモート・データベースの新しいデータが、ダウンロード・ファイルに含まれる古いデータで上書きされないこと。
  - ダウンロード・ファイルを適用すると、統合データベースで発生した変更の一部をリモート・データベースが取得しない場合には、ダウンロード・ファイルを適用しないこと。この状況は、リモート・データベースが前のファイルベースのダウンロードを適用しなかった場合に発生することがあります。

    詳細については、「自動検証」113 ページを参照してください。
- オプションで、dbmlsync はダウンロード・ファイルの世代番号と一致することを確認するために、リモート・データベースで世代番号をチェックします。

詳細については、「Mobile Link の世代番号」115 ページを参照してください。

- オプションで、sp_hook_dbmlsync_validate_download_file ストアド・プロシージャを使用して、カスタムの検証論理を作成できます。

  詳細については、「カスタム検証」116 ページを参照してください。

自動検証

dbmlsync は、最終ダウンロード・タイムスタンプ、次の最終ダウンロード・タイムスタンプ、ダウンロード・ファイルの作成時刻、トランザクション・ログに特別なチェックを実行してから、ダウンロード・ファイルを適用します。

最終ダウンロード・タイムスタンプと次の最終ダウンロード・タイムスタンプ

各ダウンロード・ファイルには、ファイルの最終ダウンロード・タイムスタンプから次の最終ダウンロード・タイムスタンプまでの間に統合データベースで発生したダウンロード対象のすべての変更が格納されています。この時刻は、統合データベースの時刻です。デフォルトでは、ファイルの最終ダウンロード時刻は 1900 年 1 月 1 日 12:00 AM で、ファイルの次の最終ダウンロード・タイムスタンプはダウンロード・ファイルが作成された時刻です。これらのデフォルト値を上書きするには、modify_last_download_timestamp スクリプトと modify_next_last_download_timestamp スクリプトを統合データベースに実装します。

リモート・サイトは、ファイルの最終ダウンロード・タイムスタンプが、リモートの最終ダウンロード・タイムスタンプ以前である場合にのみ、ダウンロード・ファイルを適用できます。これにより、リモートは統合データベースで発生した操作を失うことはありません。通常、このチェックに基づいたファイルベースのダウンロードが失敗した場合、リモート・サイトは 1 つまたは複数のダウンロード・ファイルを失っていることになります。この状況を修正するには、取得しなかったダウンロード・ファイルを適用するか、完全な同期またはダウンロード専用の同期を実行します。

さらに、リモート・サイトは、次のファイルの最終ダウンロード・タイムスタンプが、リモートの最終ダウンロード・タイムスタンプよりも後である場合にのみ、ダウンロード・ファイルを適用できます。リ

モートの最終ダウンロード・タイムスタンプは、ダウンロード対象のすべての変更をリモートが受信するまでの時刻 (統合データベースでの時刻) です。リモート・データベースの最終ダウンロード時刻は、通常またはファイルベースのダウンロードをリモートが正常に適用するたびに更新されます。このチェックを行うことにより、より新しいデータがすでにダウンロードされている場合はダウンロード・ファイルが適用されることはありません。一般的には、これが発生するのは、ダウンロード・ファイルが正常に適用されなかった場合です。たとえば、ダウンロード・ファイル F1.df が作成され、別のファイル F2.df が後で作成されたとします。このチェック機能により、F2.df の後に F1.df が適用されることはありません。これは、F2.df の後に F1.df が適用されると、F2.df の新しいデータが、F1.df の古いデータで上書きされてしまうためです。

次の最終ダウンロード・タイムスタンプに基づいたファイルベースのダウンロードが失敗した場合、このファイルを削除する以外に必要な作業はありません。新しいファイルを受信すると、同期は成功します。

作成時刻

ダウンロード・ファイルの作成時刻は、ファイルの作成が開始された時点の統合データベースでの時刻を示しています。ダウンロード・ファイルを適用できるのは、ファイルの作成時刻が、リモート・データベースの最終アップロード時刻よりも後の場合だけです。リモートの最終アップロード時刻は、リモートの正常な最終アップロードがコミットされた時点の統合データベースでの時刻です。このチェックにより、ダウンロードの作成後にアップロードされた (ダウンロードよりも新しい) データは、ダウンロード・ファイルの古いデータで上書きされることはありません。

このチェックに基づいてダウンロード・ファイルが拒否されても、必要な作業はありません。リモート・サイトは、次のダウンロード・ファイルの適用が可能になっている必要があります。

dbmlsync がアップロードを Mobile Link 同期サーバに送信しても、確認を取得できなかったためにアップロードが失敗した場合は、リモート・データベースの最終アップロード時刻が不正になることがあります。この場合、作成時刻のチェックを実行できません。また、リモート・データベースは通常の同期を完了するまでダウンロード・ファイルを適用できません。

トランザクション・ログ

dbmlsync は、リモート・データベースのトランザクション・ログをスキャンし、アップロードする必要があるすべての変更のリストを構築してから、ダウンロード・ファイルを適用します。dbmlsync がダウンロード・ファイルを適用するのは、アップロードが必要な変更のあるローに影響する操作がダウンロード・ファイルに含まれていない場合だけです。

Mobile Link の世代番号

世代番号とは、リモート・データベースがデータをアップロードしてからダウンロード・ファイルを適用するようにするためのメカニズムです。これは、統合データベースで問題が発生したためにデータが失われ、そのデータをリモート・データベースからリカバリする必要があるときに、特に役立ちます。

リモート・データベースでは、各サブスクリプションに対して別々の世代番号が自動的に管理されています。統合データベースでは、各サブスクリプションの世代番号は begin_publication スクリプトによって決定されます。リモート・データベースが正常にアップロードを行うたびに、リモート・データベースの世代番号は統合データベースの begin_publication スクリプトによって設定された値で更新されます。

ダウンロード・ファイルが作成されるたびに、begin_publication スクリプトによって設定された世代番号がダウンロード・ファイルに格納されます。リモート・サイトは、ダウンロード・ファイルの世代番号がリモート・データベースに格納されている世代番号と同じ場合にのみ、ダウンロード・ファイルを適用します。

注意

begin_publication スクリプトによってファイルベースのダウンロード用に生成された世代番号が変更された場合、リモート・データベースは正常なアップロードを実行してから、新しいダウンロード・ファイルを適用する必要があります。

---

sp_hook_dbmlsync_validate_download_file ストアド・プロシージャを使用すると、デフォルトで行われる世代番号のチェックを無効にすることができます。

Mobile Link の世代番号の管理については、次の項目を参照してください。

- 「begin_publication 接続イベント」443 ページ
- 「end_publication 接続イベント」494 ページ
- 『Mobile Link クライアント』>「sp_hook_dbmlsync_validate_download_file」

カスタム検証

カスタムの検証論理を作成すると、ダウンロード・ファイルをリモート・データベースに適用する必要があるかどうかを判断できます。これには、sp_hook_dbmlsync_validate_download_file ストアド・プロシージャを使用します。このストアド・プロシージャを使用すると、ダウンロード・ファイルを拒否し、デフォルトで行われる世代番号のチェックを無効にすることができます。

dbmlsync -be オプションを使用すると、文字列をファイルに埋め込むことができます。-be オプションは、ダウンロード・ファイルを作成するときに、ファイル定義データベースに対して使用します。この文字列は、#hook_dict テーブルを介して sp_hook_dbmlsync_validate_download_file に渡され、検証論理で使用できます。

詳細については、『Mobile Link クライアント』>「sp_hook_dbmlsync_validate_download_file」を参照してください。

例

この項には、非常に簡単な 2 つの例が用意されています。それぞれの例では、統合データベースと 1 つのテーブルのみを使用して、ファイルベースのダウンロードの同期を設定します。1 番目はスナップショットの例で、2 番目はタイムスタンプベースの例です。

スナップショットの例

この例では、スナップショット同期のファイルベースのダウンロードを実行します。最初に、ファイルベースのダウンロードで必要な 3 つのデータベースを設定し、次に、データをダウンロードする方法を示します。この例は、参考にするだけでもかまいませんし、テキストをカット・アンド・ペーストしてサンプルを実行することもできます。

サンプル用のデータベースの作成

次のコマンドは、この例で使用される統合データベース、リモート・データベース、ファイル定義データベースの 3 つのデータベースを作成します。

```
dbinit scons.db
 dbinit sremote.db
 dbinit sfdef.db
```

次のコマンドは、この 3 つのデータベースを起動し、統合データベースへの接続に使用する Mobile Link のデータ・ソース名を作成して、Mobile Link 同期サーバを起動します。

```
 dbeng9 -n sfdef_eng sfdef.db
 dbeng9 -n scons_eng scons.db
 dbeng9 -n sremote_eng sremote.db
 dbdsn -y -w fbd_demo -c
"eng=scons_eng;dbf=scons.db;uid=dba;
    pwd=sql;astart=off;astop=off"
 start dbmlsrv9 -v+ -c "dsn=fbd_demo"
    -zu+ -ot scons.txt
```

スナップショットの例で使用する統合データベースの設定

この例では、統合データベースには T1 というテーブルが 1 つあります。統合データベースに接続すると、次の SQL を実行してテーブル T1 を作成できます。

```
CREATE TABLE T1 (
     pk    INTEGER PRIMARY KEY,
     c1    INTEGER
 );
```

次のコードは、filebased というスクリプト・バージョンを作成し、そのスクリプト・バージョンのダウンロード・スクリプトを作成します。

```
CALL ml_add_table_script( 'filebased',
   'T1', 'download_cursor',
      'SELECT pk, c1 FROM T1' );
```

次のコードは、normal というスクリプト・バージョンを作成し、そのスクリプト・バージョンのアップロード・スクリプトとダウンロード・スクリプトを作成します。

```
CALL ml_add_table_script ( 'normal', 'T1',
   'upload_insert',
   'INSERT INTO T1 VALUES (?,?)');


CALL ml_add_table_script( 'normal', 'T1',
   'upload_update',
   'UPDATE T1 SET c1 = ? WHERE pk = ? ' );

 CALL ml_add_table_script( 'normal', 'T1',
  'upload_delete',
     'DELETE FROM T1 WHERE pk = ?' );

 CALL ml_add_table_script( 'normal', 'T1',
   'download_cursor',
      'SELECT pk, c1 FROM T1' );

 COMMIT;
```

次のコマンドは、ストアド・プロシージャ begin_pub を作成し、begin_pub が、"normal" スクリプト・バージョンと "filebased" スクリプト・バージョンの両方を対象とした begin_publication スクリプトであることを指定します。

```
CREATE PROCEDURE begin_pub (
       INOUT generation_num integer,
             IN    username       varchar(128),
             IN    pubname        varchar(128) )
BEGIN
   SET gnum=1;
END;

CALL ml_add_connection_script(
  'filebased',
  'begin_publication',
        '{ call begin_pub( ?, ?, ? ) }' );

CALL ml_add_connection_script( 'normal',
  'begin_publication',
        '{ call begin_pub( ?, ?, ? ) }' );
```

スナップショット例で使用するリモート・データベースの作成

この例では、リモート・データベースにも T1 というテーブルが 1 つあります。リモート・データベースに接続し、次の SQL コマンドを実行して、テーブル T1、パブリケーション P1、ユーザ U1 を作成します。また、この SQL は P1 に対する U1 のサブスクリプションも作成します。

```
CREATE TABLE T1 (
     pk    INTEGER PRIMARY KEY,
     c1    INTEGER
);

CREATE PUBLICATION P1 (
    TABLE T1
);



CREATE SYNCHRONIZATION USER U1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR U1;
```

次のコードは、sp_hook_dbmlsync_validate_download_file フックを作成して、ユーザ定義の検証論理をリモート・データベースに実装します。

```
 CREATE PROCEDURE
sp_hook_dbmlsync_validate_download_file()
 BEGIN
     DECLARE udata    varchar(256);
 SELECT value
     INTO udata
     FROM #hook_dict
     WHERE name = 'user data';


     IF udata <> 'ok' THEN
     UPDATE #hook_dict
        SET value = 'FALSE'
        WHERE name = 'apply file';
     END IF;
 END
```

スナップショット例で使用するファイル定義データベースの作成

ファイルベースのダウンロードを使用する Mobile Link システムには、ファイル定義データベースが必要です。このデータベースのスキーマはファイルベースのダウンロードで更新されるリモート・データベースのスキーマと同じですが、データとステータス情報は格納されていません。ファイル定義データベースは、ダウンロード・ファイルに格納されるデータの構造を定義するためだけに使用します。リモート・データベースの Mobile Link グループのユーザ名で定義された、多数のグループのリモート・データベースに対して、1 つのファイル定義データベースを使用できます。

次のコードは、この例で使用するファイル定義データベースを定義します。このコードはリモート・データベースと同じスキーマを作成し、さらに以下を作成します。

- P1 という名前のパブリケーション。T1 テーブルのすべてのローをパブリッシュします。ファイル定義データベースとリモート・データベースでは、同じパブリケーション名を使用する必要があります。
- G1 という名前の Mobile Link ユーザ。このユーザは、ファイルベースのダウンロードで更新されるすべてのリモート・データベースを表しています。
- パブリケーションに対するサブスクリプション

sfdef.db に接続してから、次のコードを実行してください。

```
CREATE TABLE T1 (
     pk    INTEGER PRIMARY KEY,
     c1    INTEGER
 );



 CREATE PUBLICATION P1 (
     TABLE T1
 );

 CREATE SYNCHRONIZATION USER G1;



 CREATE SYNCHRONIZATION SUBSCRIPTION
 TO P1
 FOR G1;
```

初期同期の準備

ダウンロード・ファイルを適用できるようにするために新しいリモート・データベースを準備するには、通常の同期を実行するか、dbmlsync -bg オプションを使用してダウンロード・ファイルを作成します。この例は、通常の同期を実行して新しいリモート・データベースを初期化する方法を示しています。

リモート・データベースの初期同期は、以前に作成した normal というスクリプト・バージョンで実行できます。

```
dbmlsync -c "uid=dba;pwd=sql;eng=sremote_eng;
   dbf=sremote.db" -v+ -e "sv=normal"
```

スナップショット例におけるファイルベースのダウンロードの実行

統合データベースに接続し、ファイルベースのダウンロードで同期される次のようなデータをいくつか挿入します。

```
INSERT INTO T1 VALUES( 1, 1 );
INSERT INTO T1 VALUES( 2, 4 );
INSERT INTO T1 VALUES( 3, 9 );
COMMIT;
```

次のコマンドは、ファイル定義データベースのあるコンピュータで実行してください。次の処理が行われます。

- dbmlsync -bc オプションにより、ダウンロード・ファイルが作成され、file1.df という名前が付けられます。
- -be オプションにより、"OK" という文字列がダウンロード・ファイルに追加され、sp_dbmlsync_validate_download_file フックへのアクセスが可能になります。

```
dbmlsync -c
    "uid=dba;pwd=sql;eng=sfdef_eng;dbf=sfdef.db"
    -v+ -e "sv=filebased" -bc file1.df -be ok -ot
fdef.txt
```

ダウンロード・ファイルを適用するには、-ba オプションと、適用するダウンロード・ファイルの名前を指定して、リモート・データベースで dbmlsync を実行します。

```
dbmlsync -c "uid=dba;pwd=sql;eng=sremote_eng;
    dbf=sremote.db" -v+ -ba file1.df -ot remote.txt
```

これで、リモート・データベースに変更が適用されました。Interactive SQL を開いてリモート・データベースに接続し、次に示す SQL コマンドを実行して、リモート・データベースにデータがあることを確認します。

```
SELECT * FROM T1
```

スナップショット例のクリーンアップ

次のコマンドは、3 つのデータベース・エンジンをすべて停止し、ファイルを消去します。

```
del file1.df
 dbmlstop -h -w
 dbstop -y -c eng=sfdef_eng
 dbstop -y -c eng=scons_eng
 dbstop -y -c eng=sremote_eng
 dberase -y sfdef.db
 dberase -y scons.db
 dberase -y sremote.db
```

タイムスタンプベースの例

この例では、タイムスタンプベースの同期のファイルベースのダウンロードを実行します。最初に、ファイルベースのダウンロードで必要な 3 つのデータベースを設定し、次に、この機能を使ってデータをダウンロードする方法を示します。この例は、参考にするだけでもかまいませんし、テキストをカット・アンド・ペーストしてサンプルを実行することもできます。

サンプル用のデータベースの作成

次のコマンドは、この例で使用される統合データベース、リモート・データベース、ファイル定義データベースの 3 つのデータベースを作成します。

```
dbinit tcons.db
 dbinit tremote.db
 dbinit tfdef.db
```

次のコマンドは、この 3 つのデータベースを起動し、統合データベースへの接続に使用する Mobile Link のデータ・ソース名を作成して、Mobile Link 同期サーバを起動します。

```
 dbeng9 -n tfdef_eng tfdef.db
 dbeng9 -n tcons_eng tcons.db
 dbeng9 -n tremote_eng tremote.db
 dbdsn -y -w tfbd_demo -c
"eng=tcons_eng;dbf=tcons.db;uid=dba;
    pwd=sql;astart=off;astop=off"
 start dbmlsrv9 -v+ -c "dsn=tfbd_demo" -zu+ -ot
tcons.txt
```

タイムスタンプの例で使用する統合データベースの設定

この例では、統合データベースには T1 というテーブルが 1 つあります。統合データベースに接続すると、次のコードを実行してテーブル T1 を作成できます。

```
CREATE TABLE T1 (
     pk        INTEGER PRIMARY KEY,
     c1        INTEGER,
     last_modified    TIMESTAMP DEFAULT TIMESTAMP
 );
```

次のコードは、最小限の数のスクリプトで構成される normal というスクリプト・バージョンを定義します。このスクリプト・バージョンは、ファイルベースのダウンロードを*使用しない*同期に使用されます。

```
CALL ml_add_table_script( 'normal', 'T1',
    'upload_insert',
       'INSERT INTO T1( pk, c1) VALUES( ?, ? )' );

 CALL ml_add_table_script( 'normal', 'T1',
    'upload_update',
    'UPDATE T1 SET c1 = ? WHERE pk = ? ' );

 CALL ml_add_table_script( 'normal', 'T1',
    'upload_delete',
       'DELETE FROM T1 WHERE pk = ?' );

 CALL ml_add_table_script( 'normal', 'T1',
    'download_cursor',
     'SELECT pk, c1 FROM T1 WHERE last_modified >= ?' );
```

次のコードは、すべてのサブスクリプションの世代番号を 1 に設定します。世代番号は、統合データベースが消失または破損し、アップロードが必要となった場合に使用すると便利です。

```
CREATE PROCEDURE begin_pub (
        INOUT generation_num integer,
              IN    username       varchar(128),
              IN    pubname        varchar(128) )
 BEGIN
     SET generation_num = 1;
 END;

 CALL ml_add_connection_script( 'normal',
     'begin_publication',
                 '{ call begin_pub( ?, ?, ? ) }' );

 COMMIT;
```

次のコードは、filebased というスクリプト・バージョンを定義します。このスクリプト・バージョンは、ファイルベースのダウンロードの作成に使用されます。

```
CALL ml_add_connection_script( 'filebased',
     'begin_publication',
                '{ call begin_pub( ?, ?, ? ) }' );

 CALL ml_add_table_script( 'filebased', 'T1',
     'download_cursor',
     'SELECT pk, c1 FROM T1 WHERE last_modified >= ?' );
```

次のコードは、最後の 5 日間に発生したすべての変更がダウンロード・ファイルに追加されるように最終ダウンロード時刻を設定します。最後の 5 日間に作成されたすべてのダウンロード・ファイルを取得していないリモートは、通常の同期を実行しないと、ファイルベースのダウンロードを適用できません。

```
CREATE PROCEDURE ModifyLastDownloadTimestamp(
               INOUT last_download_timestamp TIMESTAMP,
                 IN    ml_username
VARCHAR(128) )
 BEGIN
     SELECT dateadd( day, -5, CURRENT TIMESTAMP )
     INTO last_download_timestamp;
 END;


 CALL ml_add_connection_script( 'filebased',
    'modify_last_download_timestamp',
          'CALL ModifyLastDownloadTimestamp( ?, ? )' );

 COMMIT;
```

タイムスタンプベースの同期で使用するリモート・データベースの作成

この例では、リモート・データベースにも T1 というテーブルが 1 つあります。リモート・データベースに接続した後、次の SQL コマンドを実行して、テーブル T1、パブリケーション P1、ユーザ U1 を作成します。また、このコードは P1 に対する U1 のサブスクリプションも作成します。

```
CREATE TABLE T1 (
     pk    INTEGER PRIMARY KEY,
     c1    INTEGER
 );

 CREATE PUBLICATION P1 (
     TABLE T1
 );
```

```
CREATE SYNCHRONIZATION USER U1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR U1;
```

次のコードは、sp_hook_dbmlsync_validate_download_file ストアド・プロシージャを定義します。このストアド・プロシージャは、文字列 "ok" が埋め込まれていないダウンロード・ファイルの適用を防止します。

```
CREATE PROCEDURE
sp_hook_dbmlsync_validate_download_file()
 BEGIN
     DECLARE udata    varchar(256);

     SELECT value
     INTO udata
     FROM #hook_dict
     WHERE name = 'user data';

     IF udata <> 'ok' THEN
       UPDATE #hook_dict
          SET value = 'FALSE'
          WHERE name = 'apply file';
     END IF;
 END
```

タイムスタンプベースの同期で使用するファイル定義データベースの作成

次のコードは、タイムスタンプベースの同期で使用するファイル定義データベースを定義します。また、このコードは、テーブル、パブリケーション、ユーザ、そのパブリケーションに対するユーザのサブスクリプションを作成します。

```
CREATE TABLE T1 (
     pk    INTEGER PRIMARY KEY,
     c1    INTEGER
 );

 CREATE PUBLICATION P1 (
     TABLE T1
 );
```

```
CREATE SYNCHRONIZATION USER G1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR G1;
```

初期同期の準備

ダウンロード・ファイルを適用できるようにするために新しいリモート・データベースを準備するには、通常の同期を実行するか、dbmlsync -bg オプションを使用してダウンロード・ファイルを作成します。この例では、-bg の使用方法を示します。

次のコードは、統合データベースの filebased_init というスクリプト・バージョンを定義します。このスクリプト・バージョンには、1 つの begin_publication スクリプトがあります。

```
CALL ml_add_table_script(
   'filebased_init', 'T1', 'download_cursor',
      'SELECT pk, c1 FROM T1' );

CALL ml_add_connection_script(
   'filebased_init',
   'begin_publication',
         '{ call begin_pub( ?, ?, ? ) }' );

COMMIT;
```

次のコマンド・ラインでは、filebased_init というスクリプト・バージョンと -bg オプションを使用して初期ダウンロード・ファイルを作成、適用します。

```
dbmlsync -c
"uid=dba;pwd=sql;eng=tfdef_eng;dbf=tfdef.db"
   -v+ -e "sv=filebased_init" -bc tfile1.df -be ok -bg
   -ot tfdef1.txt

dbmlsync -c
"uid=dba;pwd=sql;eng=tremote_eng;dbf=tremote.db"
   -v+ -ba tfile1.df -ot tremote.txt
```

タイムスタンプベースの同期のファイルベースのダウンロードを実行する

統合データベースに接続し、ファイルベースのダウンロードで同期される次のようなデータをいくつか挿入します。

```
INSERT INTO T1(pk, c1) VALUES( 1, 1 );
 INSERT INTO T1(pk, c1) VALUES( 2, 4 );
 INSERT INTO T1(pk, c1) VALUES( 3, 9 );
 commit;
```

次のコマンド・ラインは、新しいデータを含むダウンロード・ファイルを作成します。

```
dbmlsync -c
    "uid=dba;pwd=sql;eng=tfdef_eng;dbf=tfdef.db"
    -v+ -e "sv=filebased" -bc tfile2.df -be ok -ot
tfdef2.txt
```

次のコマンド・ラインは、ダウンロード・ファイルをリモート・データベースに適用します。

```
dbmlsync -c
"uid=dba;pwd=sql;eng=tremote_eng;dbf=tremote.db"
    -v+ -ba tfile2.df -ot tfdef3.txt
```

これで、リモート・データベースに変更が適用されました。Interactive SQL を開いてリモート・データベースに接続し、次に示す SQL コマンドを実行して、リモート・データベースにデータがあることを確認します。

```
SELECT * FROM T1
```

タイムスタンプベースの同期のクリーンアップ

次のコマンドは、3 つのデータベース・エンジンをすべて停止し、ファイルを消去します。

```
del file1.df
 dbmlstop -h -w
 dbstop -y -c eng=tfdef_eng
 dbstop -y -c eng=tcons_eng
 dbstop -y -c eng=tremote_eng
 dberase -y tfdef.db
 dberase -y tcons.db
 dberase -y tremote.db
```

第 6 章

Mobile Link のパフォーマンス

この章の内容

この章では、Mobile Link 同期のパフォーマンスを向上する方法について説明します。

Mobile Link のパフォーマンスの詳細については、http://my.sybase.com/detail?id=1009664 にある *Mobile Link のパフォーマンス*に関するホワイトペーパー (英文) を参照してください。

パフォーマンスの監視方法を説明したチュートリアルについては、『Mobile Link チュートリアル』> 「チュートリアル : Mobile Link スクリプトの作成と同期のモニタリング」を参照してください。

パフォーマンスに関するヒント

Mobile Link から最高のパフォーマンスを導き出すための提案を以下に示します。

- **テスト**　配備する前に、運用環境で使用するのと同じハードウェアとネットワークを使用して容量のテストを実行します。このテストで、以下に示すパフォーマンスのヒントを試します。

- **競合の回避**　同期スクリプトでは競合を回避します。別の方法としては、同時実行性を最大化します。

  たとえば、begin_download スクリプトが、テーブル内のカラムを増分してダウンロードの合計数をカウントするとします。複数のユーザが同時に同期を行う場合、このスクリプトによって効果的にダウンロードが直列化されます。begin_synchronization または end_synchronization スクリプトはコミットの直前に呼び出されるため、これらのスクリプトの場合も同じカウンタが有効です。

  競合の詳細については、「競合」136 ページを参照してください。

  同期のトランザクション構造については、「同期処理のトランザクション」23 ページを参照してください。

- **最適な数のワーカ・スレッドの使用**　Mobile Link の −w オプションを使用して、Mobile Link ワーカ・スレッド数を、最適なスループットが得られる最小値に設定します。使用状況に合った最適な数を見つけるには、実験が必要です。

  ワーカ・スレッドの数が多いほど、多くの同期を同時に実行してスループットが向上します。

  ワーカ・スレッドの数を小さい値にしておくと、統合データベースでの競合発生回数、統合データベースへの接続数、最適なキャッシュに必要なメモリを少なくすることができます。

  たとえば、処理速度の速いクライアントでのテストでは、ワーカ・スレッドは 5 つ程度で最適なスループットが得られました。処理速度の遅いクライアントで、ダウンロードのスループットを最大にするには、より多くのワーカ・スレッドが必要です。また、-wu オプションを使って、同時にアップロードできる数

を制限すると、アップロードの最適スループットが得られます。極端に処理速度の遅いクライアントを使ったテストでは、ワーカ・スレッドを数百にして、同時アップロードを 5 つだけにすると、アップロードとダウンロードで最高のスループットを得られました。これらの数は特定のテスト・セットに基づいています。配備によって特徴が異なるので、-w と -wu の最適な値を判別するためにテストしてください。

ワーカ・スレッドの詳細については、「ワーカ・スレッド数」137 ページを参照してください。

詳細については、『Mobile Link 管理ガイド』> 「-w オプション」および『Mobile Link 管理ガイド』> 「-wu オプション」を参照してください。

- **ASA クライアントの場合、クライアント側のダウンロード・バッファを有効にする**　Adaptive Server Anywhere クライアントの場合、ダウンロード・バッファにより、クライアントがダウンロードを適用するのを待たずに、Mobile Link ワーカ・スレッドがダウンロードを転送できます。ダウンロード・バッファはデフォルトで有効になっています。ただし、ダウンロード確認が有効になっている場合は、ダウンロード・バッファを使用できません (次の項目を参照)。

  ダウンロード・バッファのサイズ設定の詳細については、『Mobile Link クライアント』> 「DownloadBufferSize (dbs) 拡張オプション」を参照してください。

- **ASA クライアントのダウンロード確認は、有効にしないでください。** デフォルトでは、ダウンロード確認は無効になっています。これは、Mobile Link ワーカ・スレッドを解放します。ダウンロード確認を無効にしておかないと、クライアントからのダウンロード成功の確認を待機することになります。また、無効にしておけば、ワーカ・スレッドが使用している接続を解放することにもなります。さらに、Mobile Link 同期サーバによるダウンロードのバッファも可能になります。

  Ultra Light クライアントはダウンロードをバッファしないため、ダウンロードの確認を無効にする利点はあまりありません。

ダウンロード確認の詳細については、『Mobile Link クライアント』> 「SendDownloadACK (sa) 拡張オプション」を参照してください。

- **アップロード・キャッシュ・サイズの設定**　ディスクへのアップロード・キャッシュのオーバフローを避けるため、アップロード・キャッシュのサイズを、最大アップロード・ストリームのサイズとワーカ・スレッド数の積より大きく設定します。アップロード・キャッシュ・サイズは dbmlsrv9 -u オプションで設定します。

  詳細については、『Mobile Link 管理ガイド』> 「-u オプション」を参照してください。

- **ダウンロード・キャッシュ・サイズの設定**　ディスクへのダウンロード・バッファのオーバフローを避けるため、ダウンロード・キャッシュのサイズを、最大ダウンロードのサイズとワーカ・スレッド数の積よりも大きく設定します。ダウンロード・キャッシュのサイズは dbmlsrv9 -d オプションで設定します。

  ダウンロード・バッファに割り付けるメモリの設定の詳細については、『Mobile Link 管理ガイド』> 「-d オプション」を参照してください。

- **BLOB キャッシュ・サイズの設定**　ローに LONG VARCHAR 型または LONG BINARY 型のデータがある場合は、ローの最大 BLOB データとワーカ・スレッド数の積の 2 倍より大きく BLOB キャッシュのサイズを設定すると、BLOB キャッシュからディスクへのアクセスを避けることができます。BLOB キャッシュのサイズは dbmlsrv9 -bc オプションで設定します。

  詳細については、『Mobile Link 管理ガイド』> 「-bc オプション」を参照してください。

- **データベース接続の最大数の設定**　Mobile Link データベース接続の最大数を、同期スクリプト・バージョンの通常の数と Mobile Link ワーカ・スレッド数の積に 1 を加えた数に設定します。これにより、Mobile Link がデータベース接続を閉じたり作成したりする必要が少なくなります。接続の最大数は dbmlsrv9 -cn オプションで設定します。

詳細については、「Mobile Link データベース接続」139 ページおよび『Mobile Link 管理ガイド』> 「-cn オプション」を参照してください。

- **十分な物理メモリの確保**　Mobile Link を実行しているコンピュータに、他のメモリ要件に加えて、アップロード、ダウンロード、BLOB の各キャッシュを確保する十分な物理メモリがあることを確認してください。

- **十分な処理能力の使用**　十分な処理能力を Mobile Link に確保して、Mobile Link サーバの処理がボトルネックにならないようにしてください。Adaptive Server Anywhere 統合データベースとの比較テストでは、Adaptive Server Anywhere と Mobile Link の両方に負荷をかけ、同じコンピュータで実行した場合、Mobile Link が必要とする処理能力は Adaptive Server Anywhere の 1/3 ～ 1/2 でした。

- **最小の冗長ロギングの使用**　ビジネス・ニーズに合う最小の冗長ロギングを使用してください。デフォルトでは、冗長ロギングは設定されておらず、Mobile Link はディスクにログを書き込みません。ロギングの冗長性を制御するには、-v オプションを使用します。また、-o オプションか -ot オプションを使用すると、ファイルへのロギングを有効にできます。

  冗長ログ・ファイルの代わりに、Mobile Link モニタで同期をモニタできます。Mobile Link モニタは Mobile Link 同期サーバと同じコンピュータ上に存在しなくてもかまいません。また、モニタ接続が Mobile Link サーバのパフォーマンスに与える影響はほとんどありません。詳細については、「Mobile Link モニタ」143 ページを参照してください。

- **Java または .NET の同期論理と SQL 同期論理**　Java または .NET の同期論理と、SQL 同期論理では、スループットに著しい差は見られません。ただし、Java と .NET の同期論理では、同期ごとに余分なオーバーヘッドが発生し、より多くのメモリが必要です。

さらに、SQL 同期論理は統合データベースが稼動しているコンピュータで実行されますが、Java または .NET の同期論理は Mobile Link サーバが稼動しているコンピュータで実行されます。このため、統合データベースの負荷が高い場合は、Java または .NET の同期論理が適していることがあります。

- **優先同期**　一部のテーブルを他のテーブルより高い頻度で同期する必要がある場合は、それらのテーブル用に別のパブリケーションとサブスクリプションを作成します。この優先パブリケーションを他のパブリケーションより頻繁に同期し、他のパブリケーションをピーク時以外の時間に同期することができます。

- **必要なローだけのダウンロード**　必要なローだけをダウンロードするように注意します。それぞれの同期ですべてのローをダウンロードする同期スクリプトを記述する方が簡単ですが、不要なローをダウンロードすると、同期のパフォーマンスが低下します。

- **スクリプト実行時の最適化**　統合データベースのスクリプトを実行するときのパフォーマンスは重要です。テーブルに対してインデックスを作成し、アップロード・カーソル・スクリプトとダウンロード・カーソル・スクリプトによって必要なローだけが効率的に特定されるようにすると便利です。ただし、インデックスが多すぎるとアップロードに時間がかかります。

- **ASA クライアントからの大規模なアップロードでは、ローの数を推定する**　多数のローをアップロードするときは、アップロードするロー数の推定値を dbmlsync に指定すると、アップロード速度を大幅に改善できます。この操作には、dbmlsync -urc オプションを使用します。

  詳細については、『Mobile Link クライアント』> 「-urc オプション」を参照してください。

Mobile Link のパフォーマンスに影響を与える主要な要因

Mobile Link 同期のためのスループットなど、システム全体のパフォーマンスは、通常、そのシステムのある時点のボトルネックによって制限されます。Mobile Link 同期では、同期スループットを制限するボトルネックとして次のものが考えられます。

- **統合データベースのパフォーマンス**　Mobile Link で特に重要なものは、Mobile Link スクリプトの実行速度です。複数のワーカ・スレッドがスクリプトを同時に実行するので、最高のスループットを得るには、同期スクリプトでのデータベース競合を避ける必要があります。

- **Mobile Link から統合データベースへの通信の帯域幅**　Mobile Link と統合データベースが同じコンピュータで稼働している場合や、別々のコンピュータで稼働していても高速ネットワークで接続されている場合には、帯域幅がボトルネックになる可能性は低くなります。

- **Mobile Link を実行しているコンピュータの処理速度**　Mobile Link を実行しているコンピュータの処理能力が低い場合や、Mobile Link のワーカ・スレッドとバッファに十分なメモリがない場合には、Mobile Link の実行速度が同期のボトルネックになることがあります。バッファとワーカ・スレッドが物理メモリに収まる場合は、Mobile Link サーバのパフォーマンスがディスク速度に左右されることはありません。

- **Mobile Link のワーカ・スレッド数**　スレッド数が小さければ小さいほど、データベース接続数は少なくなるので、統合データベースでの競合発生率は低くなり、オペレーティング・システムのオーバヘッドは少なくなります。ただし、ワーカ・スレッドの数があまりに少ないと、ワーカ・スレッドが開放されるまでクライアントが待機したままになったり、統合データベースへの接続数が少ないために、効率的な重複ができなくなったりします。

- **クライアントから Mobile Link への通信の帯域幅**　ダイヤルアップ接続、広域ネットワーク (WAN) や無線ネットワークでの接続など、通信速度の遅い接続では、クライアントと Mobile Link のワーカ・スレッドがデータ転送を待たなければならないことがあります。

- **クライアントの処理速度**　ダウンロードには、ローやインデックスの書き込みなど、より多くのクライアント処理が含まれるため、クライアントの処理速度の遅さは、アップロードよりもダウンロードでボトルネックになる可能性が高くなります。

Mobile Link のパフォーマンスのチューニング

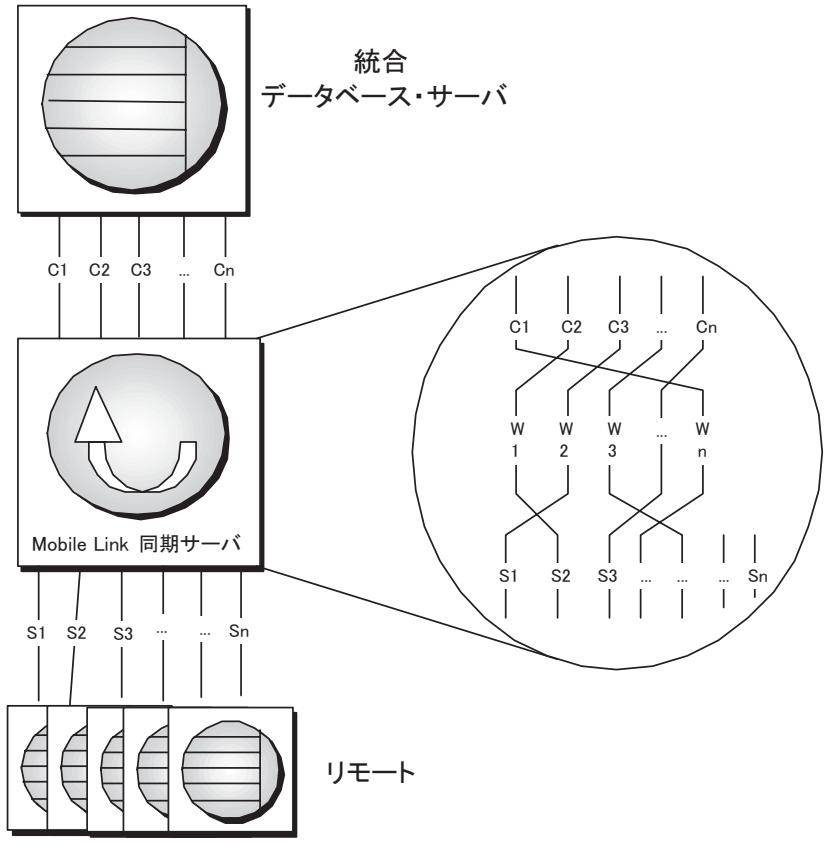
Mobile Link 同期のスループットを最大限にするには、複数の同期が同時に発生し、効率よく実行することが重要になります。複数の同期を同時に実行するために、Mobile Link ではワーカ・スレッドを同期ごとに割り当てます。ワーカ・スレッドは、クライアントからアップロードされた変更を受け取り、それらを統合データベースに適用します。この処理が終わると、ワーカ・スレッドは統合データベースから変更をフェッチして、それらをクライアントにダウンロードします。各ワーカ・スレッドは、同期スクリプトを使い、統合データベースへの単一の接続を使用して、変更の適用とフェッチを行います。

競合

最も重要な要因は、同期スクリプトでのデータベース競合を避けることです。複数のクライアントがデータベースを使用する他の場合と同様、クライアントがデータベースに同時にアクセスするときには、データベース競合を最小限にします。同期のたびに修正が必要なデータベースのローがあると、競合の発生率が高くなります。たとえば、カウンタを増分するスクリプトがある場合、カウンタを更新することがボトルネックになる可能性があります。

次の図は、以下の内容を示しています。

- 統合データベースへの接続プール (C1 ～ C*n*)
- 同期要求の数 (S1 ～ S*n*)
- Mobile Link ワーカ・スレッド (W1 ～ W*n*)



ワーカ・スレッドの数より同期要求の数が多くなると、同期完了後にワーカ・スレッドが使用できるようになるまで、余分な要求はキューイングされます。ワーカ・スレッド数と接続数は制御できますが、Mobile Link は、1 つのワーカ・スレッドに最低 1 つの接続があることを常に確認します。ワーカ・スレッドよりも多くの接続がある場合、余分な接続はアイドル状態になります。スクリプト・バージョンが複数ある場合には、後述するように、余分な接続が役に立つこともあります。

ワーカ・スレッド数

同期スクリプトで発生する競合以外に、同期スループットの最も重要な要因になるのがワーカ・スレッド数です。ワーカ・スレッド数は、同時に進行する同期の数を制御します。

最適なワーカ・スレッド数を判別するにはテストが不可欠です。

ワーカ・スレッドの数を増やすと、より多くの同期を重複させたり、スループットを向上させたりすることができる反面、重複する同期の間でリソースやデータベースの競合が増えて、1 つの同期にかかる時間も増えます。ワーカ・スレッドの数を増やすと、個々の同期の時間が長くなることで、より多く同時に同期を実行することが重要になります。ただし、さらにワーカ・スレッドを追加すると、スループットが低下します。使用する環境に最も適したワーカ・スレッド数を決定するには実験が必要ですが、次の情報も参考になります。

パフォーマンス・テストで、最高のアップロード・スループットは、ワーカ・スレッド数が比較的少ないときに発生することがわかりました。ほとんどの場合、3 ～ 10 のワーカ・スレッドが最適でした。統合データベース、データ容量、データベース・スキーマ、同期スクリプトの複雑さ、使用したハードウェアなどの要因によって結果は変動します。ボトルネックの一般的な原因は、統合データベース内でアップロード・スクリプトの SQL を同時に実行するワーカ・スレッド間の競合です。

ダウンロードに最適なワーカ・スレッド数は、クライアントから Mobile Link への帯域幅とクライアントの処理速度に依存しています。処理速度の遅いクライアントで、最適なダウンロード・パフォーマンスを得るには、より多くのワーカ・スレッドが必要です。これは、アップロードと比較すると、ダウンロードでは、より多くのクライアント処理が必要になる一方、それほど多くの統合データベース処理を必要としないからです。

Adaptive Server Anywhere クライアントでは、ダウンロード確認をオフのままにしておくと (オプションのダウンロード・バッファリングは無効にしない)、ダウンロードに対するワーカ・スレッドの最適数を減らすことができます。これは、ワーカ・スレッドがクライアントによるダウンロードの適用を待つ必要がないためです。Ultra Light クライアントではほとんど効果がありません。Ultra Light クライアントでは、ダウンロードを受信したら、バッファリングせずにダウンロードを適用するからです。

ダウンロード確認の無効化の詳細については、『Mobile Link クライアント』> 「SendDownloadACK (sa) 拡張オプション」を参照してください。

ダウンロードとアップロードの最高のスループットを得るために、Mobile Link には 2 つのオプションが用意されています。1 つは、ダウンロードを最適にするワーカ・スレッドの総数を指定するオプションです。もう 1 つは、アップロードを同時に適用できる数を制限して、アップロードのスループットを最適にするオプションです。

-w オプションはワーカ・スレッドの総数を制御します。デフォルトは 5 です。

-wu オプションは、アップロードを同時に統合データベースに適用できるワーカ・スレッド数を制限します。デフォルトでは、すべてのワーカ・スレッドがアップロードを同時に適用できますが、この場合、統合データベースで重大な競合が発生します。-wu オプションを使用すると、この競合を軽減すると同時に、多数のワーカ・スレッドでダウンロードを最適化し、アップロードを受け取ることができます。-wu オプションは、その数値がワーカ・スレッドの合計数未満の場合にだけ有効です。

詳細については、『Mobile Link 管理ガイド』> 「-w オプション」および 『Mobile Link 管理ガイド』> 「-wu オプション」を参照してください。

Mobile Link データベース接続

Mobile Link では、データベースの接続はワーカ・スレッドごとに作成されます。-cn オプションを使用すると、Mobile Link に対してデータベース接続のより大きなプールを作成するように指定できますが、Mobile Link が接続を閉じるか異なるスクリプト・バージョンを使用する必要がないかぎり、余分な接続はアイドル状態になります。

Mobile Link がデータベース接続を閉じ、新しい接続を開く場合が 2 つあります。1 つはエラーが発生した場合です。もう 1 つは、クライアントがある同期スクリプト・バージョンを要求し、現在使用されている接続の中にその同期バージョンを利用できる接続がない場合です。

注意
それぞれのデータベース接続はスクリプト・バージョンと関連付けられています。バージョンを変更するには、接続を閉じて再度開いてください。

複数の同期バージョンを使用している場合は、プール接続の最大数をワーカ・スレッド数 (デフォルト数) よりも大きな値にすることもあります。これによって、Mobile Link は、異なる同期バージョンが要求されるたびにデータベース接続を閉じて新しいデータベース接続を開く必要がなくなります。

定常的に複数のスクリプト・バージョンを使用するのであれば、接続数を増やすことで Mobile Link での接続開閉要求を減らすことができます。接続数がワーカ・スレッド数とバージョン数の積になる場合は、接続数を増やす必要はありません。

次のコマンド・ラインには、2 つのスクリプト・バージョンに対して Mobile Link をチューニングする例が示されています。

```
dbmlsrv9 -c "dsn=ASA 9.0 Sample" -w 5 -cn 10
```

データベース接続の最大使用可能数は、スクリプト・バージョンとワーカ・スレッドの積に 1 を加えたものです。したがって、データベース接続を閉じたり開いたりしないで確実に同期バージョンに対応するには、-cn を 10 に設定します。

次の例は、3 つのスクリプト・バージョンに対応する Mobile Link のチューニングです。

```
dbmlsrv9 -c "dsn=ASA 9.0 Sample" -w 7 -cn 21
```

スクリプト・バージョンの数に合わせて接続数を設定する方法の詳細については、『Mobile Link 管理ガイド』> 「-cn オプション」を参照してください。

Mobile Link のパフォーマンスのモニタ

さまざまなツールを使用して、同期のパフォーマンスをモニタできます。

Mobile Link モニタは、同期をモニタするためのグラフィカル・ツールです。このツールでは、同期の各部分における所要時間を、Mobile Link ユーザ別またはワーカ・スレッド別にソートして表示できます。

詳細については、「Mobile Link モニタ」143 ページを参照してください。

また、同期をモニタするための Mobile Link スクリプトがいくつかあります。これらのスクリプトにより、ビジネス論理でパフォーマンス統計を使用できます。たとえば、パフォーマンス情報を格納して後で分析したり、同期の実行時間が長すぎる場合に DBA に警告できます。詳細については以下を参照してください。

- 『Mobile Link 管理ガイド』> 「download_statistics 接続イベント」
- 『Mobile Link 管理ガイド』> 「download_statistics テーブル・イベント」
- 『Mobile Link 管理ガイド』> 「synchronization_statistics 接続イベント」
- 『Mobile Link 管理ガイド』> 「synchronization_statistics テーブル・イベント」
- 『Mobile Link 管理ガイド』> 「time_statistics 接続イベント」
- 『Mobile Link 管理ガイド』> 「time_statistics テーブル・イベント」
- 『Mobile Link 管理ガイド』> 「upload_statistics 接続イベント」
- 『Mobile Link 管理ガイド』> 「upload_statistics テーブル・イベント」

第 7 章

Mobile Link モニタ

この章の内容

Mobile Link モニタは、Mobile Link 同期をモニタするためのツールです。この章では、Mobile Link モニタの使用方法について説明します。

モニタの使用方法を説明したチュートリアルについては、『Mobile Link チュートリアル』> 「チュートリアル：Mobile Link スクリプトの作成と同期のモニタリング」を参照してください。

概要

Mobile Link モニタは、同期のパフォーマンスに関する詳細情報を提供する Mobile Link 管理ツールです。

モニタを起動して Mobile Link 同期サーバに接続すると、そのモニタリング・セッションで発生するすべての同期に関する統計情報の収集が開始されます。接続を切断するか、Mobile Link サーバを停止するまで、モニタはデータを収集し続けます。

モニタのインタフェースでは、表形式またはグラフィカル形式でデータを表示できます。データをバイナリ形式で保存して後からモニタに表示したり、.csv 形式で保存して Microsoft Excel などの別のツールで開いたりすることもできます。または、Mobile Link がサポートしているリレーショナル・データベースなどの ODBC データ・ソースにエクスポートすることも可能です。

同期に関するさまざまな情報を、モニタの出力で確認できます。たとえば、エラーが発生した同期や、指定したその他の条件に一致する同期をすばやく特定できます。期間の異なる同期においてほぼ同時に終了したフェーズがあるかどうかをチェックすると、(同期は前のフェーズが完了するのを待ってから処理を続行するため) 可能性のある同期スクリプトの競合を特定できます。

特に Mobile Link 同期サーバから別のコンピュータでモニタを実行する場合、モニタリングによってパフォーマンスが低下することはないため、Mobile Link モニタは開発時と運用時に日常的に使用できます。

Mobile Link モニタの起動

Mobile Link モニタの起動時に同期が行われている場合、モニタは、ワーカ・スレッドが解放されるまで起動できません。そのため、同期は、モニタを起動してから開始してください。モニタが起動した後は、Mobile Link ワーカ・スレッドを使用しません。

Mobile Link 同期サーバごとに、モニタのインスタンスを 1 つ稼働できます。

❖ データのモニタリングを開始するには、次の手順に従います。

1. [スタート] メニューで、[プログラム] － [SQL Anywhere 9] － [Mobile Link] － [Mobile Link モニタ] を選択します。

   または、コマンド・プロンプトで **dbmlmon** と入力することもできます。詳細については、以下を参照してください。

2. 統合データベースと Mobile Link 同期サーバを起動します (起動してない場合)。

3. Mobile Link モニタで、[モニタ] － [Mobile Link サーバへの接続] を選択します。

   [Mobile Link サーバへの接続] ダイアログが表示されます。

   モニタ接続は、Mobile Link 同期サーバへの同期接続と同じように開始されます。たとえば、-zu+ を指定して Mobile Link サーバを起動した場合は、ここで指定するユーザ ID が何であっても構いません。すべての Mobile Link モニタ・セッションに対して、スクリプト・バージョンが for_MLMonitor_only に設定されます。

   [Mobile Link サーバへの接続] ダイアログで次の情報を入力してください。

   - **[ホスト]**　Mobile Link 同期サーバが稼働しているコンピュータです。デフォルトでは、モニタが稼働しているコンピュータです。

- **[ネットワーク・プロトコル]** これらの値は、Mobile Link 同期サーバが同期要求に使用しているのと同じプロトコルとポートに設定してください。
- **[追加のネットワーク・パラメータ]** オプションのパラメータを設定できます。次のパラメータを設定できます。複数のパラメータを指定する必要がある場合は、セミコロンで区切ります。
  - **buffer_size=***number* (HTTP と HTTPS のみ)
  - **client_port=***nnnn*
  - **client_port=***nnnn-mmmmm*
  - **persistent={0|1}**
  - **proxy_host=***proxy_hostname* (HTTP と HTTPS のみ)
  - **proxy_port=***proxy_portnumber* (HTTP と HTTPS のみ)
  - **url_suffix=***suffix* (HTTP と HTTPS のみ)
  - **version=***versionnumber* (HTTP と HTTPS のみ)

  ネットワーク・パラメータの詳細については、『Mobile Link クライアント』> 「Ultra Light 同期クライアントのネットワーク・プロトコルのオプション」を参照してください。

4 同期を開始します。

  収集されたデータがモニタに表示されます。

コマンド・ラインでの dbmlmon の起動

コマンド・ライン・オプションを使用すると、モニタでファイルを開いたり、起動時に Mobile Link 同期サーバへ接続することができます。次に構文を示します。

dbmlmon [*connect-options* | *inputfile*.{ **mlm** | **csv** } | **-?**]

文中の各項目を次に説明します。

connect-options これは、次のうちの 1 つまたは複数です。

-u *ml_username*

-p *password*

-x { **tcpip** | **http** | **https** } [**(** *keyword=value*;... **)**]

上記で説明している keyword=value のペアは、ホスト、プロトコル、および追加のネットワーク・パラメータです。

-o *outputfile*.{ **mlm** | **csv** }

-o オプションは、接続の最後でモニタを閉じ、指定したファイルにセッションを保存します。

-?　**dbmlmon -?** と入力すると、dbmlmon の構文が表示されます。

*注意：*オプション -u と -x は接続するために必須です。

❖ Mobile Link モニタを停止するには、次の手順に従います。

1. モニタで、[モニタ] ー [Mobile Link サーバからの切断] を選択します。データの収集が停止します。

   Mobile Link 同期サーバをシャットダウンするか、モニタを終了して、データの収集を停止することもできます。

   モニタを終了する前に、このセッションのデータを保存できます。詳細については、「モニタのデータの保存」154 ページを参照してください。

2. モニタを終了する準備ができたら、[ファイル] - [閉じる] を選択します。

Mobile Link モニタの使用

このモニタには 3 つのウィンドウ枠があります。

- **[詳細テーブル]** 一番上のウィンドウ枠です。これは、各同期の合計所要時間と、同期の各部分に要した時間を示すスプレッドシートです。

  詳細については、「[詳細テーブル] ウィンドウ枠」149 ページを参照してください。

- **[チャート]** 真ん中のウィンドウ枠です。ここには、データがグラフィックで表示されます。このウィンドウ枠の下の目盛りは時間を表します。[概要] ウィンドウ枠でデータの周りにボックスを描くか、[ビュー] ー [移動] を選択して、[チャート] ウィンドウ枠に表示されているデータを選択できます。

  詳細については、「[チャート] ウィンドウ枠」150 ページを参照してください。

- **[概要]** 一番下のウィンドウ枠です。ここには、すべてのデータの概要が表示されます。データを選択して [チャート] に表示するには、[概要] 内でクリックしてボックスを描きます。[チャート] には、このボックスに含まれるすべてのデータが表示されます。

  詳細については、「[概要] ウィンドウ枠」152 ページを参照してください。

また、データをカスタマイズするための [オプション] ダイアログと、詳細な情報を確認するための [プロパティ] ダイアログがあります。詳細については、次の項目を参照してください。

- 「[オプション] ダイアログ」152 ページ
- 「セッション・プロパティ」153 ページ
- 「同期プロパティ」153 ページ

[詳細テーブル] ウィンドウ枠

[詳細テーブル] には、同期の各部分の所要時間に関する情報が表示されます。すべての時間は Mobile Link 同期サーバによって測定されます。対応するスクリプトが定義されていなくても、所要時間が 0 ではない場合があります。

[ツール] ー [オプション] を開き、次に [テーブル] タブを開くと、[詳細テーブル] ウィンドウ枠に表示されるカラムを選択できます。使用可能な統計の説明については、「Mobile Link の統計のプロパティ」158 ページを参照してください。

デフォルトでは、次のカラムが表示されます。

- **[sync]**　各同期を識別します。
- **[worker]**　同期を実行した Mobile Link ワーカ・スレッドを識別します。ワーカは *n.m* で識別されます。*n* はストリーム番号、*m* はスレッド番号です。
- **[user]**　同期ユーザを識別します。
- **[version]**　同期スクリプトのバージョンです。

  スクリプト・バージョンの詳細については、「スクリプト・バージョン」298 ページを参照してください。
- **[start_time]**　Mobile Link 同期サーバが同期を開始した日時です (これは、クライアントが同期を要求した時点よりも遅い場合があります)。
- **[duration]**　同期の合計時間 (秒)。
- **[verify_upload]**　認証を必要とする同期設定の場合に、Mobile Link が同期要求、ユーザ名、パスワードを検証するのに要する時間 (秒)。
- **[preload_upload]**　Mobile Link がクライアントからアップロードされたデータを受信するのに要する時間 (秒)。
- **[begin_sync]**　begin_synchronization スクリプトが実行されている場合、実行に要する時間 (秒)。

- **[upload]** アップロードを統合データベースに適用するのに要する時間 (秒)。これは、begin_upload スクリプトから end_upload スクリプトまでの時間です。
- **[prepare_for_download]** prepare_for_download スクリプトが実行されている場合、実行に要する時間 (秒)。
- **[download]** データのダウンロードに要する時間 (秒)。これは、begin_download スクリプトから end_download スクリプトまでの時間です。ダウンロード確認が有効になっている場合は、ダウンロードをリモート・データベースに適用し、確認を返す時間が含まれます。
- **[end_sync]** end_synchronization スクリプトが実行されている場合、実行に要する時間 (秒)。

テーブルを特定のカラムでソートするには、カラムの見出しをクリックします。モニタに新しいデータが表示された場合は、追加されたときにソートされます。

[ビュー] ー [詳細テーブル] をクリアすることで、[詳細テーブル] ウィンドウ枠を閉じることができます。

[チャート] ウィンドウ枠

[チャート] ウィンドウ枠には、[詳細テーブル] と同じ情報がグラフィックで表示されます。[チャート] 内のバーは、各同期で要した時間を表し、バーの各部分は同期のフェーズを表します。

データの表示

[詳細テーブル] で同期をクリックすると、その同期が選択されます。

同期をダブルクリックすると、その同期の [同期セッション・プロパティ] が開きます。詳細については、「同期プロパティ」153 ページを参照してください。

スレッドまたはユーザごとにデータをグループ化

データをワーカ・スレッドまたはユーザごとにグループ化できます。[ビュー] ー [ユーザ別] または [ビュー] ー [ワーカ・スレッド別] を選択します。

データの拡大

表示されているデータを選択するには、次の 3 つの方法があります。

- **ズーム・オプション**　[表示] メニューのズーム・オプションとツールバーのズーム・ボタンを使用して、ズーム・インやズーム・アウトができます。同期によって使用可能な領域を埋めるには、[選択範囲にズーム] を使用します。

- **スクロールバー**　[チャート] ウィンドウ枠の下のスクロールバーをクリックしてスライドさせます。

- **[移動] ダイアログ**　[ビュー] ー [移動] を選択してこのダイアログを開きます。次のような [移動] ダイアログが表示されます。

  [開始の日付と時刻] では、[チャート] ウィンドウ枠に表示するデータの起動時刻を指定できます。この設定を変更する場合は、少なくとも日付／時刻の年月日を指定してください。

  [チャートの範囲] では、表示する期間を指定できます。チャート範囲は、ミリ秒、秒、分、時間、または日数で指定できます。チャート範囲によって、データの細分性が決定されます。時間の長さが短いほど、より詳細に表示されます。

- **[概要] ウィンドウ枠**　[概要] ウィンドウ枠のボックスは、[チャート] に表示されている領域を示しています。このボックスを使用すると、表示するデータの部分をすばやく選択できます。ボックスのサイズを変更したりボックスを移動したりして、異なるデータを表示したり、異なる細分性でデータを表示したりできます。ボックスを小さくすると、[チャート] に表示されるデータの間隔が短くなり、より詳細なデータが表示されます。現在のボックスをクリックして移動しても、表示倍率は変化しません。[概要] の内部をドラッグして、ボックスをもう一度描くと、表示の倍率と位置を変更できます。

時間軸

[チャート] ウィンドウ枠の下には、期間を示す目盛りがあります。時間のフォーマットは、表示される時間の間隔に応じて自動的に再調整されます。目盛り上にカーソルを置くと、いつでも完全な日付／時刻を表示できます。

デフォルトの色スキーム

[オプション] ダイアログ ([ツール] メニューから開く) では、[チャート] ウィンドウ枠の色を表示または設定できます。[チャート] ウィンドウ枠のデフォルトの色スキームでは、アップロードを緑色、ダウンロードを赤色、開始／終了フェーズを青色、各フェーズの以前の部分を暗い影で表します。

色の設定については、「[オプション] ダイアログ」152 ページを参照してください。

[概要] ウィンドウ枠

[概要] ウィンドウ枠には、モニタのセッション全体の概要が表示されます。[チャート] ウィンドウ枠に現在表示されている領域は、[概要] ウィンドウ枠内にボックスで示されます。[概要] ウィンドウ枠内でクリックしてボックスを移動すると、[チャート] に表示されるデータの起動時刻が移動します。また、[概要] 内でドラッグしてボックスをもう一度描き、ボックスの位置とサイズを変更するとデータの起動時刻と範囲が変更されます。

[概要] ウィンドウ枠は、[モニタ] ウィンドウの他の部分から切り離すことができます。[オプション] ダイアログで、[概要] タブを開き、[概要のウィンドウをメイン・ウィンドウにアタッチしたままにする] チェックボックスをオフにします。

詳細については、「[オプション] ダイアログ」152 ページを参照してください。

[ビュー] ー [概要] をクリアすることで [概要] ウィンドウ枠を閉じることができます。

[オプション] ダイアログ

[オプション] ダイアログでは、[チャート] ウィンドウ枠 (Mobile Link モニタの中央のウィンドウ枠) と [概要] ウィンドウ枠 (下のウィンドウ枠) のグラフィック表示の色とパターンなど、いくつかの設定を指定できます。

[オプション] ダイアログを開くには、モニタを開き、[ツール] ー [オプション] を選択します。

デフォルトのリストア

デフォルト設定をリストアするには、ファイル *mlMonitorSettings* を削除します。このファイルは、ユーザ・プロファイル・ディレクトリにあります。

セッション・プロパティ

[セッション・プロパティ] ダイアログには、モニタリング・セッションに関する基本情報が表示されます。

[セッション・プロパティ] ダイアログを開くには、モニタを開き、[ファイル] ー [プロパティ] を選択します。

同期プロパティ

[詳細テーブル] または [チャート] 内で同期をダブルクリックすると、その同期のプロパティが表示されます。

すべてのテーブルの統計 (その同期のすべてのテーブルの合計) か、個々のテーブルに関する統計を表示できます。ドロップダウン・リストには、その同期に関係したテーブルのリストが表示されます。

同期プロパティの統計値の説明については、「Mobile Link の統計のプロパティ」158 ページを参照してください。

モニタのデータの保存

モニタ・セッションのデータを、バイナリ・ファイル (.mlm) または値をカンマで区切ったテキスト・ファイル (.csv) として保存するか、リレーショナル・データベースのテーブルまたは Microsoft Excel ファイルとして保存できます。

ファイルへの保存

データをファイルとして保存するには、[ファイル] － [名前を付けて保存] を選択します。

- 保存したデータを Mobile Link モニタで表示する場合は、データをバイナリ (.mlm) ファイルとして保存します。ファイルを再び開くには、[ファイル] － [開く] を選択します。
- 保存したデータを Microsoft Excel などの別のツールで表示する場合は、カンマで区切ったファイル (.csv) として保存します。この場合、セッションのプロパティ・シートと同期のプロパティ・シートの情報のうち、テーブルごとの情報とセッションの起動時刻／終了時刻を除くすべてが保存されます。.csv ファイルをモニタで開くこともできます。

  .csv ファイル・フォーマットでは、時間がミリ秒数で格納されます。

データが自動的にファイルに保存されるように指定できます。これを行うには、[ツール] － [オプション] を選択し、[一般] タブで出力ファイル名を入力します。出力ファイルが新しいデータで上書きされます。

リレーショナル・データベースまたは Excel へのエクスポート

モニタのデータは、ODBC または JConnect 接続を使用してエクスポートすることもできます。JConnect を使用する場合、Adaptive Server Anywhere または Adaptive Server Enterprise データベースへのエクスポートのみが可能です。ODBC を使用する場合、Mobile Link によってサポートされているリレーショナル・データベースまたは Excel にエクスポートできます。

データをエクスポートするときは、モニタ・セッションのすべてのカラムに加え、export_time というカラムがエクスポートされます。export_time カラムは、エクスポートの実行時間を識別します。

カラムのいくつかは予約語のため、データ・ソースは引用符付き識別子が有効になっている必要があります。モニタによって、Adaptive Server Anywhere、Adaptive Server Enterprise、Microsoft SQL Server のデータベースは自動的に引用符付き識別子が有効になります。引用符付き識別子が有効になっていないと、エクスポートは失敗します。

❖ データベースまたは Excel にデータをエクスポートするには、次の手順に従います。

1. モニタ情報を収集した後、Mobile Link 同期サーバから切断します。
2. Mobile Link モニタで、[ファイル] － [データベースへのエクスポート] を選択します。

   [データベースへのエクスポート] ダイアログが表示されます。
3. 出力のオプションを選択します。
   - データを保存するために作成するテーブルの名前を 2 つ指定します。指定しない場合は、デフォルト (mlm_by_sync と mlm_by_table) が使用されます。そのテーブルが存在しない場合は、モニタによって作成されます。Excel 出力では、2 つのテーブル名は作成する 2 つのワークシートを識別します。
   - 既存のテーブルにあるデータを上書きするかどうかを選択します。データを上書きしない場合は、新しいデータが既存のデータに追加されます。
4. [OK] をクリックします。

   データベースに接続するか、ODBC または JConnect を使用して Excel スプレッドシートに接続するための [接続] ダイアログが表示されます。

統計のカスタマイズ

ウォッチ・マネージャを使用すると、指定した基準を満たす同期を視覚的に区別できます。たとえば、大規模な同期、長時間の同期、長時間を要する小規模な同期、または警告を受信した同期を強調表示できます。

ウォッチ・マネージャを開くには、モニタを開き、[ツール] ー [ウォッチ・マネージャ] をクリックします。

ウォッチ・マネージャの左ウィンドウ枠には、使用可能なすべてのウォッチのリストが表示されます。右ウィンドウ枠には、アクティブなウォッチのリストが表示されます。ウォッチをアクティブ・リストに追加したり、アクティブ・リストから削除するには、左ウィンドウ枠でウォッチを選択し、該当のボタンをクリックします。

3 つの事前に定義されたウォッチ ([アクティブ]、[完了]、[失敗]) があります。事前に定義されたウォッチを編集して、表示方法を変更できます。また、右ウィンドウ枠からこれらのウォッチを削除すると、非アクティブにできます。

[チャート] には、ウォッチの条件を満たしていない同期は表示されません。すべてのウォッチを無効に ([現在のウォッチ] リストから削除) すると、[チャート] または [概要] に同期が表示されなくなります。

右ウィンドウ枠でのウォッチの順序は重要です。リストの上にあるウォッチから先に処理されます。[上へ移動] ボタンと [下へ移動] ボタンを使用して、右ウィンドウ枠でのウォッチの順序を編成できます。

事前に定義されたウォッチを使用して、別のウォッチを作成できます。ウォッチの条件を編集するには、古い条件を削除してから新しいウォッチ条件を追加します。

❖ 新しいウォッチを作成するには、次の手順に従います。

1 ウォッチ・マネージャで [新規] をクリックします。

[新規ウォッチ] ダイアログが表示されます。

2　[名前] ボックスにウォッチの名前を入力します。

3　プロパティ、比較演算子、値を選択します。

プロパティの完全なリストについては、「Mobile Link の統計のプロパティ」158 ページを参照してください。

4　[追加] をクリックします (設定を保存するために [追加] をクリックしてください)。

5　必要に応じて、別のプロパティ、演算子、値を選択して、[追加] をクリックします。

6　[チャート] ウィンドウ枠のウォッチのパターンを選択します ([チャート] ウィンドウ枠は、Mobile Link モニタの中央のウィンドウ枠です)。

7　[概要] ウィンドウ枠のウォッチの色を選択します ([概要] ウィンドウ枠は、Mobile Link モニタの下部のウィンドウ枠です)。

Mobile Link の統計のプロパティ

次に示すのは、Mobile Link モニタで使用できるプロパティのリストです。これらの統計は、[新規ウォッチ] ダイアログ、[詳細テーブル] ウィンドウ枠、[同期プロパティ] で表示できます。[同期プロパティ] ダイアログでは、プロパティ名にアンダースコアが付いていません。

[新規ウォッチ] ダイアログの詳細については、「統計のカスタマイズ」156 ページを参照してください。

[詳細テーブル] の詳細については、「[詳細テーブル] ウィンドウ枠」149 ページを参照してください。

[同期プロパティ] ダイアログの詳細については、「同期プロパティ」153 ページを参照してください。

プロパティ	注意
active	同期が進行中の場合は true
begin_sync	begin_synchronization イベントの時間
completed	同期が正常に完了した場合は true
conflicted_deletes	アップロード済みで競合が検出された削除の数
conflicted_inserts	アップロード済みで競合が検出された挿入の数
conflicted_updates	アップロード済みで競合が検出された更新の数
connection_retries	Mobile Link 同期サーバが統合データベースとの接続をリトライした回数
download	ダウンロードの時間
download_bytes	同期クライアントにダウンロードされたバイト数

プロパティ	注意
download_deleted_rows	Mobile Link 同期サーバによって (download_delete_cursor スクリプトを使用して) 統合データベースからフェッチされたロー削除の数
download_errors	ダウンロード中に発生したエラーの数
download_fetched_rows	Mobile Link 同期サーバによって (download_cursor スクリプトを使用して) 統合データベースからフェッチされたローの数
download_filtered_rows	クライアントがアップロードしたローと一致するため、Mobile Link クライアントにダウンロードされなかったフェッチ済みローの数
download_warnings	ダウンロード中に発生した警告の数
duration	Mobile Link 同期サーバが測定した同期の合計時間。これには、同期要求がキューイングされて使用可能なワーカ・スレッドを待機している時間は含まれません。
end_sync	end_synchronization イベントの時間
ignored_deletes	アップロード済みで無視された削除の数
ignored_inserts	アップロード済みで無視された挿入の数
ignored_updates	アップロード済みで無視された更新の数
preload_upload	クライアントから Mobile Link 同期サーバへのアップロード・データの転送時間
prepare_for_download	prepare_for_download イベントの時間
start_time	同期開始の日付／時刻 (ISO-8601 拡張フォーマット)。ロケール設定に応じて、このフォーマットのすべてのフィールドを *YYYY-MM-DD hh*:*mm*:*ss.sss* または *YYYY-MM-DD hh*:*mm*:*ss*,*sss* の形式で指定する必要があります。
sync	モニタ・セッション内で同期をユニークに識別する数

プロパティ	注意
sync_deadlocks	同期で検出された統合データベース内のデッドロックの合計数
sync_errors	同期で発生したエラーの合計数
sync_tables	同期に関係したクライアント・テーブルの数
sync_warnings	同期で発生した警告の合計数
upload	統合データベースにデータがアップロードされる時間
upload_bytes	同期クライアントからアップロードされたバイト数
upload_deadlocks	アップロード中に検出された統合データベース内のデッドロック数
upload_deleted_rows	同期クライアントからアップロードされたロー削除の数
upload_errors	アップロード中に発生したエラーの数
upload_inserted_rows	同期クライアントからアップロードされたロー挿入の数
upload_updated_rows	同期クライアントからアップロードされたロー更新の数
upload_warnings	アップロード中に発生した警告の数
user	Mobile Link クライアントの名前
verify_upload	同期プロトコルを検証し、同期クライアントを認証する時間
version	同期バージョンの名前
worker	同期に使用する Mobile Link ワーカ・スレッドの識別子。形式は *n.m* で、*n* はストリーム番号、*m* はスレッド番号です。

第 8 章

リダイレクタによる Web サーバを経由した同期

この章の内容

この章では、リダイレクタに焦点を当てながら、Web サーバを経由した Mobile Link 同期のルート指定方法について説明します。

概要

Mobile Link には、「**リダイレクタ**」と呼ばれる Web サーバの拡張機能が組み込まれています。これによって、クライアントと Mobile Link 同期サーバ間の要求や応答のルートが指定されます。このようなプラグインは、一般に「**リバース・プロキシ**」とも呼ばれます。

Web サーバ経由で要求を送信する主な理由は、HTTP または HTTPS 同期用の、Web サーバとファイアウォールの既存の設定を利用するためです。ただし、Web サーバはリダイレクタがなくてもプロキシとして動作できます。リダイレクタは、複数の Mobile Link 同期サーバが稼働している場合に最も便利です。

詳細については、「Web サーバを使用した場合のオプション」163 ページを参照してください。

リダイレクタを使用すると、Mobile Link 同期サーバが稼働する 1 台以上のコンピュータに特定の URL 要求をルート指定するように、Web サーバを設定できます。

Web サーバは、特定の URL または特定の範囲の URL からの要求を拡張プログラム (通常、perl 言語の CGI スクリプト、DLL、またはその他の拡張メカニズムで記述されている) に渡すように設定できます。これらの拡張プログラムは、外部データ・ソースにアクセスしたり、Web サーバがクライアントに配信する応答を提供したりできます。

負荷分散とフェールオーバ

リダイレクタは、単純なラウンド・ロビン・アルゴリズム (固定の循環的順序でサーバを選択) を使用して、負荷分散とフェールオーバを実装しています。設定された間隔で 各 Mobile Link 同期サーバをテストし、応答していないサーバへの要求の送信を停止します。Mobile Link 同期サーバが再び稼働するとそれを検出し、そのときに要求の送信を再開します。

HTTPS 同期

HTTPS 同期では、HTTP ヘッダは、RSA 暗号化を使用して SSL/TLS で暗号化されてから、サーバへの送信やサーバからの送信が行われます。HTTPS が使用されるのは、Mobile Link クライアントと Web サーバとの接続に対してのみです。Web サーバは HTTPS を復号化し、リダイレクタ経由で Mobile Link に HTTP を送信します。

HTTPS プロトコルはその他の安全なプロトコルよりも低速なため、必要な場合にのみ HTTPS プロトコルを使用することをおすすめします。

サポートされる Web サーバ

プラグインは、次の Web サーバで使用できます。

リダイレクタ・プラグイン	サポート対象
ISAPI リダイレクタ	Microsoft Web サーバ
NSAPI リダイレクタ	Windows 上の Sun One (以前の Netscape) Web サーバ
サーブレット・リダイレクタ	Apache Tomcat、UNIX 上の Sun One Web サーバを含む、Java Servlet API 2.3 をサポートする Web サーバ
ネイティブな Apache リダイレクタ	Apache Web サーバ
M-Business Anywhere リダイレクタ	M-Business Anywhere Web サーバ

Web サーバを使用した場合のオプション

リダイレクタは、Web サーバ経由で Mobile Link 同期をルート指定する 1 つの方法です。リダイレクタは、ファイアウォール経由の同期や、複数の Mobile Link 同期サーバとの同期で特に役立ちます。

Web サーバ経由で要求を送信する主な理由は、HTTP または HTTPS 同期用の、Web サーバとファイアウォールの既存の設定を利用するためです。リダイレクタは、複数の Mobile Link 同期サーバが稼働している場合に最も便利です。

また、リダイレクタを使用せずに、Web サーバを経由して同期をルート指定することもできます。この場合は、Web サーバをプロキシとして設定して、Mobile Link サーバに同期をルート指定します。Web サーバでルート指定を行う方法の詳細については、Web サーバのマニュアルを参照してください。

次の表は、Mobile Link 同期の最適なルート指定方法を判断する上での推奨事項を示しています。

	直接接続が可能	直接接続が不可能
1 つの Mobile Link 同期サーバ	HTTP の代わりに TCP/IP を使用する	Web サーバを経由し、Mobile Link 同期サーバにメッセージを送信するために HTTP または HTTPS プロキシを使用する
複数の Mobile Link 同期サーバ	HTTP または HTTPS でリダイレクタを使用する	HTTP または HTTPS でリダイレクタを使用する

「リダイレクタによる Web サーバを経由した同期」161 ページを参照してください。

リダイレクタの設定

次の項では、同期要求を管理するための Web サーバの設定方法について説明します。

❖ 設定処理の概要

1 Mobile Link クライアントと Mobile Link 同期サーバを設定します。

「Mobile Link クライアントとサーバのリダイレクタ設定」166 ページを参照してください。

2 リダイレクタ設定ファイルが Web サーバと同じコンピュータ上にあることを確認します。

「リダイレクタのプロパティの設定 (すべてのバージョン共通)」169 ページ を参照してください。

3 リダイレクタ設定ファイルを修正します。

「リダイレクタのプロパティの設定 (すべてのバージョン共通)」169 ページ を参照してください。

4 Web サーバ固有の設定を行います。

次のうちの 1 つを参照してください。

- 「Netscape/Sun Web サーバ用の NSAPI リダイレクタ」172 ページ
- 「Microsoft Web サーバ用の ISAPI リダイレクタ」177 ページ
- 「サーブレット・リダイレクタ」180 ページ
- 「Apache リダイレクタ」186 ページ
- 「M-Business Anywhere リダイレクタ」191 ページ

Mobile Link クライアントとサーバのリダイレクタ設定

この項では、Mobile Link クライアントと Mobile Link 同期サーバを、Web サーバ経由で同期するように設定する方法について説明します。次の手順では、Web サーバ経由で送信される要求に必要なパラメータを設定します。

Mobile Link クライアント

❖ **Mobile Link クライアント(Adaptive Server Anywhere および Ultra Light) を設定するには、次の手順に従います。**

1. Mobile Link クライアントの通信タイプを、HTTP または HTTPS に設定します。

   Adaptive Server Anywhere クライアントの通信タイプの設定の詳細については、『Mobile Link クライアント』> 「CommunicationType (ctp) 拡張オプション」を参照してください。

   Ultra Light クライアントの通信タイプの設定の詳細については、『Mobile Link クライアント』> 「HTTP プロトコル・オプション」または 『Mobile Link クライアント』> 「HTTPS プロトコル・オプション」を参照してください。

2. Mobile Link クライアントに、次の HTTP/HTTPS 同期プロトコル・オプションを指定します。

   - **host**　Web サーバの名前または IP アドレス
   - **port**　HTTP 要求または HTTPS 要求を受け付ける Web サーバのポート
   - **url_suffix**　この設定は、使用している Web サーバによって異なります。
     - ISAPI Web サーバの場合

       exe_dir`/iaredirect.dll/ml/`

       exe_dir は *iaredirect.dll* のロケーションです。

- NSAPI Web サーバの場合

  mlredirect`/ml/`

  mlredirect は *obj.conf* ファイルにマップされている名前です。

- Apache Tomcat を含む、サーブレット・リダイレクタを使用した Java Servlet API 2.2 をサポートするサーバの場合

  `iaredirect/ml/`

- Apache 用のネイティブなリダイレクタの場合、*httpd.conf* ファイルにある <location> タグでの選択に関わらず、次を設定します。次に例を示します。

  `iaredirect/ml/`

Ultra Light クライアントのプロトコル・オプションの設定の詳細については、『Mobile Link クライアント』> 「HTTP プロトコル・オプション」または 『Mobile Link クライアント』> 「HTTPS プロトコル・オプション」を参照してください。

Adaptive Server Anywhere クライアントのプロトコル・オプションの設定の詳細については、『Mobile Link クライアント』> 「CommunicationAddress (adr) 拡張オプション」を参照してください。

Mobile Link 同期サーバ

❖ **Mobile Link 同期サーバを設定するには、次の手順に従います。**

1. クライアントとプロキシとの通信に HTTP または HTTPS を使用するには、HTTP プロトコルを指定して Mobile Link 同期サーバを起動する必要があります。リダイレクタでは、HTTPS を直接使用できません。

   たとえば、HTTP プロトコルを指定するには、dbmlsrv9 コマンドラインで次のように入力します。

   dbmlsrv9 -x http

   詳細については、「-x オプション」269 ページを参照してください。

2 また、Mobile Link 同期サーバに対して、次のパラメータを設定できます。

- **port** Mobile Link は、HTTP プロトコルのデフォルトとしてポート番号 80 を使用し、HTTPS プロトコルのデフォルトとしてポート番号 443 を使用します。Mobile Link 同期サーバと Web サーバが同じマシン上で稼働している場合、ポート番号 80 は通常、Web サーバによって使用されます。この場合、別のポート番号を指定してください。たとえば、ポート番号 2439 を使用できます。これは、Internet Assigned Numbers Authority (IANA) に登録された Mobile Link 同期サーバ用のポート番号です。

- **contd_timeout** 同期を中止する前に、部分的に完了した同期の次の部分を受信するまでに待機する秒数です。この設定はオプションです。また、デフォルト値は 30 秒です。

  低速ネットワークで大容量の同期を行う場合は、これらの timeout パラメータの値を増やすことができます。

port と contd_timeout の詳細については、『Mobile Link 管理ガイド』> 「-x オプション」を参照してください。

リダイレクタのプロパティの設定 (すべてのバージョン共通)

この項では、リダイレクタのプロパティを設定するための、Web サーバに共通の設定手順について説明します。

❖ リダイレクタのプロパティを設定するには、次の手順に従います。

1. 「Mobile Link クライアントとサーバのリダイレクタ設定」166 ページの手順を完了します。

2. *redirector.config* を Web サーバにコピーします。

   redirector.config ファイルは、Mobile Link 同期サーバのインストール時に SQL Anywhere インストール環境のサブディレクトリ *MobiLink¥redirector* に置かれます。

   Mobile Link 同期サーバが Web サーバと同じコンピュータにインストールされていない場合には、Web サーバが設定されているコンピュータに *redirector.config* をコピーしてください。

   Microsoft Web サーバについては、*Inetpub/scripts* ディレクトリに *redirector.config* をコピーします。それ以外の Web サーバについては、どのディレクトリに *redirector.config* をコピーしてもかまいません。

3. リダイレクタ設定ファイルを設定します。

   Web サーバと Mobile Link 同期サーバ間の通信を設定するには、Web サーバがインストールされているコンピュータの *redirector.config* ファイルを編集してください。

   このファイルでは、以下の指令を設定できます。

   - **LOG_LEVEL**　ログ・ファイルに書き込まれる出力の量を設定するときに使用します。値は 0、1、2 です。デフォルトは 1 で、2 に設定すると最大量の出力が生成されます。Apache リダイレクタの場合、この設定は何も

影響しません。Apache 設定ファイル *httpd.conf* の LogLevel セクションで、ログ・レベルを設定してください。

- **ML** `ML=host:port` の形式で、Mobile Link 同期サーバを稼働しているコンピュータをリストするときに使用します。ML では大文字と小文字を区別します。

- **ML_CLIENT_TIMEOUT** ある 1 つの同期のすべての手順が同じ Mobile Link 同期サーバに送信されていることを確認するときに使います。デフォルト値は 600 秒 (10 分) です。

  情報は、同期のある一定期間、特定の Mobile Link 同期サーバに維持されるため、同期のすべての手順を 1 つの同期サーバが扱うようにします。ML_CLIENT_TIMEOUT が設定されている間、リダイレクタはクライアントとサーバ間の関係を維持します。このパラメータの値は、ユーザの同期手順の最大値より大きい値にしてください。

- **SLEEP** サーバが機能していることをリダイレクタがチェックする間隔を秒単位で設定します。デフォルト値は 1800 (30 分) です。たとえば、`SLEEP=3600` と設定します。SLEEP では大文字と小文字を区別します。

  redirector.config には、以下の規則を適用します。

  - 1 行の長さの最大値は 300 文字
  - コメントはハッシュ文字 (#) で開始する
  - 指令を定義するときには、スペースやタブを使用しない

4 以下のいずれかの項で Web サーバ固有の設定を完了します。

- 「Microsoft Web サーバ用の ISAPI リダイレクタ」177 ページ

- 「Netscape/Sun Web サーバ用の NSAPI リダイレクタ」172 ページ

- 「サーブレット・リダイレクタ」180 ページ
- 「Apache リダイレクタ」186 ページ

例

サンプル・ファイル *redirector.config* を使って説明します。このファイルで指定されている内容は、以下のとおりです。

- リダイレクタは、サーバが機能していることを 1800 秒ごとにチェックする。
- Mobile Link 同期サーバを稼働しているコンピュータで、要求を処理できるコンピュータは 3 台。複数のサーバを指定する場合、負荷分散は自動的に有効になります。

```
SLEEP=1800
 ML=myServ-pc:80
 ML=209.123.123.1:8080
 ML=myCompany.com:8081
```

Netscape/Sun Web サーバ用の NSAPI リダイレクタ

NSAPI リダイレクタは、Sun One Web サーバ用に提供されています。これは、以前の Netscape iPlanet Enterprise Edition Web サーバです。次に、Sun One サービス・パック 1 の設定手順を説明します。

このリダイレクタは、Windows でのみ使用できます。UNIX で Netscape/Sun Web サーバ用のリダイレクタを使用するには、サーブレット・リダイレクタを使用します。「サーブレット・リダイレクタ」180 ページを参照してください。

❖ Sun One 用 NSAPI リダイレクタを設定するには、次の手順に従います。

1. 「リダイレクタのプロパティの設定 (すべてのバージョン共通)」169 ページ の手順を完了します。

2. 必要に応じて、Web サーバが設定されているコンピュータに *iaredirect.dll* ファイルをコピーします。このファイルは、Mobile Link 同期サーバとともに、SQL Anywhere インストール環境の *MobiLink¥redirector¥nsapi* サブディレクトリにインストールされます。

3. Sun One Web サーバの設定ファイル *obj.conf* を次のように更新します。

サンプル・ファイル
Mobile Link 同期サーバ用として設定済みの *obj.conf* の完全なサンプル・コピーは、*MobiLink¥redirector¥nsapi* に保管されています。サンプル・ファイル名は *obj.conf.example* です。このサンプル・ファイルを使用して、次のセクションがファイルのどこに該当しているかを確認できます。

ファイル *magnus.conf* と *obj.conf* の次のセクションを更新します。

- *magnus.conf* で、*iaredirect.dll* と *redirector.config* が配置される場所を指定します。

Init セクションの末尾に、次のテキストを追加します。*location* にはファイルの実際のロケーションを入力します (*iaredirect.dll* と *redirector.config* は Web サーバと同じコンピュータに置いてください。ロケーションはそれぞれ別でもかまいません)。

```
Init fn="load-modules" shlib="location/
iaredirect.dll"
 funcs="redirector,initialize_redirector"
 Init fn="initialize_redirector"
configFile="location/redirector.config"
```

- *obj.conf* で、URL で使用されるリダイレクタの名前を指定します。

  "default object" セクションの先頭に、次のテキストを追加します。このセクションは、*mlredirect* をどのような値にも変更できることを除いて、以下のようにしてください。*http://host:port/mlredirect/ml/** という形式のすべての要求は、リダイレクタとともに稼働している Mobile Link 同期サーバのいずれかに送信されます。

  ```
  <Object name=default>
   NameTrans fn="assign-name" from="/mlredirect/ml/
  *" name="redirectToML"
  ```

- *obj.conf* で、リダイレクタが呼び出すオブジェクトを指定します。"default object" セクションの後に、次のセクションを追加します。

  ```
  <Object name="redirectToML">
   Service fn="redirector" serverType="ml"
   </Object>
  ```

4 Mobile Link アップロード・ストリームのバッファ・サイズを設定します。

使用する Web サーバの *magnus.conf* ファイルに指令を追加して、アップロードとダウンロードのストリームに合うバッファ・サイズをバイト単位で設定します。次に例を示します。

```
ChunkedRequestBufferSize 2000000
```

この指令では、バッファが 2 MB に増えます。アップロード・データのサイズを確保するには、この値で十分です。

HTTPS 以外のネットワーク・プロトコルを使用している場合、これで設定は完了です。

5 HTTPS 同期を使用している場合は、次のようにしてサーバを設定します。

- Sun One Web Server Administration Server を起動します。

  [スタート] － [プログラム] － [iPlanet Web Server] － [Start iWS Administration Server] の順に選択します。

- Administration Server にログインします。

  [スタート] － [プログラム] － [iPlanet Web Server] － [Administer Web Server] の順に選択します。

  プロンプトが表示されたら、ユーザ ID とパスワードを入力します。

- [Servers] タブで、使用しているサーバをリストから選択し、[Manage] をクリックします。

- [Security] タブの [request a Certificate] をクリックします。

- 証明書要求を生成し、認証局による署名を得るか、gencert を使用して署名します。gencert には、別のライセンスが必要です。

  - 証明書要求に認証局からの署名を得るには、フォームに入力します。
  - gencert ユーティリティを使用するには、フォームに入力します。ただし、認証局の電子メール・アドレスの代わりに、自分の電子メール・アドレスを入力します。証明書要求のテキストをファイルに保存してから、gencert ユーティリティを実行します。詳細については、『Mobile Link 管理ガイド』 > 「証明書生成ユーティリティ」を参照してください。

- [Security] タブの [Install Certificate] をクリックします。フォームに入力し、署名付きの証明書のロケーションを指定します。
- [Manage Certificates] をクリックし、証明書が正しくインストールされていることを確認します。
- [Preferences] タブで、[Add Listen Socket] をクリックします。必要なパラメータを指定します。HTTPS のデフォルト・ポートは 443 です。[Security] ドロップダウン・リストから [On] を選択し、HTTPS 同期をアクティブにします。

  HTTPS の使用の詳細については、「Mobile Link トランスポート・レイヤ・セキュリティ」205 ページを参照してください。

例

次に、カスタマイズが必要な *magnus.conf* のセクションの例を示します。

```
Init fn="load-modules" shlib="D:/iaredirect.dll"
 funcs="redirector,initialize_redirector"
 Init fn=" initialize_redirector " configFile="D:/
redirector.config"
```

次に、カスタマイズが必要な *obj.conf* のセクションの例を示します。

```
<Object name=default>
 NameTrans fn="assign-name" from="/mlredirect/ml/*"
name="redirectToML"
 ...
 <Object name="redirectToML">
 Service fn="redirector" serverType="ml"
 </Object>
```

❖ 設定をテストするには、次の手順に従います。

1. 次の構文を使用してリダイレクタを呼び出します。

   `http://`*host*`:`*port*`/mlredirect/ml/`

2 ログ・ファイルをチェックして、リダイレクタが要求をログに記録しているかどうかを確認します。

*注意：*このテストは、Mobile Link 同期サーバへの接続を作成しません。

Microsoft Web サーバ用の ISAPI リダイレクタ

Microsoft Web サーバを使用している場合は、ISAPI 版のリダイレクタを使用できます。IIS 5.0 のセットアップ手順を次に示します。

❖ Microsoft Web サーバに ISAPI リダイレクタを設定するには、次の手順に従います。

1. 「リダイレクタのプロパティの設定 (すべてのバージョン共通)」169 ページ の手順を完了します。

2. *iaredirect.dll* ファイルを、Web サーバが設定されているコンピュータの *Inetpub/scripts* にコピーします。

   iaredirect.dll ファイルは、Mobile Link 同期サーバとともに、SQL Anywhere ディレクトリの下にある *MobiLink¥redirector¥isapi* にインストールされています。

   Inetpub/scripts ディレクトリは、Microsoft Web サーバのインストール・ディレクトリにあります。

3. *redirector.config* ファイルを、Web サーバが設定されているコンピュータの *Inetpub/scripts* にコピーします。

   HTTPS 以外のネットワーク・プロトコルを使用している場合、これで設定は完了です。正しく設定できなかった場合は、以下の「注意」を参照してください。

4. HTTPS 同期を使用している場合は、次のようにしてサーバを設定します。

   - [マイ コンピュータ] を右クリックし、ポップアップ・メニューから [管理] を選択します。
   - 左ウィンドウ枠で、[サービスとアプリケーション] フォルダを開きます。[インターネット情報サービス] を選択します。
   - 右ウィンドウ枠でデフォルトの Web サイトを右クリックし、ポップアップ・メニューで [構成] を選択します。

- [ディレクトリ セキュリティ] タブをクリックします。
- [サーバー証明書] をクリックします。

  [サーバー証明書] ウィザードが表示されます。
- [証明書の新規作成] を選択して証明書要求を生成します。表示されるプロンプトに従い、証明書要求をファイルに出力することを選択します。
- 証明書に署名します。

  証明書に署名するには、サード・パーティの認証局または gencert ユーティリティを使用できます。ただし、gencert ユーティリティには別のライセンスが必要です。詳細については、「証明書生成ユーティリティ」612 ページを参照してください。
- [サーバー証明書] をクリックします。

  [サーバー証明書] ウィザードが表示されます。ウィザードには、署名付きの証明書のインストールが可能であるというプロンプトが表示されています。このプロンプトに従います。
- [証明書の表示] をクリックし、証明書が正しくインストールされていることを確認します。

  HTTPS の使用の詳細については、「Mobile Link トランスポート・レイヤ・セキュリティ」205 ページを参照してください。

注意

Inetpub/scripts ディレクトリは、Web サーバのインストール時に実行アクセス権付きで作成されます。インターネット インフォメーション サービスを使用してディレクトリに実行アクセス権を付与できるのであれば、*redirector.config* と *iaredirect.dll* を異なるディレクトリに保管できます。

Mobile Link 同期サーバに接続できない場合は、*Inetpub/scripts* ディレクトリを指す仮想ディレクトリがないことが、問題の原因の可能性があります。これに該当する場合は、インターネット インフォメーション サービスを起動して、仮想ディレクトリを手動で作成します。

この仮想ディレクトリは *Inetpub/scripts* を指し、スクリプトと実行プログラムに対して実行アクセス権を持つ必要があります。その手順については、IIS のオンライン・ヘルプを参照してください。

❖ 設定をテストするには、次の手順に従います。

1. 次の構文を使用して ISAPI リダイレクタを呼出します。

   protocol:*//host*[:*port*]*/exec_dir/***iaredirect.dll/ml/**

   文中の各項目を次に説明します。

   - **protocol**　**http** または **https** です。
   - **host**　Web サーバのホスト名です。
   - **port**　デフォルトのポートでない場合は、Web サーバが受信しているポートです。
   - **exec_dir**　リダイレクタの dll である *iaredirect.dll* がインストールされているディレクトリです。デフォルトのディレクトリは *scripts* です。

     次に例を示します。

     ```
     http://server:8080/scripts/iaredirect.dll/ml/
     ```

2. ログ・ファイルをチェックして、リダイレクタが要求をログに記録しているかどうかを確認します。

   *注意：*このテストは、Mobile Link 同期サーバへの接続を作成しません。

サーブレット・リダイレクタ

サーブレット・リダイレクタは、Java サーブレット仕様バージョン 2.3 をサポートする Web サーバでサポートされます。次の手順は、Apache Tomcat 4.0.6 用のサーブレット・リダイレクタの設定例です。

また、Apache Web サーバ用のネイティブなリダイレクタもあります。詳細については、「Apache リダイレクタ」186 ページを参照してください。

Apache Tomcat 用のサーブレット・リダイレクタを設定する手順

この項では、Tomcat サーブレット・コンテナとともに Apache Web サーバで動作するように、サーブレット版のリダイレクタをインストールする方法を説明します。リダイレクタ・ソフトウェアのテストは、Tomcat バージョン 4.0.6 と Apache 2.0.47 を使用して実行済みです。

インストールには、以下の手順が必要です。

1. 「リダイレクタのプロパティの設定 (すべてのバージョン共通)」169 ページ の手順を完了します。
2. サーブレット版のリダイレクタを Tomcat にインストールします。
3. プロキシとして動作するように Apache Web サーバを設定します。

ここでは、Tomcat と Apache のインストール先として、それぞれ *%CATALINA_HOME%* と *%APACHE_HOME%* をルート・ディレクトリとします。

❖ **Tomcat にサーブレット・リダイレクタをインストールするには、次の手順に従います。**

1 Tomcat をスタンドアロン・サーバとしてインストールします。

Tomcat バイナリは、Apache Web サイト http://archive.apache.org/dist/jakarta/tomcat-4/archive/v4.0.6/bin/ の Jakarta プロジェクトからダウンロードできます。

2 必要に応じて、Tomcat の HTTP ポートを設定します。

Tomcat はデフォルトでポート 8080 にバインドされます。競合が発生する場合、別の Web サーバがこのポートを使用している可能性があります。

- *%CATALINA_HOME%/conf/server.xml* を開きます。
- 8080 を検索します (<Connector> タグ内にあります)。
- これを使用中でないポートに変更します。

3 サーブレット・リダイレクタを Web アプリケーションとしてインストールします。

- ファイル *iaredirect.war* を *%CATALINA_HOME%/webapps* にコピーします。
- Tomcat を停止し、再起動します。

  war ファイルが拡張され、リダイレクタ Web アプリケーション用のディレクトリ *iaredirect* が作成されます。

- ファイル *%CATALINA_HOME%/webapps/iaredirect/WEB-INF/web.xml* を編集します。**redirector.config** (<init-param> タグ内にあります) を検索し、*redirector.config* ファイルのパスを修正します。

  drive:/*path*/**redirector.config** を読み取るようにエントリ **redirector.config** を変更します。Windows オペレーティング・システムの場合も、*d:/redirector.config* のように、パスの区切り文字として通常のスラッシュを使用します。

- Tomcat を停止し、再起動して、変更を有効にします。

  変更を有効にすると、配備された場所に war ファイルが存在する必要はなくなります。

- これで、リダイレクタは次の URL から呼び出すことができます。

  http://tc-host:tc-port/iaredirect/ml/

ここで *tc-host* はマシンで、*tc-port* は Tomcat が受信しているポートです。

❖ Apache Web サーバをプロキシとして設定するには、次の手順に従います。

1 Apache Web サーバをインストールします。

バイナリは、Apache の Web サイト (http://www.apache.org) からダウンロードできます。

2 必要に応じて、Apache Web サーバのポートを変更します。

ファイル *%APACHE_HOME%/conf/httpd.conf* を編集し、目的のポートの **Port** 設定を変更します。

3 プロキシとして動作するように Apache を設定します。

%APACHE_HOME%/conf/httpd.conf ファイルに、次の 2 つの指令を追加します。

```
LoadModule proxy_module {module-path}/mod_proxy.so
 LoadModule proxy_connect_module {module-path}/
mod_proxy_connect.so
 LoadModule proxy_http_module {module-path}/
mod_proxy_http.so
```

たとえば、パスは *modules/mod_proxy.so* (デフォルト) とします。

4 リダイレクタの URL を Tomcat に転送するように Apache を設定します。

%APACHE_HOME%/conf/httpd.conf ファイルで、次の 2 つの指令を追加します。こうすると、Apache は http://localhost/iaredirect/* という形式の URL を、ポート 8080 で受信している Tomcat 4 コネクタに転送します。

```
ProxyPass /iaredirect http://localhost:8080/
iaredirect
```

ポート番号は、Tomcat で使用されているポート番号に一致している必要があります。Tomcat と Apache が同じマシンで実行されていない場合は、**localhost** の代わりに、Tomcat が実行されているマシン名を指定します。

HTTPS 以外のネットワーク・プロトコルを使用している場合、これで設定は完了です。

5 HTTPS 同期を使用している場合は、次のようにしてサーバを設定します (HTTPS の設定方法は、Apache および M-Business Anywhere 用のネイティブなリダイレクタの HTTPS 手順と同じことに注意してください)。

- mod_ssl と OpenSSL のバイナリをダウンロードしてインストールします。これらのバイナリは、Apache Module Registry (http://modules.apache.org/) にあります。*mod_ssl.so* は、*%APACHE_HOME%¥modules* にコピーしてください。*libeay32.dll* と *ssleay32.dll* は、*%APACHE_HOME%¥bin* にコピーしてください。

- reqtool.exe で要求を生成し、それをサード・パーティの認証局に送信して署名を受けるか、または gencert.exe を使用して直接証明書を生成することによって、サーバ証明書とプライベート・キーを生成します。プライベート・キーは、サーバ証明書と同じファイルに保存するか、独自のファイルに保存することができます。

- 次の行を *%APACHE_HOME%¥conf¥httpd.conf* ファイルに追加します。

  ```
  LoadModule ssl_module modules/mod_ssl.so
       SSLEngine on
       SSLCertificateFile certificate_file
  ```

  ここで *certificate_file* は、サーバの証明書ファイルのパスとファイル名です。

  サーバのプライベート・キーが、サーバの証明書と異なるファイルにある場合は、次の行を追加します。

  ```
  SSLCertificateKeyFile private_key_file
  ```

ここで *private_key_file* は、サーバのプライベート・キーのパスとファイル名です。

プライベート・キーがパス・フレーズを使用して暗号化されており、win32 上で稼働している場合は、次の行を追加します。

```
SSLPassPhraseDialog exec:exe_name
```

ここで *exe_name* は、stdout にパス・フレーズを返す実行ファイルのパスとファイル名です。

または、パス・フレーズは openssl を使用してプライベート・キーから除去することも可能です。

```
openssl rsa -in src_file -out dst_file
```

ここで *src_file* は、パス・フレーズで保護されているプライベート・キーのパスとファイル名です。*dst_file* は、保護されていないプライベート・キーが含まれる出力ファイルのパスとファイル名です。ただし、こうするとサーバのセキュリティが低下することがあるので注意してください。

HTTPS の使用の詳細については、「Mobile Link トランスポート・レイヤ・セキュリティ」205 ページを参照してください。

HTTPS 設定の例

次に、HTTPS 用に Apache を設定する方法の例を示します。この例では、Apache の仮想ホスト機能を使用してポート 443 (デフォルトの HTTPS ポート) からの HTTPS の読み込みと、ポート 80 からの HTTP の読み込みを同時に行います。

```
LoadModule ssl_module modules/mod_ssl.so

Listen 80
Listen 443

NameVirtualHost *:443
<VirtualHost _default_:443>
    ServerName server_name:443
    ErrorLog logs/https_error
    CustomLog logs/https_access common
```

```
        SSLEngine on
        SSLCertificateFile rsaserver.crt
        SSLCertificateKeyFile rsaserver.key
    </VirtualHost>
```

設定の確認

❖ **設定を確認するには、次の手順に従います。**

1 次の構文を使用してリダイレクタを呼び出します。

```
http://host:port/iaredirect/ml/
```

2 ログ・ファイルをチェックして、リダイレクタが要求をログに記録しているかどうかを確認します。

*注意：*このテストは、Mobile Link 同期サーバへの接続を作成しません。

Apache リダイレクタ

Apache リダイレクタは、Apache Web サーバのバージョン 2.0.x 用のネイティブなリダイレクタです。これは、バージョン 2.0.47 でテスト済みです。Apache リダイレクタは、Windows、Solaris、Linux でサポートされています。

Tomcat を使用している場合は、サーブレット・リダイレクタも使用できます。詳細については、「サーブレット・リダイレクタ」180 ページを参照してください。

❖ Apache リダイレクタを設定するには、次の手順に従います。

1. 「リダイレクタのプロパティの設定 (すべてのバージョン共通)」169 ページ の手順を完了します。

2. 次に示すように、*mod_iaredirect.dll* ファイルまたは *mod_iaredirect.so* ファイルを、Web サーバの適切なディレクトリにコピーします。

   - Windows の Apache の場合は、SQL Anywhere インストール環境の *MobiLink¥redirector¥apache¥v20¥* サブディレクトリに、*mod_iaredirect.dll* ファイルがあります。このファイルを、Web サーバが設定されているコンピュータの *%apache-home%¥modules* ディレクトリにコピーします。

   - Solaris または Linux の Apache の場合は、SQL Anywhere インストール環境の *MobiLink/redirector/apache/v20/OS* サブディレクトリに、*mod_iaredirect.so* ファイルがあります。ここで、*OS* は、**linux** または **solaris** です。このファイルを、Web サーバが設定されているコンピュータの *%apache-home%/modules* ディレクトリにコピーします。

3. Apache Web サーバの設定ファイル *httpd.conf* を次のように更新します。

- LoadModule セクションで、次の行を追加します。

  ```
  (Windows) LoadModule iaredirect_module
  path modules/mod_iaredirect.dll

  (Solaris and Linux) LoadModule
  iaredirect_module path modules/mod_iaredirect.so
  ```

  ここで、*path* は、Apache *modules* ディレクトリのロケーションです。

- 次のセクションをファイルに追加します。

  ```
  <Location /iaredirect/ml>
      SetHandler iaredirect-handler
      iaredirectorConfigFile location/
  redirector.config
   </Location>
  ```

  ここで、*/iaredirect/ml* はリダイレクタを起動するために使用するパスで、*location* は *redirector.config* が配置されたディレクトリです。

- Solaris または Linux で Apache を使用している場合は、作成した <Location> セクションに、次のオプションのディレクティブを追加することもできます。

  - **MaxSyncUsers *number*** リダイレクタを経由して同期する Mobile Link ユーザの最大数。この値を使用して、リダイレクタに必要なリソースが割り当てられます。この値は 60 未満であってはなりません。デフォルトは 1000 です。デフォルトのユーザ数が実際の数より少ない場合のみ、この設定を変更します。

  - **ShmemDiagnosis on|off** on に設定すると、メモリ・リソースをデバッグできます。デフォルトはオフです。

4 デバッグに役立つように、リダイレクタが出力するログ情報量を増やすことができます。このためには、*httpd.conf* 内の LogLevel ディレクティブを変更して、これを **LogLevel info** に

設定します。ログ・レベルは、debug、info、notice、warn、error、crit、alert、emerg (情報量が多いものから少ないものの順) です。

HTTPS 以外のネットワーク・プロトコルを使用している場合、これで設定は完了です。

5 HTTPS 同期を使用している場合は、次のようにしてサーバを設定します (HTTPS の設定方法は、Apache Tomcat のサーブレット・リダイレクタおよび M-Business Anywhere 用のネイティブなリダイレクタの HTTPS 手順と同じことに注意してください)。

- mod_ssl と OpenSSL のバイナリをダウンロードしてインストールします。これらのバイナリは、Apache Module Registry (http://modules.apache.org/) にあります。*mod_ssl.so* ファイルは、*%APACHE_HOME%¥modules* にコピーしてください。*libeay32.dll* ファイルと *ssleay32.dll* ファイルは、*%APACHE_HOME%¥bin* にコピーしてください。

- reqtool.exe で要求を生成し、それをサード・パーティの認証局に送信して署名を受けるか、または gencert.exe を使用して直接証明書を生成することによって、サーバ証明書とプライベート・キーを生成します。プライベート・キーは、サーバ証明書と同じファイルに保存するか、独自のファイルに保存することができます。

- 次の行を *httpd.conf* ファイルに追加します。

```
LoadModule ssl_module modules/mod_ssl.so
     SSLEngine on
     SSLCertificateFile certificate_file
```

ここで *certificate_file* は、サーバの証明書ファイルのパスとファイル名です。

サーバのプライベート・キーが、サーバの証明書と異なるファイルにある場合は、次の行を追加します。

```
SSLCertificateKeyFile private_key_file
```

ここで *private_key_file* は、サーバのプライベート・キーのパスとファイル名です。

プライベート・キーがパス・フレーズを使用して暗号化されており、win32 上で稼動している場合は、次の行を追加します。

```
SSLPassPhraseDialog exec:exe_name
```

ここで *exe_name* は、stdout にパス・フレーズを返す実行ファイルのパスとファイル名です。

または、パス・フレーズは openssl を使用してプライベート・キーから除去することも可能です。

```
openssl rsa -in src_file -out dst_file
```

ここで *src_file* は、パス・フレーズで保護されているプライベート・キーのパスとファイル名です。*dst_file* は、保護されていないプライベート・キーが含まれる出力ファイルのパスとファイル名です。ただし、こうするとサーバのセキュリティが低下することがあるので注意してください。

HTTPS の使用の詳細については、「Mobile Link トランスポート・レイヤ・セキュリティ」205 ページを参照してください。

例

次の例は、Apache Web サーバが Mobile Link 同期サーバに要求をルート指定するように設定した *httpd.conf* のセクションを示しています。`AddModule` で始まる行は、バージョン 1.3.x にのみ適用されます。この例は、Windows でのみ使用できます。UNIX と Linux の場合は、*mod_iaredirect.dll* を *mod_iaredirect.so* に変更します。

```
LoadModule iaredirect_module modules/mod_iaredirect.dll
 ...
 AddModule mod_iaredirect.c
 ...
 <Location /iaredirect/ml>
     SetHandler iaredirect-handler
     iaredirectorConfigFile c:/redirector.config
 </Location>
```

❖ 設定をテストするには、次の手順に従います。

1 次の構文を使用してリダイレクタを呼び出します。

   `http://`*host*`:`*port/iaredirect*`/ml/`

   ここで、*iaredirect/ml* は、*httpd.conf* の <Location> タグで指定したパスです。

2 ログ・ファイルをチェックして、リダイレクタが要求をログに記録しているかどうかを確認します。

   注意： このテストは、Mobile Link 同期サーバへの接続を作成しません。

M-Business Anywhere リダイレクタ

M-Business Anywhere リダイレクタは、Windows、Solaris、Linux でサポートされています。M-Business Anywhere バージョン 5.5 でテスト済みです。

❖ M-Business Anywhere リダイレクタを設定するには、次の手順に従います。

1 「リダイレクタのプロパティの設定 (すべてのバージョン共通)」169 ページ の手順を完了します。

2 *mod_iaredirect.dll* ファイルまたは *mod_iaredirect.so* ファイルを、Web サーバが設定されているコンピュータの *%avantgo-home%¥bin* ディレクトリにコピーします。このファイルは、SQL Anywhere インストール環境の *MobiLink¥redirector¥avantgo¥OS* サブディレクトリにあります。ここで、*OS* は **linux** または **solaris** です (Windows の場合は、*OS* サブディレクトリはありません)。

3 Windows の場合は、M-Business Anywhere Web サーバの *sync.conf* 設定ファイルを次のように更新します。

- LoadModule セクションで、次の行を追加します。

```
LoadModule iaredirect_module path/bin/
mod_iaredirect.dll
```

  ここで、*path* は、M-Business Anywhere *bin* ディレクトリのロケーションです。

- 次のセクションをファイルに追加します。

```
<Location /iaredirect/ml>
    SetHandler iaredirect-handler
    iaredirectorConfigFile location/
redirector.config
 </Location>
```

ここで、*/iaredirect/ml* はリダイレクタを起動するために使用するパスで、*location* は *redirector.config* が配置されたディレクトリです。

- SyncLoadFile セクションで、次の行を追加します。

```
SyncLoadFile path/bin/mod_iaredirect.dll
```

ここで、*path* は、M-Business Anywhere *bin* ディレクトリのロケーションです。

4 Solaris と Linux の場合は、M-Business Anywhere Web サーバの *sync.conf* 設定ファイルを次のように更新します。

- LoadModule セクションで、次の行を追加します。

```
LoadModule iaredirect_module pathbin/
mod_iaredirect.so
```

ここで、*path* は、M-Business Anywhere *bin* ディレクトリのロケーションです。

- 次のセクションをファイルに追加します。

```
<Location /iaredirect/ml>
    SetHandler iaredirect-handler
    iaredirectorConfigFile location/
redirector.config
 </Location>
```

ここで、*/iaredirect/ml* はリダイレクタを起動するために使用するパスで、*location* は *redirector.config* が配置されたディレクトリです。

- 作成した <Location> セクションに、次のオプションのディレクティブを追加することもできます。

  - **MaxSyncUsers *number*** リダイレクタを経由して同期する Mobile Link ユーザの最大数。この値を使用して、リダイレクタに必要なリソースが割り当てられます。この値は 60 未満であってはなりません。

デフォルトは 1000 です。デフォルトのユーザ数が実際の数より少ない場合のみ、この設定を変更します。

- **ShmemDiagnosis on|off** on に設定すると、メモリ・リソースをデバッグできます。デフォルトはオフです。

5 デバッグに役立つように、リダイレクタが出力するログ情報量を増やすことができます。このためには、*sync.conf* 内の LogLevel ディレクティブを変更して、これを **LogLevel info** に設定します。ログ・レベルは、debug、info、notice、warn、error、crit、alert、emerg (情報量が多いものから少ないものの順) です。

HTTPS 以外のネットワーク・プロトコルを使用している場合、これで設定は完了です。

6 HTTPS 同期を使用している場合は、ECC と RSA の両方のセキュリティを実装します。

詳細については、『*Administrator Guide for M-Business Server*』の「Security on Windows」または「Security on UNIX」の章を参照してください。

例

次の例は、M-Business Anywhere Web サーバが Mobile Link 同期サーバに要求を送信するように設定した *sync.conf* のセクションを示しています。

この例は、Windows で使用できます。

```
LoadModule iaredirect_module "c:¥program files¥M-
Business Anywhere¥bin¥mod_iaredirect.dll"
 ...
 SyncLoadFile "c:¥program files¥M-Business
Anywhere¥bin¥mod_iaredirect.dll"
 ...
 <Location ¥iaredirect¥ml>
     SetHandler iaredirect-handler
     iaredirectorConfigFile
"c:¥AvantGoServer¥conf¥redirector.config"
 </Location>
```

次の例は、UNIX および Linux で使用できます。

```
LoadModule iaredirect_module modules/mod_iaredirect.so
 ...
 <Location /iaredirect/ml>
     SetHandler iaredirect-handler
     iaredirectorConfigFile "/redirector.config"
 </Location>
```

❖ 設定をテストするには、次の手順に従います。

1 次の構文を使用してリダイレクタを呼び出します。

`http://`*host*`:`*port/iaredirect*`/ml/`

ここで、*iaredirect* は、*sync.conf* の <Location> タグで指定したパスです。

2 ログ・ファイルをチェックして、リダイレクタが要求をログに記録しているかどうかを確認します。

*注意：*このテストは、Mobile Link 同期サーバへの接続を作成しません。

第 9 章

現在のセッション以外での Mobile Link の実行

この章の内容

この章では、Mobile Link 同期サーバをデーモンまたはサービスとして実行する方法について説明します。

Mobile Link 同期サーバは、常時利用できるように設定できます。これを簡単に行うには、コンピュータをログオフしてもサーバは稼働し続けるように、Windows 版または UNIX 版の Mobile Link 同期サーバを実行します。これを行う方法は、使用するオペレーティング・システムによって異なります。

- **UNIX デーモン** `-ud` コマンド・ライン・オプションを使用して、Mobile Link 同期サーバをデーモンとして実行できます。これにより、Mobile Link サーバをバックグラウンドで実行させ、ログオフ後も引き続き実行させることができます。
- **Windows サービス** Windows Mobile Link サーバを、サービスとして実行させることができます。

デーモンとしての UNIX 版 Mobile Link サーバの稼働

UNIX 版 Mobile Link サーバをバックグラウンドで実行し、現在のセッションから独立して稼働させるには、これをデーモンとして起動します。

❖ **UNIX 版 Mobile Link サーバをデーモンとして稼働させるには、次の手順に従います。**

- Mobile Link サーバの起動時に、-ud オプションを使用します。次に例を示します。

```
dbmlsrv9 -c "dsn=ASA 9.0 Sample;uid=DBA;pwd=SQL" -ud
```

詳細については、「-ud オプション」264 ページを参照してください。

Windows 版 Mobile Link 同期サーバをサービスとして実行する

Windows 版 Mobile Link サーバをバックグラウンドで実行し、現在のセッションから独立して稼働させるには、これを「**サービス**」として起動します。

コマンド・ラインから、または Sybase Central の [サービス] タブで、以下のサービス管理タスクを実行できます。

- サービスの追加、編集、削除
- サービスの開始、停止、一時停止
- サービスを制御するパラメータの変更
- サービスにデータベースを追加して、同時に複数のデータベースを実行できるようにする

サービスの追加、修正、削除

Sybase Central のサービス・アイコンは、実行中、一時停止、停止を示すサーバ・アイコンを使って各サービスの現在の状況を表示します。

❖ 新しいサービスを追加するには、次の手順に従います (Sybase Central の場合)。

1. Sybase Central の左ウィンドウ枠でサーバをクリックし、右ウィンドウ枠の [サービス] タブを開きます。
2. [サービスの追加] をダブルクリックします。
3. ウィザードの指示に従います。

dbsvc ユーティリティを使用してサービスを作成することもできます。詳細については、『ASA データベース管理ガイド』> 「dbsvc コマンド・ライン・ユーティリティを使用したサービスの管理」を参照してください。

❖ サービスを削除するには、次の手順に従います (Sybase Central の場合)。

1. Sybase Central の左ウィンドウ枠でサーバをクリックし、右ウィンドウ枠の [サービス] タブを開きます。
2. 右ウィンドウ枠で、削除するサービスのアイコンを右クリックし、ポップアップ・メニューの [削除] を選択します。

❖ サービスのパラメータを変更するには、次の手順に従います。

1. Sybase Central の左ウィンドウ枠でサーバをクリックし、右ウィンドウ枠の [サービス] タブを開きます。
2. 右ウィンドウ枠で、変更するサービスを右クリックし、ポップアップ・メニューから [プロパティ] を選択します。
3. [サービス] プロパティ・シートのタブで、必要に応じてパラメータを変更します。
4. 変更が完了したら [OK] をクリックします。

サービス設定の変更は、次回のサービス実行時から有効となります。

起動オプションの設定

次のオプションは、Mobile Link サービスの起動時の動作を制御します。これらのオプションは、サービスのプロパティ・シートの [一般] タブで設定します。

- **[自動]** [自動] を選択すると、サービスは Windows オペレーティング・システムが起動すると必ず起動します。この設定は、データベース・サーバやその他の常に稼働しているアプリケーションに適しています。
- **[手動]** [手動] を選択すると、サービスは Administrator アクセス権を持つユーザが起動したときにのみ起動します。Administrator アクセス権については、Windows のマニュアルを参照してください。
- **[無効]** [無効] を選択すると、サービスは起動しません。

起動オプションは、コンピュータが次回 Windows を起動するときに適用されます。

コマンド・ライン・オプションの指定

サービス・プロパティ・シートの [設定] タブには、サービスのコマンド・ライン・オプションを入力するためのテキスト・ボックスがあります。このボックスには、実行プログラムの名前は入力しないでください。

たとえば、冗長ロギングと 3 つのワーカ・スレッドを使用する Mobile Link 同期サービスを起動するときは、パラメータ・ボックスで次のように入力します。

```
-c "dsn=ASA 9.0 Sample;uid=DBA;pwd=SQL"
 -vc
 -w 3
```

サービスのコマンド・ライン・オプションは、実行プログラムのものと同じです。Mobile Link で使用するコマンド・ライン・オプションの詳細については、「Mobile Link 同期サーバのオプション」235 ページを参照してください。

アカウント・オプションの設定

サービスを実行できるアカウントを選択できます。ほとんどのサービスは、特別なアカウントの LocalSystem で実行され、これがサービスのデフォルト・オプションになっています。[サービス] プロパティ・シートの [アカウント] タブを開き、アカウント情報を入力して、別のアカウントでログオンするようにサービスを設定できます。

LocalSystem 以外のアカウントでサービスを実行するには、そのアカウントがサービスとしてログオン権限を持っていなければなりません。この権限は、Windows のユーザー マネージャの [高度なユーザー権利] で付与できます。

サービスのアイコンがタスクバーとデスクトップのどちらに表示されるかは、選択したアカウントと、[デスクトップとの対話をサービスに許可] チェックボックスをオン／オフのどちらに設定したかによって異なります。

- サービスが LocalSystem で実行されていて、サービスのプロパティ・シートの [デスクトップとの対話をサービスに許可] チェックボックスがオンになっている場合、サービスを実行しているコンピュータの Windows NT/2000/XP にログインしたすべてのユーザのデスクトップにアイコンが表示されます。した

がって、すべてのユーザがアプリケーション・ウィンドウを開き、サービスとして実行されているプログラムを停止することができます。

- サービスが LocalSystem で実行されていて、サービスのプロパティ・シートの [デスクトップとの対話をサービスに許可] チェックボックスがオフになっている場合、ユーザのデスクトップにアイコンは表示されません。サービスの状態を変更する権限を与えられているユーザのみ、サービスを停止できます。
- サービスが他のアカウントで実行されている場合、デスクトップにアイコンは表示されません。サービスの状態を変更する権限を与えられているユーザのみ、サービスを停止できます。

実行ファイルの変更

Sybase Central でサービスに関連するプログラム実行ファイルを変更するには、[サービス] プロパティ・シートの [設定] タブをクリックし、[ファイル名] ボックスに新しいパスとファイル名を入力します。

実行ファイルを新しいディレクトリに移動する場合は、この内容も変更します。

サービスの開始、停止、一時停止

❖ **サービスを開始、停止、または一時停止するには、次の手順に従います。**

1. Sybase Central の左ウィンドウ枠でサーバをクリックし、右ウィンドウ枠の [サービス] タブを開きます。
2. サービスを右クリックし、ポップアップ・メニューから [起動]、[停止]、または [一時停止] を選択します。

   一時停止したサービスを再開するには、サービスを右クリックしてポップアップ・メニューから [続行] を選択します。

サービスを開始すると、停止するまで実行を続けます。Sybase Central を閉じたり、ログオフしてもサービスは停止しません。

サービスを停止すると、データベースへの接続はすべて閉じられ、データベース・サーバは停止されます。他のアプリケーションのプログラムは終了します。

サービスを一時停止すると、アプリケーションによる動作が行われなくなります。ただし、アプリケーションは停止せず、サーバ・サービスではデータベースへのクライアント接続は閉じられません。通常の使用では、サービスを一時停止する必要はありません。

一度に複数のサービスを実行する

Windows の [コントロール パネル] にあるサービス マネージャを使用していくつかのタスクを実行することはできますが、Windows のサービス マネージャで Mobile Link サービスをインストールしたり設定したりすることはできません。Mobile Link のすべてのサービス管理は、Sybase Central から行うことができます。

Windows の [コントロール パネル] から [サービス マネージャ] を開くと、サービスのリストが表示されます。Adaptive Server Anywhere サービスの名前は、サービスをインストールしたときに入力したサービス名に、Adaptive Server Anywhere がプレフィクスを付けたものです。インストールされたすべてのサービスが、リストに表示されます。

この項では、一度に複数のサービスを実行する方法について説明します。

サービスの依存性

状況によっては、互いに依存し合う複数の実行プログラムをサービスとして実行できます。たとえば同期を行う場合は、Mobile Link 同期サーバとデータベース・サーバを実行します。

このような場合には、サービスを正しい順序で開始してください。統合データベース・サーバが開始する前に Mobile Link 同期サービスを起動すると、Mobile Link 同期サービスは統合データベース・サーバを検出できないためにエラーになります。データベース・サーバが実行中のときに Mobile Link サーバを起動するという順序を守ってください (統合データベース・サーバが別のコンピュータ上にある場合は、この限りではありません)。

サービス・グループを使用すると、これらの問題を防ぐことができます。サービス・グループは Sybase Central で管理します。

サービス・グループ

システムの各サービスを、サービス・グループの各メンバに割り当てることができます。デフォルトでは、各サービスはグループに属しています。Mobile Link 同期サーバのデフォルトのグループは ASANYMobiLink です。

サービスが適切なグループのメンバであることを確認してから、正しい順序でサービスが開始するように設定してください。Sybase Central では、サービスが割り当てられているグループを調べ、このグループを変更することができます。

❖ サービスが割り当てられているグループを調べて変更するには、次の手順に従います。

1. Sybase Central の左ウィンドウ枠でサーバをクリックし、右ウィンドウ枠の [サービス] タブを開きます。
2. サービスを右クリックし、ポップアップ・メニューから [プロパティ] を選択します。
3. [依存性] タブをクリックします。最上部のテキスト・ボックスにサービスが割り当てられているグループの名前が表示されます。
4. [変更] をクリックして、システムの使用可能なグループのリストを表示します。
5. いずれかのグループを選択するか、新しいグループの名前を入力します。
6. [OK] をクリックして、そのグループにサービスを割り当てます。

サービスの依存性の管理

Sybase Central を使用して、サービスの依存性を指定できます。次に例を示します。

- 現在のサービスよりも前に、少なくとも 1 つのグループを開始できます。
- 現在のサービスよりも前に、すべてのサービスを開始できます。

❖ サービスまたはグループを依存性のリストに追加するには、次の手順に従います。

1. Sybase Central の左ウィンドウ枠でサーバをクリックし、右ウィンドウ枠の [サービス] タブを開きます。
2. サービスを右クリックし、ポップアップ・メニューから [プロパティ] を選択します。
3. [依存性] タブをクリックします。
4. [サービスの追加] または [サービス・グループの追加] をクリックして、サービスまたはグループを依存性リストに追加します。
5. リストからサービスまたはグループを 1 つ選択します。
6. [OK] をクリックして、そのサービスまたはグループを依存性のリストに追加します。

Mobile Link サーバ起動時のトラブルシューティング

この項では、Mobile Link サーバの起動時によく発生する問題について説明します。

ネットワーク通信ソフトウェアが実行されているかを確認する

適切なネットワーク通信ソフトウェアをインストールし、実行してから、Mobile Link サーバを実行します。信頼性の高いネットワーク・ソフトウェアを 1 つのネットワークだけが導入された状態で実行している場合は、問題はありません。Mobile Link サーバを実行する前に、ネットワーク通信を必要とする他のソフトウェアが正しく動作していること確認してください。

TCP/IP プロトコルを使用している場合は、ping と telnet が正しく動作していることを確認します。ping アプリケーションと telnet アプリケーションは、多数の TCP/IP プロトコル・スタックに付属します。

ネットワーク通信の起動時の問題をデバッグする

ネットワークで接続を確立するときに問題が生じた場合は、クライアントとサーバの両方でデバッグ・オプションを使用し、問題点を診断することができます。サーバ・ウィンドウに起動情報が表示されます。-o オプションを使用して、出力ファイルに結果のログを取ります。

第 10 章

Mobile Link トランスポート・レイヤ・セキュリティ

この章の内容

この章では、Mobile Link 同期サーバと Mobile Link クライアント間の通信を、Mobile Link トランスポート・レイヤ・セキュリティを使用して保護する方法について説明します。

個別にライセンスを取得可能なオプションが必要

トランスポート・レイヤ・セキュリティを使用するには、個別にライセンスを取得可能な SQL Anywhere Studio セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。

このコンポーネントの注文方法については、『SQL Anywhere Studio の紹介』> 「別途ライセンスが入手可能なコンポーネント」を参照してください。

概要

個別にライセンスを取得可能なオプションが必要
トランスポート・レイヤ・セキュリティを使用するには、個別にライセンスを取得可能な SQL Anywhere Studio セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。

このコンポーネントの注文方法については、『SQL Anywhere Studio の紹介』> 「別途ライセンスが入手可能なコンポーネント」を参照してください。

トランスポート・レイヤ・セキュリティは IETF 標準プロトコルであり、デジタル証明書とパブリック・キー暗号方式を使用して、クライアント／サーバ・アプリケーションを保護します。

クライアントは信用されたパブリック証明書を使用してデータを暗号化し、最初のクライアント／サーバ・ハンドシェイクでサーバを認証します。クライアントから送られてくるデータは、Mobile Link 同期サーバ証明書に保存されている、対応するプライベート・キーによってのみ複号化できます。

サーバ認証の場合は、Mobile Link 同期サーバが、そのパブリック証明書をクライアントに送信します。クライアントは、証明書のフィールドと証明書に埋め込まれているデジタル署名を使用して、サーバのアイデンティティを確認します。

効率性

トランスポート・レイヤ・セキュリティ標準は、パブリック・キー暗号方式に伴う非効率性を上回ります。安全な接続が確立されると、クライアントとサーバは共通キーを交換します。以降の通信では、非常に効率的な対称暗号が使用されます。

サポートされるプラットフォーム

FIPS 承認のセキュリティ・オプションは Windows でのみ使用できます。Ultra Light では動作しません。

FIPS 140-2 承認

連邦情報処理規格 (FIPS) 140-2 では、セキュリティ・アルゴリズムの要件を指定しています。ただし、SSL などのセキュリティ・プロトコルやトランスポート・レイヤ・セキュリティの要件は指定していません。FIPS 140-2 は、米国商務省標準技術局 (NIST：National Institute of Standards and Technology) およびカナダ通信安全保障局 (CSE：Canadian Communication Security Establishment) を通じて、米国政府とカナダ政府から付与されます。Certicom は、Windows で実装されるセキュリティ・アルゴリズムについて、FIPS 証明を取得しています。

SQL Anywhere Studio が提供するトランスポート・レイヤ・セキュリティでは、Certicom ソフトウェア内の基本となる FIPS 承認アルゴリズムを使用できます。

トランスポート・レイヤ・セキュリティを使用するには、セキュリティ・オプションを別途購入する必要があります。

トランスポート・レイヤ・セキュリティの注文方法については、『SQL Anywhere Studio の紹介』> 「別途ライセンスが入手可能なコンポーネント」を参照してください。

FIPS 承認のセキュリティ・アルゴリズムを使用すると、データベース・ファイルを暗号化したり、データベース・クライアント／サーバ通信、Web サービス、Mobile Link クライアント／サーバ通信における通信を暗号化できます。

詳細については、次の項目を参照してください。

- 『SQL Anywhere Studio セキュリティ・ガイド』> 「データベースの暗号化」
- 『SQL Anywhere Studio セキュリティ・ガイド』> 「Web サービスでのトランスポート・レイヤ・セキュリティの使用」
- 「トランスポート・レイヤ・セキュリティを使用する Mobile Link 同期サーバの起動」219 ページ
- 「トランスポート・レイヤ・セキュリティを使用する Mobile Link クライアントの設定」222 ページ

トランスポート・レイヤ・セキュリティの設定

Mobile Link トランスポート・レイヤ・セキュリティを設定するには、次の手順に従います。

- **デジタル証明書を作成する** パブリック証明書とサーバ証明書を作成します。パブリック証明書が Mobile Link クライアントに配布されるのに対して、サーバ証明書は Mobile Link 同期サーバに安全に保存されます。

  「デジタル証明書の作成」209 ページを参照してください。

- **トランスポート・レイヤ・セキュリティを指定して Mobile Link 同期サーバを起動する** dbmlsrv9 コマンド・ライン・オプションを使用して、セキュリティのタイプ、サーバ証明書、プライベート・キーを保護するパスワードを指定します。

  「トランスポート・レイヤ・セキュリティを使用する Mobile Link 同期サーバの起動」219 ページを参照してください。

- **トランスポート・レイヤ・セキュリティを使用するように Mobile Link クライアントを設定する** 信頼できるパブリック証明書のパスとファイル名を指定します。

  Mobile Link 同期クライアント・ユーティリティ (dbmlsync) または Ultra Light アプリケーションを使用して、適切なセキュリティまたはネットワーク・プロトコル・オプションを指定します。

  「トランスポート・レイヤ・セキュリティを使用する Mobile Link クライアントの設定」222 ページを参照してください。

デジタル証明書の作成

トランスポート・レイヤ・セキュリティを設定するには、デジタル証明書を生成します。

自己署名証明書の作成、エンタープライズ・ルート証明書と証明書チェーンの使用、認証局 (CA) による証明書への署名を行うことができます。

- **自己署名証明書** 自己署名証明書は、単一の Mobile Link 同期サーバで構成される単純な設定において使用されます。この場合、信頼できるパブリック証明書を作成するために使用されたプライベート・キーは、民間の認証局や専用の設備ではなく、Mobile Link 同期サーバに保存されます。

  「自己署名ルート証明書」210 ページを参照してください。

- **エンタープライズ・ルート証明書** エンタープライズ・ルート証明書を使用すると、複数のサーバが配備されている環境での、データの整合性と拡張性が向上します。

  - 信頼できるパブリック証明書を作成するために使用されるプライベート・キーは、安全な中央のロケーションに保存できます。
  - クライアントを再設定することなく、Mobile Link 同期サーバを追加できます。

  「証明書チェーン」211 ページを参照してください。

- **民間の認証局** エンタープライズ・ルート証明書の代わりに、サードパーティの認証局を使用できます。民間の認証局は、プライベート・キーを保存するための専用の設備を備えており、高品質なサーバ証明書を作成します。

  詳細については、「証明書チェーン」211 ページおよび「グローバル署名証明書」214 ページを参照してください。

証明書ユーティリティ

SQL Anywhere Studio の証明書生成ユーティリティ gencert を使用すると、証明書を作成できます。このユーティリティでは、証明書識別情報とファイル情報の入力が要求され、RSA または楕円曲線暗号化方式が使用されます。証明書読み込みユーティリティ readcert を使用すると、証明書の値を表示し、証明書チェーンを検証できます。

自己署名ルート証明書

自己署名ルート証明書は、単一の Mobile Link 同期サーバで構成される単純な設定において使用されます。この場合、信頼できるパブリック証明書を作成するために使用されたプライベート・キーは、民間の認証局や専用の設備ではなく、Mobile Link 同期サーバに保存されます。

ヒント
複数の Mobile Link 同期サーバを操作している場合、または証明書の高度な整合性が要求される場合は、エンタープライズ・レベルの証明書チェーンを使用します。

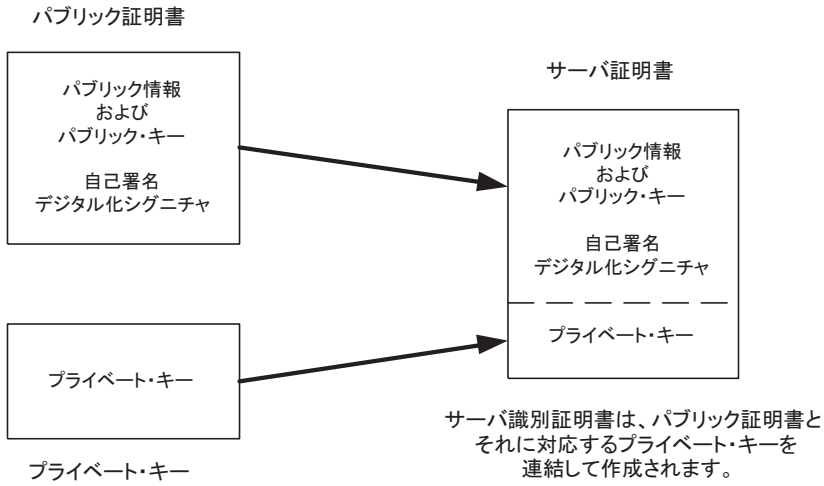
証明書チェーンの設定については、「証明書チェーン」211 ページを参照してください。

自己署名証明書を設定するには、gencert ユーティリティを使用して次の証明書を生成します。

- **パブリック証明書** 自己署名パブリック証明書は、Mobile Link クライアントに配布されます。これは、識別情報、Mobile Link 同期サーバのパブリック・キー、サーバ認証用の自己署名デジタル署名が含まれる電子文書です。

  デジタル署名とサーバ認証の詳細については、「サーバ認証」222 ページを参照してください。

- **サーバ証明書** サーバ証明書は、Mobile Link 同期サーバに安全に保存されます。これは、自己署名パブリック証明書 (Mobile Link クライアントに配布) と、対応するプライベート・キーを組み合わせたものです。プライベート・キーによって、Mobile Link 同期サーバは、Adaptive Server Anywhere または Ultra Light クライアントから送信されてくるメッセージを復号化できます。



自己署名ルート証明書を生成する方法については、『Mobile Link 管理ガイド』> 「証明書生成ユーティリティ」を参照してください。

証明書チェーン

自己署名証明書ではなく、証明書チェーンを使用すると、マルチサーバ環境のセキュリティと拡張性を向上できます。証明書チェーンでは、認証局またはエンタープライズ・ルート証明書による、Mobile Link 同期サーバ証明書への署名が必要です。

自己署名証明書の詳細については、「自己署名ルート証明書」210 ページを参照してください。

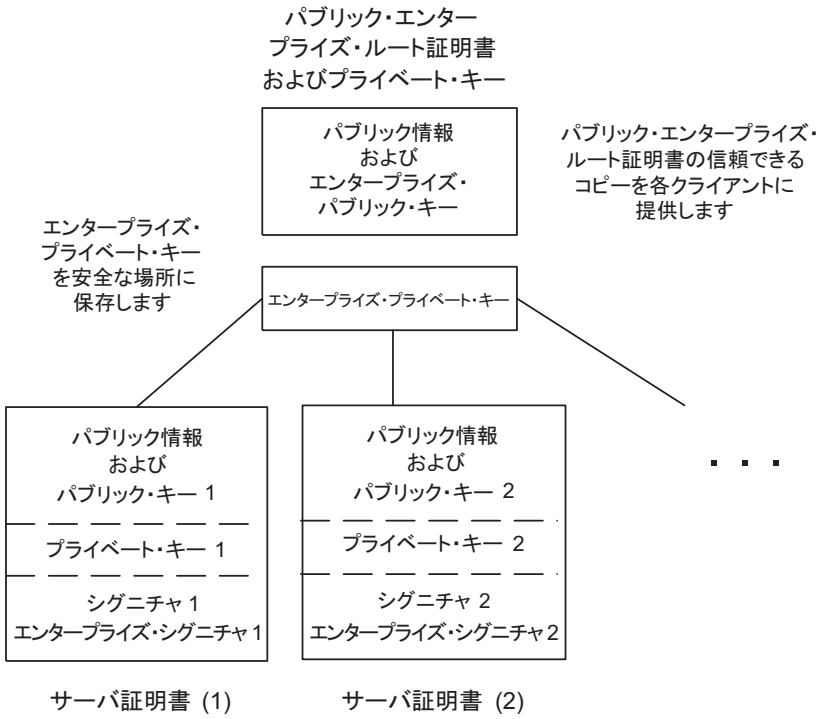
証明書チェーンを使用する利点

証明書チェーンには、次の利点があります。

- **拡張性** Mobile Link 同期クライアントが、エンタープライズ・ルート証明書または認証局によって署名されたすべての証明書を信用するように設定できます。新しい Mobile Link 同期サーバを追加しても、クライアントは新しいパブリック証明書のコピーを必要としません。

- **セキュリティ** エンタープライズ・ルート証明書のプライベート・キーは、Mobile Link 同期サーバにはありません。ルート証明書のプライベート・キーを高セキュリティのロケーションに保存したり、専用の設備を備えている認証局を使用することで、サーバ認証の整合性が保護されます。

次の図は、エンタープライブ・ルート証明書の基本アーキテクチャを示しています。



マルチサーバ環境で使用する証明書を作成するには、次の手順に従います。

- パブリック・エンタープライズ・ルート証明書およびエンタープライズ・プライベート・キーを生成します。

パブリック・エンタープライズ・ルート証明書は、Mobile Link クライアントに配布します。エンタープライズ・プライベート・キーは安全なロケーションに保存します。専用の設備の方が安全です。

- Mobile Link 同期サーバごとにサーバ証明書を生成します。

  パブリック・エンタープライズ・ルート証明書とエンタープライズ・プライベート・キーを使用して、各サーバ証明書に署名します。

サードパーティの認証局を使用して、サーバ証明書に署名することもできます。民間の認証局は、プライベート・キーを保存するための専用の設備を備えており、高品質なサーバ証明書を作成します。

詳細については、「グローバル署名証明書」214 ページを参照してください。

エンタープライズ・ルート証明書

エンタープライズ・ルート証明書を使用すると、複数のサーバが配備されている環境での、データの整合性と拡張性が向上します。

- 信頼できるパブリック証明書を作成するために使用されるプライベート・キーは、安全な中央のロケーションに保存できます。
- クライアントを再設定することなく、Mobile Link 同期サーバを追加できます。

エンタープライズ・ルート証明書を設定するには、エンタープライズ・ルート証明書と、サーバ証明書に署名するときに使用するエンタープライズ・プライベート・キーを作成します。

サーバ証明書の作成については、「署名付きのサーバ証明書」214 ページを参照してください。

エンタープライズ・ルート証明書を生成する方法については、『Mobile Link 管理ガイド』> 「証明書生成ユーティリティ」を参照してください。

署名付きのサーバ証明書

Mobile Link 同期サーバごとにサーバ証明書を生成します。これらの証明書はエンタープライズ・ルート証明書の署名を受けるため、gencert -s オプションを使用します。

署名付きサーバ証明書の生成については、『Mobile Link 管理ガイド』> 「証明書生成ユーティリティ」を参照してください。

Mobile Link 同期サーバごとに署名付きサーバ証明書を生成する方法については、『Mobile Link 管理ガイド』> 「証明書生成ユーティリティ」を参照してください。

グローバル署名証明書

民間の認証局とは、高品質の証明書の作成と、これらの証明書を使用した証明書要求への署名を事業としている組織です。

グローバル署名証明書には、次の利点があります。

- 会社内での通信の場合、共通して信用するものとして、外部の認可された認証局を使用すると、システムのセキュリティの信頼性が高まります。認証局は、署名を行ったすべての証明書の識別情報が正確であることを保証する必要があります。
- 認証局は、証明書を生成するための管理された環境と高度な方法を提供します。
- ルート証明書のプライベート・キーは、秘密にしておきます。企業内ではこの重要情報を格納するのに適した場所がない可能性がありますが、認証局では専用の設備を設計して管理できます。

グローバル署名証明書の設定

グローバル署名証明書を設定するには、次の手順に従います。

- Certicom の reqtool ツールを使用して、証明書要求を作成します。

  「reqtool を使用してグローバル証明書を取得する」215 ページを参照してください。

- 認証局を使用して、各 Mobile Link 同期サーバの証明書要求に署名します。

「グローバル証明書をサーバ証明書として使用する」216 ページ
を参照してください。

エンタープライズ・ルート証明書へのグローバル署名
エンタープライズ・ルート証明書にグローバル署名することができます。これは、認証局が、他の証明書に署名できる証明書を生成する場合のみ適用されます。

reqtool を使用してグローバル証明書を取得する

Mobile Link トランスポート・レイヤ・セキュリティは、Certicom SSL/TLS Plus ライブラリを基にしています。このライブラリは、楕円曲線証明書または RSA 証明書が必要です。グローバル証明書は正しいフォーマットで証明書を提供する認証局から入手できます。

証明書を取得するには、いくつかの方法があります。たとえば、reqtool ユーティリティを使用して取得できます。このユーティリティは、セキュリティ・コンポーネントのインストール時にインストールされます。このツールはサーバのプライベート・キーとグローバル証明書要求を作成します。

例

次の例では、楕円曲線証明書要求を作成します。

```
> 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: IAS
 Common Name: IAS_Waterloo
 Enter password to protect private key : mypwd123
 Enter file path to save request : global.req
 Enter file path to save private key :
serv1_private_key.pri
```

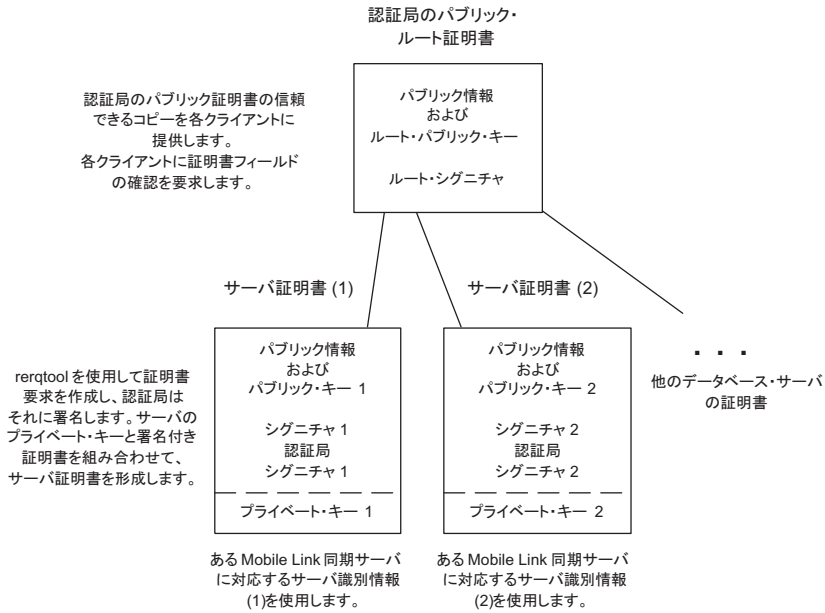
ファイル *global.req* には、パブリック証明書と要求情報があります。このファイルの内容を、証明書を発行している Web サイトのフォームにコピーします。認証局は、要求に署名し、パブリック証明書 *global.crt* を作成します。

ファイル *serv1_private_key.pri* には、対応するプライベート・キーが含まれています。このファイルは入力したパスワードで保護されますが、パスワードによる保護は弱いものなので、このファイルを安全なロケーションに格納してください。

reqtool の使用の詳細については、SQL Anywhere 9 インストール環境の *win32* サブディレクトリにあるマニュアル *reqtool.pdf* を参照してください。

グローバル証明書をサーバ証明書として使用する

グローバル署名証明書を直接、Mobile Link 同期サーバ証明書として使用できます。次の図は、マルチサーバ環境の設定を示しています。



グローバル署名証明書を直接、データベース・サーバ証明書として使用できます。次の図は、マルチサーバ環境の設定を示しています。

サーバ識別情報を作成するには、認証局の署名を受けたパブリック証明書と、reqtool ユーティリティを使用して作成したプライベート・キーを連結します。

reqtool ユーティリティの詳細については、「reqtool を使用してグローバル証明書を取得する」215 ページを参照してください。

次の例では、グローバル署名パブリック証明書 *global.crt* とプライベート・キー *serv1_private_key.pri* を連結して、サーバ証明書 *server1_certificate.crt* を作成します。

```
copy global.crt+serv1_private_key.pri
server1_certificate.crt
```

dbmlsrv9 コマンド・ラインで、サーバ証明書 *server1_certificate.crt* と、プライベート・キー *serv1_private_key.pri* のパスワードを参照します。

詳細については、「トランスポート・レイヤ・セキュリティを使用する Mobile Link 同期サーバの起動」219 ページを参照してください。

認証局のパブリック証明書を信用するようにクライアントを設定する

Mobile Link 同期サーバにアクセスするクライアントが、チェーン内のルート証明書を信用することを確認する必要があります。グローバル署名証明書の場合、ルート証明書は認証局のパブリック証明書です。

証明書フィールドの確認
グローバル署名証明書を使用する場合、各 Mobile Link クライアントはフィールド値を確認して、同じ認証局が他のクライアント用に署名した証明書を信用することを避けなければなりません。

グローバル署名証明書の証明書フィールドの確認については、「証明書フィールドの確認」223 ページを参照してください。

サーバ証明書を信用するように Mobile Link クライアントを設定する方法については、「トランスポート・レイヤ・セキュリティを使用する Mobile Link クライアントの設定」222 ページを参照してください。

グローバル署名証明書を使用して信用を確立する方法については、「グローバル署名証明書」214 ページを参照してください。

トランスポート・レイヤ・セキュリティを使用する Mobile Link 同期サーバの起動

トランスポート・レイヤ・セキュリティを使用して Mobile Link 同期サーバを起動するには、サーバ証明書と、サーバのプライベート・キーを保護するパスワードを指定します。

トランスポート・レイヤ・セキュリティを設定する手順の概要については、「トランスポート・レイヤ・セキュリティの設定」208 ページを参照してください。

TCP/IP を使用するサーバの保護

TCP/IP を使用している場合は、dbmlsrv9 -x サーバ・オプションを使用して、証明書と certificate_password パラメータを指定します。次に示すのは、dbmlsrv9 コマンド・ラインの一部です。

-x tcpip(security=*cipher***(certificate=***server-certificate***;certificate_password=***password***;...))**

- ***cipher*** RSA 暗号化の場合は **rsa_tls** を指定し、楕円曲線暗号化の場合は **ecc_tls** をそれぞれ指定します。FIPS 承認の RSA 暗号化の場合は、**rsa_tls_fips** を指定します。rsa_tls_fips の場合は、Certicom の別の FIPS 140-2 承認ソフトウェアを使用します。これは、rsa_tls (バージョン 9.0.2 以降) を使用するクライアントと互換性があり、rsa_tls_fips を使用するクライアントは、rsa_tls を使用するサーバとも互換性があります。rsa_tls_fips は、Windows 上の Adaptive Server Anywhere データベースでのみ使用できます。

  cipher は、証明書を作成するときに使用される暗号化と一致する必要があります。

- ***server-certificate*** サーバ証明書のパスとファイル名を指定します。

  サーバ証明書の作成については、「デジタル証明書の作成」209 ページを参照してください。サーバ証明書は、自己署名証明書、または認証局やエンタープライズ・ルート証明書の署名を受けた証明書のいずれかです。

- ***password*** サーバ証明書のプライベート・キーのパスワードを指定します。このパスワードは、サーバ証明書を作成するときに指定します。

dbmlsrv9 -x オプションの詳細については、「-x オプション」269 ページを参照してください。

HTTPS を使用するサーバの保護

HTTPS を使用している場合は、dbmlsrv9 -x オプションを使用して、証明書と certificate_password パラメータを直接指定します。次に示すのは、dbmlsrv9 コマンド・ラインの一部です。

-x *protocol***(certificate=***server-certificate***;certificate_password=***password***;...))**

- ***protocol*** **https**、または FIPS 承認の RSA 暗号化の場合は **https_fips** を指定します。https_fips は、Certicom の FIPS 140-2 承認ソフトウェアという別のソフトウェアを使用しますが、https (およびバージョン 9.0.2 以降) を使用するクライアントと互換性があります。https_fips は、Windows 上の Adaptive Server Anywhere データベースでのみ使用できます。

- ***server-certificate*** サーバ証明書のパスとファイル名を指定します。

  HTTPS では、RSA 証明書を使用する必要があります。

  サーバ証明書の作成については、「デジタル証明書の作成」209 ページを参照してください。

- ***password*** サーバ証明書のプライベート・キーのパスワードを指定します。このパスワードは、サーバ証明書を作成するときに指定します。

dbmlsrv9 -x オプションの詳細については、「-x オプション」269 ページを参照してください。

静的型 Java Ultra Light アプリケーション

静的型 Java Ultra Light では、トランスポート・レイヤ・セキュリティ用の別のプロトコルを使用します。

静的型 Java Ultra Light でのトランスポート・レイヤ・セキュリティの使用については、『Ultra Light 静的型 Java ユーザーズ・ガイド』> 「トランスポート・レイヤ・セキュリティ」を参照してください。

例

次の例では、セキュリティのタイプ (RSA)、サーバ証明書、サーバのプライベート・キーを保護するパスワードを指定します。

```
dbmlsrv9 -c "dsn=my_cons"
  -x
tcpip(security=rsa_tls(certificate=c:¥test¥serv_rsa1.cr
t;certificate_password=pwd))
```

楕円曲線証明書の場合は、次のように入力します。

```
dbmlsrv9 -c "dsn=my_cons"
  -x
tcpip(security=ecc_tls(certificate=c:¥test¥serv_ecc1.cr
t;certificate_password=pwd))
```

dbmlsrv9 -x オプションの詳細については、「-x オプション」269 ページを参照してください。

サーバ証明書 (この例では *serv1.crt*) の作成については、「デジタル証明書の作成」209 ページを参照してください。

設定ファイルとファイル非表示ユーティリティ dbfhide を使用して、コマンド・ライン・オプションを非表示にすることができます。詳細については、「@data オプション」242 ページを参照してください。

トランスポート・レイヤ・セキュリティを使用する Mobile Link クライアントの設定

Mobile Link トランスポート・レイヤ・セキュリティを使用するように Adaptive Server Anywhere クライアントまたは Ultra Light クライアントを設定できます。各クライアントに対して、信用されたパブリック証明書、暗号化のタイプ、ネットワーク・プロトコルを指定します。

トランスポート・レイヤ・セキュリティを設定する手順の概要については、「トランスポート・レイヤ・セキュリティの設定」208 ページを参照してください。

サーバ認証

リモート・クライアントは、サーバ認証を使用することで、Mobile Link 同期サーバのアイデンティティを確認できます。デジタル署名と証明書フィールドの確認が一緒に機能して、サーバ認証が実現します。

デジタル署名

Mobile Link 同期サーバ証明書には、データの整合性を維持し、不正侵入を防ぐための、複数のデジタル署名が含まれています。デジタル署名の作成は、次の手順で行われます。

- 証明書で実行されるアルゴリズムが、ユニークな値またはハッシュを生成します。
- 証明書への署名または認証局のプライベート・キーを使用して、ハッシュが暗号化されます。
- デジタル署名と呼ばれる暗号化ハッシュが、証明書に埋め込まれます。

デジタル署名は、自己署名、あるいはエンタープライズ・ルート証明書または認証局の署名を受けています。

Mobile Link クライアントが Mobile Link 同期サーバにアクセスする場合、各クライアントがトランスポート・レイヤ・セキュリティを使用するように設定されていると、サーバはそのパブリック証明書のコピーをクライアントに送信します。クライアントは、証明書に含まれているサーバのパブリック・キーを使用して証明書のデジタル署名を復号化し、証明書の新しいハッシュを算出して、2 つの値を比較します。値が一致する場合は、サーバの証明書の整合性が確認されます。

自己署名証明書の詳細については、「自己署名ルート証明書」210 ページを参照してください。

エンタープライズ・ルート証明書と認証局の詳細については、「証明書チェーン」211 ページを参照してください。

証明書フィールドの確認

グローバル署名証明書を使用する場合、各クライアントは証明書のフィールド値を確認して、同じ認証局が他のクライアント用に署名した証明書を信用することを避けなければなりません。この問題を解決するには、証明書の識別情報部分のフィールド値をテストするようにクライアントに要求します。認証局は、署名を行ったすべての証明書の識別情報が正確であることを保証する必要があります。

グローバル署名証明書の詳細については、「グローバル署名証明書」214 ページを参照してください。

gencert ユーティリティを使用して証明書を作成する場合は、組織、組織単位、通称のフィールドに値を入力します。対応する Mobile Link クライアント接続パラメータを使用して、これらのフィールドを確認します。

- **組織** 組織フィールドは、certificate_company Mobile Link クライアント接続パラメータに対応します。

  『Mobile Link クライアント』> 「certificate_company」を参照してください。

- **組織単位** 組織単位フィールドは、certificate_unit Mobile Link クライアント接続パラメータに対応します。

  『Mobile Link クライアント』> 「certificate_unit」を参照してください。

- **通称** 通称フィールドは、certificate_name Mobile Link クライアント接続パラメータに対応します。

  『Mobile Link クライアント』> 「certificate_name」を参照してください。

  通称フィールドは、certificate_name 暗号化接続パラメータに対応します。

Mobile Link 同期クライアントの設定については、次の項目を参照してください。

- 「トランスポート・レイヤ・セキュリティを使用する Ultra Light クライアントの設定」228 ページ
- 「クライアント・セキュリティ・オプション」224 ページ

デジタル証明書の作成については、「デジタル証明書の作成」209 ページを参照してください。

クライアント・セキュリティ・オプション

Adaptive Server Anywhere および Ultra Light は、共通の接続パラメータ・セットを使用して、トランスポート・レイヤ・セキュリティを設定します。

trusted_certificates パラメータ

Mobile Link クライアントは、trusted_certificates 接続パラメータを使用して、信用された Mobile Link 同期サーバ証明書を指定します。信用された証明書は、サーバの自己署名パブリック証明書、パブリック・エンタープライズ・ルート証明書、民間の証明局に属するパブリック証明書のいずれかです。

詳細については、次の項目を参照してください。

- 『Mobile Link クライアント』> 「trusted_certificates」
- 「デジタル証明書の作成」209 ページ

証明書フィールドの確認

証明書フィールドを確認するには、certificate_company、certificate_unit、certificate_name の各接続パラメータを使用します。これは、サーバ認証の重要な手順です。サードパーティの認証局を使用して証明書にグローバル署名している場合は、証明書フィールドを確認してください。

証明書フィールドの確認については、次の項目を参照してください。

- 「証明書フィールドの確認」223 ページ
- 「グローバル署名証明書」214 ページ
- 「サーバ認証」222 ページ

トランスポート・レイヤ・セキュリティを使用する Adaptive Server Anywhere クライアントの設定

ここでは、HTTPS または TCP/IP でトランスポート・レイヤ・セキュリティを使用する Adaptive Server Anywhere クライアントの設定方法について説明します。

HTTPS におけるトランスポート・レイヤ・セキュリティ

Mobile Link トランスポート・レイヤ・セキュリティは、Mobile Link HTTPS プロトコルの本来の機能です。HTTPS でトランスポート・レイヤ・セキュリティを使用するには、ADR 拡張オプションを使用して、trusted_certificates 接続パラメータを指定します。次に、dbmlsync コマンド・ラインの一部を示します。

-e "ctp=*protocol***;adr=trusted_certificates=***public-certificate***..."**

- ***protocol*** **https**、または FIPS 承認の RSA 暗号化の場合は **https_fips** を指定します。https_fips は、Certicom の FIPS 140-2 承認ソフトウェアという別のソフトウェアを使用しますが、https およびバージョン 9.0.2 以降を使用する Mobile Link 同期サーバと互換性があります。
- ***public-certificate*** 信用されたパブリック証明書のパスとファイル名を指定します。

HTTPS または HTTPS_FIPS の場合は、RSA 暗号化を使用して作成した証明書を使用してください。

trusted_certificates および他のクライアント・セキュリティ・パラメータの詳細については、「クライアント・セキュリティ・オプション」224 ページを参照してください。

パブリック証明書の作成または取得の詳細については、「デジタル証明書の作成」209 ページを参照してください。

HTTPS パラメータの詳細については、『Mobile Link クライアント』>「CommunicationAddress (adr) 拡張オプション」の HTTPS の項目を参照してください。

例

次の例では、dbmlsync コマンド・ラインで、HTTPS を使用する RSA セキュリティを指定します。

```
dbmlsync -c "eng=rem1;uid=dba;pwd=mypwd"
   -e
"ctp=https;adr=trusted_certificates=c:¥temp¥public_cert
.crt;certificate_company=Sybase,
Inc.;certificate_unit=IAS;certificate_name=MobiLink)"
```

CREATE SYNCHRONIZATION SUBSCRIPTION 文または ALTER SYNCHRONIZATION SUBSCRIPTION 文を使用して、CommunicationAddress 拡張オプションを指定することもできます。この方法でも同じ情報が提供されますが、その情報はデータベースに保存されます。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO pub1
  FOR user1
  ADDRESS
'trusted_certificates=c:¥temp¥public_cert.crt;
      certificate_company=Sybase, Inc.;
      certificate_unit=IAS;
   certificate_name=MobiLink'
  OPTION scriptversion='ver1';
```

TCP/IP におけるトランスポート・レイヤ・セキュリティ

Adaptive Server Anywhere クライアントが、TCP/IP でトランスポート・レイヤ・セキュリティを使用するように設定するには、CommunicationAddress (adr) 拡張オプションを使用して、信用された証明書を指定します。

trusted_certificates および証明書フィールドを直接指定できる HTTPS とは異なり、TCP/IP の場合は、セキュリティ・オプションと次の構文を使用します。

adr=security=*cipher***(trusted_certificates=***public-certificate***; ...)**

- ***cipher*** RSA 暗号化の場合は **rsa_tls** を指定し、楕円曲線暗号化の場合は **ecc_tls** をそれぞれ指定します。

  FIPS 承認の RSA 暗号化の場合は、**rsa_tls_fips** を指定します。rsa_tls_fips は別の承認ライブラリを使用しますが、rsa_tls を使用するサーバと互換性があります。rsa_tls を使用するクライアントは、rsa_tls_fips を使用するサーバと互換性があります。rsa_tls_fips は、Windows 上の SQL Anywhere Studio データベースで使用できます。

  cipher は、証明書を作成するときに使用される暗号化と一致する必要があります。

- ***public-certificate*** 信用されたパブリック証明書のパスとファイル名を指定します。

trusted_certificates および他のクライアント・セキュリティ・パラメータの詳細については、「クライアント・セキュリティ・オプション」224 ページを参照してください。

パブリック証明書の作成または取得の詳細については、「デジタル証明書の作成」209 ページを参照してください。

セキュリティ・パラメータのリストや CommunicationAddress (adr) 拡張オプションの詳細については、『Mobile Link クライアント』> 「CommunicationAddress (adr) 拡張オプション」を参照してください。

例

次の例では、dbmlsync コマンド・ラインで、RSA セキュリティと TCP/IP を指定します。

```
dbmlsync -c "eng=rem1;uid=myuid;pwd=mypwd"
   -e
"ctp=tcpip;adr=port=3333;security=rsa_tls(trusted_certi
ficates=c:¥test¥public_cert.crt;certificate_company=Syb
ase,
Inc.;certificate_unit=IAS;certificate_name=MobiLink)"
```

CREATE SYNCHRONIZATION SUBSCRIPTION 文または ALTER SYNCHRONIZATION SUBSCRIPTION 文を使用して、CommunicationAddress 拡張オプションを指定することもできます。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO pub1
  FOR user1
  ADDRESS
'port=3333;security=rsa_tls(trusted_certificates=public
_cert.crt;
      certificate_company=Sybase, Inc.;
      certificate_unit=IAS;
   certificate_name=MobiLink )'
  OPTION scriptversion='ver1';
```

詳細については、『Mobile Link クライアント』> 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」および『Mobile Link クライアント』> 「ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」を参照してください。

トランスポート・レイヤ・セキュリティを使用する Ultra Light クライアントの設定

ここでは、HTTPS または TCP/IP でトランスポート・レイヤ・セキュリティを使用する Ultra Light クライアントの設定方法について説明します。

HTTPS におけるトランスポート・レイヤ・セキュリティ

Mobile Link トランスポート・レイヤ・セキュリティは、Mobile Link HTTPS プロトコルの本来の機能です。HTTPS および Ultra Light クライアントを使用する場合は、信用された証明書と、ネットワーク・プロトコル・オプションとして、証明書フィールドを直接指定できます。

HTTPS では、RSA 証明書を使用する必要があります。

Ultra Light インタフェースで HTTPS プロトコルを指定する方法については、『Mobile Link クライアント』> 「HTTPS プロトコル・オプション」を参照してください。

ul_synch_info の security_stream および security_parms セキュリティ同期パラメータは、TCP/IP でのみ使用可能であり、HTTPS を使用している場合は効果がありません。

セキュリティ同期パラメータの詳細については、『Mobile Link クライアント』> 「Security 同期パラメータ」および「TCP/IP におけるトランスポート・レイヤ・セキュリティ」231 ページを参照してください。

❖ HTTPS でトランスポート・レイヤ・セキュリティを使用するように Ultra Light クライアントを設定するには、次の手順に従います。

1. 信用されたルート証明書を Ultra Light スキーマで指定します。

   - **静的 Ultra Light インタフェース** Ultra Light 静的型 C/C++、Embedded SQL、静的型 Java を含む静的 Ultra Light インタフェースの場合は、ulgen -r オプションを使用します。次の例では、静的型 C/C++ Ultra Light に対して、信用されたルート証明書 *ent_root.crt* を指定します。

     ```
     ulgen -a -c "dsn=myCons" -j Product -s ISampleSQL
        -f SampleDB -r ent_root.crt
     ```

     ulgen -r オプションの詳細については、『Ultra Light データベース・ユーザーズ・ガイド』> 「Ultra Light ジェネレータ」を参照してください。

- **Ultra Light コンポーネント・インタフェース** Ultra Light コンポーネント・インタフェースの場合は、Schema Painter を使用して、信用された証明書を指定します。Schema Painter の [データベース・スキーマのプロパティ] ダイアログには、[証明] タブがあります。

  詳細については、『SQL Anywhere Studio ヘルプ』> 「[データベース・スキーマ] プロパティ・シート : [証明] タブ」を参照してください。

2 同期用の HTTPS プロトコルを指定します。

次の例では、C/C++ Ultra Light の ULHTTPSStream() 関数を使用します。

```
auto ul_synch_info synch_info;
 conn.InitSynchInfo( &synch_info );
 synch_info.user_name = UL_TEXT( "50" );
 synch_info.version = UL_TEXT( "ul_default" );
 ...
 synch_info.stream = ULHTTPSStream();
 ...
```

ULHTTPSStream() の詳細については、『Ultra Light C/C++ ユーザーズ・ガイド』> 「ULHTTPSStream 関数」を参照してください。

3 HTTPS プロトコル・オプションを指定します。

次の例では、ul_synch_info の stream_parms フィールドを使用して標準の HTTPS パラメータを指定し、C/C++ Ultra Light の証明書フィールドを確認します。

```
auto ul_synch_info synch_info;
 ...
 synch_info.stream = ULHTTPSStream();
 synch_info.stream_parms = TEXT(
       "port=9999;
        certificate_company=Sybase, Inc.;
        certificate_unit=IAS;
        certificate_name=MobiLink");
     synch_info.security_stream = NULL;
     synch_info.security_parms = NULL;
```

certificate_company、certificate_unit、certificate_name の各パラメータは、証明書のフィールドを確認するために使用されています。

証明書フィールドの確認については、「証明書フィールドの確認」223 ページを参照してください。

trusted_certificates HTTPS プロトコル・オプションを指定することもできます。このオプションを指定すると、Ultra Light スキーマに埋め込まれている、信用された証明書情報 (手順 1) が上書きされます。このパラメータは、Windows NT/2000/XP または Windows CE でのみ使用できます。

```
auto ul_synch_info synch_info;
 ...
 synch_info.stream = ULHTTPSStream();
 synch_info.stream_parms = TEXT(
       "port=9999;
        trusted_certificates=¥rsaroot.crt;
        certificate_company=Sybase, Inc.;
        certificate_unit=IAS;
        certificate_name=MobiLink");
     synch_info.security_stream = NULL;
     synch_info.security_parms = NULL;
```

HTTPS オプションの詳細については、『Mobile Link クライアント』> 「HTTPS プロトコル・オプション」を参照してください。

TCP/IP におけるトランスポート・レイヤ・セキュリティ

Ultra Light クライアントが、TCP/IP でトランスポート・レイヤ・セキュリティを使用するように設定するには、信用されたルート証明書と適切なセキュリティ・ストリームを指定します。

❖ TCP/IP でトランスポート・レイヤ・セキュリティを使用するように Ultra Light クライアントを設定するには、次の手順に従います。

1. 信用されたルート証明書を Ultra Light スキーマで指定します。

- **静的 Ultra Light インタフェース** Ultra Light 静的型 C/C++、Embedded SQL、静的型 Java を含む静的 Ultra Light インタフェースの場合は、ulgen -r オプションを使用します。次の例では、静的型 C/C++ Ultra Light に対して、信用されたルート証明書を指定します。

```
ulgen -a -c "dsn=myCons" -j Product -s ISampleSQL
   -f SampleDB -r ent_root.crt
```

  ulgen -r オプションの詳細については、『Ultra Light データベース・ユーザーズ・ガイド』> 「Ultra Light ジェネレータ」を参照してください。

- **Ultra Light コンポーネント・インタフェース** Ultra Light コンポーネント・インタフェースの場合は、Schema Painter を使用して、信用された証明書を指定します。Schema Painter の [データベース・スキーマのプロパティ] ダイアログには、[証明] タブがあります。

  詳細については、『SQL Anywhere Studio ヘルプ』> 「[データベース・スキーマ] プロパティ・シート : [証明] タブ」を参照してください。

2 プログラミング・インタフェースで、適切なセキュリティまたはプロトコル・オプションを指定します。

- **静的型 Java Ultra Light** 静的型 Java Ultra Light では、トランスポート・レイヤ・セキュリティ用に個別のストリームが必要であり、個別のセキュリティ・パラメータは提供しません。次の例では、UlSecureSocketStream を、楕円曲線セキュリティのプロトコルとして使用します。

```
UlSynchOptions opts = new UlSynchOptions();
 opts.setStream(new UlSecureSocketStream() );
 opts.setStreamParms( "host=myserver;"
    + "port=2439;"
    + "certificate_company=Sybase Inc.;"
    + "certificate_unit="IAS;"
    + "certificate_name=Mobilink");
 // set other options here
 conn.synchronize( opts );
```

静的型 Java Ultra Light でのトランスポート・レイヤ・セキュリティの使用については、『Ultra Light 静的型 Java ユーザーズ・ガイド』> 「トランスポート・レイヤ・セキュリティ」を参照してください。

- **セキュリティ同期パラメータを使用する Ultra Light インタフェース** 静的型 Ultra Light C++、Embedded SQL、C++ Ultra Light コンポーネントを含む、該当する Ultra Light インタフェースに対して、セキュリティ同期パラメータを設定します。他のコンポーネント・インタフェースの場合は、トランスポート・レイヤ・セキュリティには HTTPS を使用してください。

  ULSecureRSATLSStream() または ULSecureCerticomTLSStream() を使用して、RSA セキュリティまたは楕円曲線セキュリティをそれぞれ指定できます。

  詳細については、『Mobile Link クライアント』> 「Security 同期パラメータ」を参照してください。

  次の例では、C/C++ Ultra Light に対して、RSA セキュリティと TCP/IP を指定します。

```
ul_synch_info synch_info;
 ...
 synch_info.stream = ULSocketStream();
 synch_info.security = ULSecureRSATLSStream();
 synch_info.security_parms = TEXT(
       "port=9999;
        certificate_company=Sybase, Inc.;
        certificate_unit=IAS;
        certificate_name=MobiLink");
```

  certificate_company、certificate_unit、certificate_name の各パラメータは、証明書のフィールドを確認するために使用されています。

  証明書フィールドの確認については、「証明書フィールドの確認」223 ページを参照してください。

trusted_certificates セキュリティ・パラメータを指定することもできます。このパラメータを指定すると、Ultra Light スキーマに埋め込まれている、信用された証明書情報 (手順 1) が上書きされます。このパラメータは、Windows NT/2000/XP または Windows CE でのみ使用できます。

```
ul_synch_info synch_info;
 ...
 synch_info.stream = ULSocketStream();
 synch_info.security = ULSecureRSATLSStream();
 synch_info.security_parms = TEXT(
       "port=9999;
        trusted_certificates=¥rsaroot.crt;
        certificate_company=Sybase, Inc.;
        certificate_unit=IAS;
        certificate_name=MobiLink");
```

ul_synch_info の security_parms フィールドの詳細については、『Ultra Light C/C++ ユーザーズ・ガイド』> 「security_parms 同期パラメータ」を参照してください。

- **ActiveSync を使用する Ultra Light インタフェース** トランスポート・レイヤ・セキュリティ と ActiveSync については、『Mobile Link クライアント』> 「ActiveSync プロトコル・オプション」を参照してください。
- **HotSync を使用する Ultra Light インタフェース** トランスポート・レイヤ・セキュリティ と HotSync については、『Mobile Link クライアント』> 「HotSync プロトコル・オプション」を参照してください。

第 11 章

Mobile Link 同期サーバのオプション

この章の内容

この章では、Mobile Link 同期サーバ dbmlsrv9 の起動時に設定できるオプションについて説明します。

Mobile Link 同期サーバ

Mobile Link 同期サーバを使用して、リモート・データベースまたはアプリケーションと ODBC 準拠の統合データベースの同期を行うことができます。

機能

Mobile Link 同期サーバを起動します。

構文

dbmlsrv9 -c "*connection-string***"** [*options*]

オプション	説明
@data	指定された環境変数または設定ファイルからオプションを読み込む。「@data オプション」242 ページを参照してください。
-a	同期エラー時の自動再接続を無効にする。「-a オプション」242 ページを参照してください。
-b	文字列のブランクの埋め込みを削除する。「-b オプション」243 ページを参照してください。
-bc *size*	BLOB のキャッシュ用に予約するメモリ容量を指定する。「-bc オプション」244 ページを参照してください。
-bn *size*	競合の検出のため BLOB を比較するときに考慮する最大バイト数を指定する。「-bn オプション」244 ページを参照してください。
-c *"keyword=value; …"*	統合データベース用の ODBC データベース接続パラメータを指定する。「-c オプション」245 ページを参照してください。
-cn *connections*	統合データベース・サーバとの最大同時接続数を設定する。「-cn オプション」246 ページを参照してください。

オプション	説明
-cr *count*	データベース接続リトライの最大回数を設定する。「-cr オプション」246 ページを参照してください。
-ct *connection-timeout*	接続が使用されなくなってからタイムアウトになるまでの時間を設定する。「-ct オプション」246 ページを参照してください。
-d *number*	ダウンロード・キャッシュのサイズを指定する。「-d オプション」247 ページを参照してください。
-dd *directory*	ダウンロード・ストリームの格納場所を指定する。「-dd オプション」247 ページを参照してください。
-dl	コンソールにすべてのログ・メッセージを表示する。「-dl オプション」248 ページを参照してください。
-ds *size*	再起動可能なダウンロード・ストリームの格納に使用できるディスク領域の容量を指定する。「-ds オプション」248 ページを参照してください。
-e *filename*	送信されたリモート・エラー・ログを指定のファイルに格納する。「-e オプション」248 ページを参照してください。
-et *filename*	ファイルをトランケートし、リモート同期ログを新しいファイルに追加する。「-et オプション」249 ページを参照してください。
-f	同期スクリプトに変更がないものとみなす。「-f オプション」249 ページを参照してください。
-fr	テーブル・データ・スクリプトがない場合は、同期をアボートせず、エラーの発行だけを行う。「-fr オプション」250 ページを参照してください。

オプション	説明
-m [*filename*]	QAnywhere のメッセージ機能を有効にする。「-m オプション」251 ページを参照してください。
-notifier	サーバ起動同期用に Notifier を起動する。「-notifier オプション」253 ページを参照してください。
-o *logfile*	ファイルにメッセージのログを取る。「-o オプション」254 ページを参照してください。
-on *size*	ログ・ファイルの最大サイズを設定する。「-on オプション」255 ページを参照してください。
-oq	起動エラーの発生時にポップアップ・ダイアログを表示しない。「-oq オプション」256 ページを参照してください。
-os *size*	出力ファイルの最大サイズ。「-os オプション」256 ページを参照してください。
-ot *logfile*	先にトランケートしてから、ファイルにメッセージのログを取る。「-ot オプション」257 ページを参照してください。
-ps *num*	接続ごとに、キャッシュする準備文の最大数を設定する。「-ps オプション」257 ページを参照してください。
-q	同期サーバのウィンドウを最小化する。「-q オプション」258 ページを参照してください。
-r *retries*	デッドロックされたアップロードを、この回数までリトライする。「-r オプション」258 ページを参照してください。

オプション	説明
-rd *delay*	デッドロックされたトランザクションをリトライするまでの最大遅延秒数を設定する。「-rd オプション」258 ページを参照してください。
-s *count*	一度にフェッチされ、送信されるローの最大数を指定する。「-s オプション」259 ページを参照してください。
-sl dnet *script-options*	.NET CLR のオプションを設定し、起動時に仮想マシンを強制的にロードする。「-sl dnet オプション」259 ページを参照してください。
-sl java *script-options*	Java 仮想マシンのオプションを設定し、起動時に仮想マシンを強制的にロードする。「-sl java オプション」261 ページを参照してください。
-t *ODBC-output-file*	このファイルに Mobile Link が発行する ODBC 呼び出しのログを取る。「-t オプション」263 ページを参照してください。
-tt *ODBC-output-file*	このファイルに Mobile Link が発行する ODBC 呼び出しのログを取る。ただし、すでにファイルが存在する場合は、最初にそのファイルを削除します。「-tt オプション」263 ページを参照してください。
-u *size*	アップロード・ストリームのキャッシュ用に予約するメモリ容量を指定する。「-u オプション」264 ページを参照してください。
-ud	UNIX プラットフォーム上でデーモンとして実行する。「-ud オプション」264 ページを参照してください。
-us	Mobile Link がアップロード内容のないテーブルのアップロード・スクリプトを呼び出さないようにする。「-us オプション」265 ページを参照してください。

オプション	説明
-v [*levels*]	ログ・ファイルに書き込まれるメッセージのタイプを制御する。「-v オプション」265 ページを参照してください。
-w *count*	ワーカ・スレッド数を設定する。「-w オプション」266 ページを参照してください。
-wu *count*	アップロード処理の同時実行を許可されるワーカ・スレッドの最大数を設定する。「-wu オプション」268 ページを参照してください。
-x *protocol*[**(***network-parameters***)**]	通信プロトコルを指定する。オプションで、*parameter=value* の形式でネットワーク・パラメータを指定します。複数のパラメータを指定する場合はセミコロンで区切ります。「-x オプション」269 ページを参照してください。
-za	アクティブなスクリプトを生成できるようにする。「-za オプション」275 ページを参照してください。
-ze	サンプル・スクリプトを生成できるようにする。「-ze オプション」277 ページを参照してください。
-zp	統合データベースとリモート・データベース間にタイムスタンプの競合がある場合に、最小精度よりも高い精度を持つタイムスタンプ値を競合検出に使用する。「-zp オプション」278 ページを参照してください。
-zs *name*	サーバ名を指定する。「-zs オプション」278 ページを参照してください。
-zt *number*	Mobile Link 同期サーバを実行するのに使用されるプロセッサの最大数を指定する。「-zt オプション」279ページを参照してください。

オプション	説明
-zu { **+** | **-** }	authenticate_user スクリプトが未定義の場合に、ユーザの自動的な追加を制御する。「-zu オプション」279 ページを参照してください。
-zw 1,…5	表示する警告メッセージのレベルを制御する。「-zw オプション」279 ページを参照してください。
-zwd *code*	特定の警告コードを無効にする。「-zwd オプション」280 ページを参照してください。
-zwe *code*	特定の警告コードを有効にする。「-zwe オプション」281 ページを参照してください。

説明

Mobile Link 同期サーバは、統合データベース・サーバとの接続を ODBC を介して開きます。その後、クライアント・アプリケーションからの接続を受け入れ、同期処理を制御します。

Mobile Link 同期サーバには、さまざまなデータベース管理システムとの互換性があります。たとえば、Adaptive Server Anywhere、Adaptive Server Enterprise、Oracle、Microsoft SQL Server、IBM DB2 などです。

–c オプションを使用して統合データベースの接続パラメータを指定してください。コマンド・ライン・オプションは、任意の順序で指定できます。ここでは、便宜上、–c オプションをコマンド文字列の最初の項目としています。接続文字列の前であれば、オプション・リストのどこにでも指定できます。

ODBC データ・ソースが統合データベースを自動的に起動するように設定していない場合には、統合データベースを実行してから Mobile Link サーバを起動してください。

dbmlsrv9 オプション

この項では、Mobile Link 同期サーバのすべてのコマンド・ライン・オプションについて説明します。

@data オプション

機能

指定された環境変数または設定ファイルからオプションを読み込みます。

構文

dbmlsrv9 -c "*connection-string***" @***data* …

説明

このオプションを使用すると、指定された環境変数または設定ファイルから dbmlsrv9 コマンド・ライン・オプションを読み込むことができます。指定された名前と同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。

設定ファイルの詳細については、『ASA データベース管理ガイド』> 「設定ファイルの使用」を参照してください。

設定ファイル内のパスワードやその他の情報を保護する場合は、ファイル非表示ユーティリティを使用すると、設定ファイルの内容を難読化できます。

『ASA データベース管理ガイド』> 「dbfhide コマンド・ライン・ユーティリティを使用してファイル内容を隠す」を参照してください。

-a オプション

機能

Mobile Link 同期サーバに対して同期エラーの発生時に再接続しないように指示します。

構文

dbmlsrv9 -c "*connection-string***" -a** …

説明

同期中にエラーが発生すると、Mobile Link 同期サーバは自動的に統合データベースとの接続を切断してから、接続を再確立します。再接続が行われると、認識されている状態から確実に次の同期が開始します。このような動作が不要な場合は、このオプションを使用して無効にしてください。ステータス情報の管理は、プログラマの要求や、DBMS を処理する Mobile Link スクリプトの設定方法に応じて異なり

ます。このことは、Oracle や Adaptive Server Anywhere のデータベース、またはサポートされている他の製品にもあてはまります。クライアント・アプリケーションによっては、一部のステータス情報を初期化し直す操作が必要になることがあります。

-b オプション

機能

VARCHAR、CHAR、LONG、VARCHAR、または LONG CHAR 型のカラムの場合、同期中に文字列から後続ブランクを削除します。

構文

dbmlsrv9 -c "*connection-string***" -b** …

説明

このオプションは、Adaptive Server Anywhere の CHAR データ型と統合データベースが使用する CHAR または VARCHAR データ型の違いを解決するためのものです。Adaptive Server Anywhere の CHAR データ型は VARCHAR と同じです。しかし、Adaptive Server Anywhere ではないほとんどの統合データベースで、CHAR(n) データ型は n 文字までブランクが埋め込まれます。

-b が指定されていると、Mobile Link 同期サーバは、リモートのカラムが文字列の場合に CHAR、VARCHAR、LONG CHAR、または LONG VARCHAR 型のカラムに対して文字列から後続ブランクを削除します。これを実行した後に、現在の同期でアップロードされたローをフィルタします。トリムされたデータは、次にリモート・データベースへダウンロードされます。

このオプションは、競合する更新を検出するためにも使用できます。各アップロード更新ローに対して、Mobile Link 同期サーバは統合データベースから指定されたプライマリ・キーのローをフェッチし、そのローを更新前イメージと比較して、この更新が競合する更新であるかどうかを判断します。-b を使用する場合、Mobile Link は CHAR、VARCHAR、LONG CHAR、または LONG VARCHAR 型のカラムから後続ブランクをトリムしてから、この比較を行います。

例

-b オプションを使用しない場合、Adaptive Server Anywhere または Ultra Light リモートから統合データベースの CHAR(10) カラムにアップロードされるプライマリ・キー値 'abc' は、'abc' の後に 7 個のブランク・スペースが続く値になります。同じローがダウンロードされると、リモートでは 'abc' の後に 7 個のブランク・スペースが続く値として扱われます。リモート・データベースがブランクを埋め込まない場

合、リモートは 2 つのローを持つことになります。それは、'abc' というローと 'abc' の後に 7 個のブランク・スペースが続くローです。これで、リモートで重複するローが存在することになります。

-b オプションを使用する場合、Adaptive Server Anywhere または Ultra Light リモートから統合データベースの CHAR(10) カラムにアップロードされるプライマリ・キー値 'abc' は、'abc' の後に 7 個のスペースが続く値になります。7 個のスペースが埋め込まれた値は 10 個の文字になりますが、同じローがダウンロードされるときに Mobile Link サーバは後続のスペースを削除するため、その値はリモートでは 'abc' として扱われます。このように、-b オプションは重複するローの問題を修正します。

-bc オプション

機能

BLOB のキャッシュ・サイズを設定します。

構文

dbmlsrv9 -c "*connection-string***" -bc** *size* [**k** | **m** | **g**] …

説明

BLOB のキャッシュに使用するメモリの容量です。メモリがさらに必要な場合、Mobile Link 同期サーバは代わりにディスク領域を使用します。このため、値が小さすぎるとパフォーマンスが低下します。最小推奨サイズを計算するには、任意の 1 つのローにあるすべての BLOB データの最大サイズとワーカ・スレッド数を乗算し、その結果を 4 倍にします。これにより、大きなメモリ予約量を確保できます。

size には、予約するメモリ容量をバイト単位で指定します。単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれサフィックス k、m、または g を使用してください。デフォルトは 524,288 バイトです。

-bn オプション

機能

競合検出中に比較する最大 BLOB バイト数を設定します。

構文

dbmlsrv9 -c "*connection-string***" -bn** *size*…

説明

2 つの BLOB が類似する値または同じ値で構成されている場合は、処理対象のデータの量が多くなるため、フィルタや競合検出などで両者の比較操作が高負荷になることがあります。このオプションを指定すると、Mobile Link 同期サーバは、*size* で指定された最初のバイト数だけを比較対象とします。デフォルトでは、2 つの BLOB 全体を比較します。

場合によっては、比較するデータの最大量を制限することで、同期の速度をかなり上げることができますが、エラーが発生する可能性もあります。たとえば、2 つの大きな BLOB の最後の数バイトだけが異なる場合、Mobile Link 同期サーバは、実際には異なるのに、同一であると見なす可能性があります。

-c オプション

機能

統合データベースの接続パラメータを指定します。

構文

dbmlsrv9 -c "*connection-string***"**…

説明

接続文字列には、統合データベースへの接続に十分な Mobile Link 同期サーバ情報を指定します。接続文字列は必須です。

接続文字列では、接続パラメータを *keyword=value* の形式で指定します。パラメータとパラメータの間はセミコロンで区切り、スペースは入れないでください。

接続パラメータをコマンド・ラインで指定しない場合は、ODBC データ・ソースの指定に含めてください。RDBMS と ODBC データ・ソースを調べて、必要な接続データを判断してください。

SQL Anywhere 接続パラメータの完全なリストについては、『ASA データベース管理ガイド』> 「接続パラメータ」を参照してください。

パスワードを非表示にする方法については、『ASA データベース管理ガイド』> 「ファイル非表示ユーティリティ」を参照してください。

例

```
dbmlsrv9 -c "dsn=odbcname;uid=DBA;pwd=sql"
```

-cn オプション

機能

統合データベースとの同時接続の最大数を設定します。

構文

dbmlsrv9 -c "*connection-string***" -cn** *value*…

説明

Mobile Link 同期サーバから統合データベースへの最大同時接続数を指定します。最小値 (デフォルト) は、ワーカ・スレッド数に 1 を加えた値です。指定した値が小さすぎると、警告が表示されます。

ワーカ・スレッド数よりも大きい値を指定すると、統合データベースへの接続が低速の場合や、複数のスクリプトのバージョンが使用されている場合に、特にパフォーマンスが向上します。データベース接続の最適な最大数は、スクリプトのバージョン数とワーカ・スレッド数を乗算し、それに 1 を加えた値です。この最適値を上回る数の接続を使用しても、同期速度が上がるとはかぎりません。また、Mobile Link 同期サーバと統合データベース・サーバの両方で、必要以上にリソースが消費されます。

-cr オプション

機能

データベース接続リトライの最大回数を設定します。

構文

dbmlsrv9 -c "*connection-string***" -cr** *value*…

説明

接続不良のときに、Mobile Link 同期サーバが終了前にデータベースへの接続を試行する最大回数を設定します。デフォルト値では、接続が 3 回リトライされます。

-ct オプション

機能

接続が使用されなくなってから、タイムアウトになって Mobile Link 同期サーバによって切断されるまでの時間を分単位で設定します。

構文

dbmlsrv9 -c "*connection-string***" -ct** *connection-timeout*…

説明

指定された時間にわたって使用されなかった Mobile Link データベース接続は、サーバによって解放されます。-ct オプションを使用してタイムアウトを設定できます。デフォルトのタイムアウト期間は 60 分です。

-d オプション

機能

ダウンロード・キャッシュのサイズを指定します。

構文

dbmlsrv9 -c "*connection-string***" -d** *number* [**k** | **m** | **g**]…

説明

ダウンロード確認が不要な場合、Mobile Link はダウンロード・ストリームをダウンロード・キャッシュにバッファします。ダウンロード・トランザクションをコミットするのにクライアントからの確認は必要ないので、バッファされたダウンロード・ストリームは、コミット後にクライアントに送信されます。

単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれサフィックス k、m、または g を使用してください。ダウンロード・キャッシュのデフォルト・サイズは 0.5 MB です。

-dd オプション

機能

再起動可能なダウンロードで使用します。ダウンロード・ストリームの格納場所を指定します。

構文

dbmlsrv9 -c "*connection-string***" -dd** *directory* …

説明

Mobile Link 同期サーバを停止すると、ダウンロード用のディレクトリがクリアされます。つまり、サーバを再起動すると、ダウンロードを再起動できません。

デフォルト・ディレクトリは、TEMP ディレクトリです。

参照

『Mobile Link 管理ガイド』> 「失敗したダウンロードの再開」

『Mobile Link クライアント』> 「-dc オプション」

「-ds オプション」248 ページ

-dl オプション

機能

画面にすべてのログ・メッセージを表示します。

構文

dbmlsrv9 -c "*connection-string***" -v -dl** ...

説明

Mobile Link 同期サーバのウィンドウにすべてのログ・メッセージを表示します。デフォルトでは、ログ・ファイルへの出力時には (-o を使用)、メッセージの一部だけがウィンドウに表示されます。多数のメッセージがある場合は、このオプションを指定するとパフォーマンスが低下することがあります。

-ds オプション

機能

再起動可能なダウンロードで使用します。再起動可能なダウンロード・ストリームの格納に使用するディスク領域の容量を指定します。

構文

dbmlsrv9 -c "*connection-string***" -ds** *size* [**k** | **m** | **g**]…

説明

単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれサフィックス k、m、または g を使用してください。デフォルトは 10 MB です。

参照

『Mobile Link 管理ガイド』> 「失敗したダウンロードの再開」

『Mobile Link クライアント』> 「-dc オプション」

「-dd オプション」247 ページ

-e オプション

機能

Adaptive Server Anywhere Mobile Link クライアントから送信されたエラー・ログを格納します。

構文

dbmlsrv9 -c "*connection-string***" -e** *filename*…

説明

-e オプションを指定しないと、Adaptive Server Anywhere Mobile Link クライアントからのエラー・ログは、ファイル *dbmlsrv.mle* に格納されます。-e オプションは、Mobile Link 同期サーバに対してエラー・

ログを指定のファイルに格納するように指示します。デフォルトでは、リモート・サイトでエラーが発生すると、dbmlsync は Mobile Link 同期サーバにリモート・ログ・メッセージを 32 キロバイトまで送信します。

このオプションは、同期の問題を診断するために、リモート・エラー・ログへの一元化されたアクセスを可能にします。

リモート・サイトから配信される情報量は、dbmlsync の拡張オプション ErrorLogSendLimit で制御できます。

参照

- 「-et オプション」249 ページ
- 『Mobile Link クライアント』> 「ErrorLogSendLimit (el) 拡張オプション」

-et オプション

機能

既存のファイルをトランケートした後で、Adaptive Server Anywhere Mobile Link クライアントから送信されたエラー・ログを指定のファイルに格納します。

構文

dbmlsrv9 -c "*connection-string***" -et** *filename…*

説明

-et オプションは -e オプションと同じですが、エラー・ログ・ファイルがトランケートされてから、そのファイルに新しいエラーが追加されます。

リモート・サイトから配信される情報量は、dbmlsync の拡張オプション ErrorLogSendLimit で制御できます。

参照

- 『Mobile Link クライアント』> 「ErrorLogSendLimit (el) 拡張オプション」
- 「-e オプション」248 ページ

-f オプション

機能

同期スクリプトを一度だけロードして、パフォーマンスを向上させます。

構文 **dbmlsrv9 -c "***connection-string***" -f**…

説明 -f オプションを指定しないと、Mobile Link 同期サーバは、通常の操作中に同期スクリプトが変更されたかどうかを調べます。このチェックは開発中には役立ちますが、運用環境ではパフォーマンスが低下することがあります。-f オプションを使用すると、Mobile Link 同期サーバは、Mobile Link セッションごとに 1 回だけ同期スクリプトをロードします。

-fr オプション

機能 テーブル・データ・スクリプトがない場合は、同期をアボートせず、警告の発行だけを行います。

構文 **dbmlsrv9 -c "***connection-string***" -fr**…

説明 -fr オプションを使用しない場合、同期にデータをアップロードまたはダウンロードするスクリプトが 1 つも含まれていなければ、Mobile Link 同期サーバはアボートします。このオプションを使用すると、Mobile Link はアボートする代わりに警告を発行します。

双方向同期またはアップロード専用同期の場合、-fr オプションを使用すると、データがアップロードされて、対応するアップロード・スクリプトがない場合はエラーが発生します。

- INSERT をアップロードしようとして、upload_insert、upload_new_row_insert、upload_cursor、または new_row_cursor スクリプトが定義されていない場合
- DELETE をアップロードしようとして、upload_delete、upload_old_row_insert、upload_cursor、または old_row_cursor スクリプトが定義されていない場合

  UPDATE をアップロードしようとして、upload_update、upload_new_row_insert、upload_cursor、または new_row_cursor スクリプトが定義されていない場合

ダウンロード専用同期の場合、-fr オプションを使用すると、同期内のテーブルにダウンロード・スクリプト (download_cursor または download_delete_cursor) がない場合はエラーが発生します。

-m オプション

機能　QAnywhere のメッセージ機能を有効にします。

構文　**dbmlsrv9 -c "***connection-string***" -m** [*message-properties-file*] …

説明　オプションの *message-properties-file* は、コネクタと転送ルールのプロパティ設定を指定するファイル名を含めて、QAnywhere メッセージ機能のプロパティ設定を指定します。

message-properties-file では、1 行に 1 つのプロパティを、プロパティ名、等号 (=)、プロパティ値の形式で指定します。

QAnywhere プロパティ・ファイルに設定できるプロパティは次のとおりです。

- **ianywhere.qa.server.logLevel** メッセージ機能のログ・レベル。プロパティ値は、1、2、3、4 のいずれかです。1 は、メッセージ・エラーのみを記録することを指定します。2 は、さらに警告も記録することを指定します。3 は、さらに情報メッセージも記録することを指定します。4 は、Mobile Link 同期サーバで転送される各 QAnywhere メッセージの詳細を含めて、さらに詳細な情報メッセージも記録することを指定します。デフォルトは 2 です。

  これらのログ・メッセージは、Mobile Link 同期サーバ・コンソールに出力されます。dbmlsrv9 の -o オプションまたは -ot オプショが指定されている場合、メッセージは Mobile Link 同期サーバのログ・ファイルに出力されます。

- **ianywhere.qa.compressionLevel** QAnywhere コネクタが受信する各メッセージに適用される、デフォルトの圧縮量。圧縮は 0 ～ 9 の範囲の整数で表されます。0 は圧縮がない状態で、9 は最も圧縮された状態です。デフォルトは 0 です。

  コネクタの圧縮レベルをコネクタ・プロパティ・ファイルにも設定した場合は、その設定が、該当のコネクタに対して優先されます。詳細については、『QAnywhere ユーザーズ・ガイド』> 「JMS コネクタ・プロパティ・ファイルの設定」を参照してください。

- **ianywhere.qa.server.connectorPropertiesFiles** QAnywhere コネクタの設定を、JMS などの外部のメッセージ・システムに対して指定する、1 つ以上のファイルのリスト。デフォルトは、コネクタなしです。

  詳細については、『QAnywhere ユーザーズ・ガイド』> 「JMS コネクタの使用方法」を参照してください。

- **ianywhere.qa.server.transmissionRulesFile** メッセージの転送と持続性に関するルールを指定するファイル。デフォルトでは、メッセージのフィルタはなく、メッセージの最終ステータスがメッセージの発信者に転送されると、メッセージは削除されます。

- **ianywhere.qa.server.autoRulesEvaluationPeriod** メッセージの転送ルールと持続性ルールを含む、ルールの評価時間 (ミリ秒単位)。一般に、ルールは、サーバ・ストアへのメッセージの転送中に評価されるため、ルール評価時間は、時間の影響を大きく受けるルールのみが対象です。デフォルト値は 60000 (1 分) です。

参照

- 『QAnywhere ユーザーズ・ガイド』> 「QAnywhere の概要」
- 『QAnywhere ユーザーズ・ガイド』> 「QAnywhere 転送ルール」
- 『QAnywhere ユーザーズ・ガイド』> 「JMS コネクタ・プロパティ・ファイルの設定」
- 『QAnywhere ユーザーズ・ガイド』> 「QAnywhere メッセージング用 Mobile Link 同期サーバの起動」
- 『QAnywhere ユーザーズ・ガイド』> 「JMS 統合用 Mobile Link サーバの起動」

例

たとえば、次の行を *qanymsgprop.ini* という QAnywhere メッセージ・プロパティ・ファイルに設定すると、QAnywhere に対して次の内容が指定されます。

- *connector.ini* ファイルからコネクタ・プロパティを読み込む
- *rules.ini* ファイルからサーバ転送ルールを読み込む
- ログ・レベルを 4 (詳細) に設定する

```
ianywhere.qa.server.connectorPropertiesFiles=connector.
ini
 ianywhere.qa.server.transmissionRulesFiles=rules.ini
 ianywhere.qa.server.logLevel=4
```

この設定を実行するには、-m オプションとメッセージ・プロパティ・ファイル名を指定して、Mobile Link 同期サーバを起動します。次に、コマンド・ラインの一部を示します。

```
dbmlsrv9 -m qanymsgprop.ini ...
```

-notifier オプション

機能

サーバによって開始される同期用に Notifier を起動します。

構文

dbmlsrv9 -c "*connection-string***" -notifier** [*notifier-properties-file*] …

説明

Notifier の設定ファイル名を指定した場合、またはファイル名を指定していないが、デフォルトの Notifier プロパティ・ファイル (*config.notifier*) がある場合は、そのファイルを使用して Notifier が設定されます。このファイルは、統合データベースの ml_properties テーブルに格納されている設定情報よりも優先されます。

それ以外の場合は、統合データベースの ml_properties テーブルに格納されている設定情報が使用されます。

-notifier オプションを使用する場合、有効にしたすべての Notifier が起動されます。

Notifier の有効化の詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「enable プロパティ」を参照してください。

AIX 4.3.3 では、-notifier オプションを使用できません。詳細については、http://www-106.ibm.com/developerworks/eserver/articles/JavaPart1.html を参照してください。

参照

- 『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「複数の場所でのプロパティの設定」
- 『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「Notifier のプロパティ・ファイル」

- 『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「Notifier」
- 『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「Mobile Link 通知プロパティ」

-o オプション

機能

Mobile Link 同期サーバのメッセージ・ログ・ファイルに出力メッセージのログを取り、コンソール・ウィンドウに記録されるデータを制限します。

構文

dbmlsrv9 -c "*connection-string***" -o** *logfile*…

説明

指定したファイルにすべてのログ・メッセージを書き込みます。Mobile Link 同期サーバ・ウィンドウには (表示されている場合)、通常はログが取られたすべてのメッセージのうち、一部だけが表示されることに注意してください。

同期中にエラーが発生した場合、Mobile Link 同期サーバは、この出力ファイルに完全なエラー・コンテキストを示します。エラー・コンテキストには次のような情報が含まれます。

- **ユーザ名**　同期中に Mobile Link Adaptive Server Anywhere アプリケーションによって提供される実際のユーザ名。
- **変更後のユーザ名**　modify_user スクリプトで変更されたユーザ名。
- **トランザクション**　エラーが発生したトランザクションのリスト。トランザクションには、authenticate_user、begin_synchronization、upload、prepare_for_download、download、または end_synchronization があります。
- **テーブル名**　テーブル名がある場合はテーブル名。テーブル名がない場合は NULL。
- **ローの操作**　操作には、INSERT、UPDATE、DELETE、または FETCH がある。
- **ロー・データ**　エラーが発生したローのすべてのカラム値。

- **スクリプト・バージョン**　現在同期に使用されているスクリプト・バージョン。
- **スクリプト**　エラーの原因となったスクリプト。

選択した冗長性のレベルに関係なく、ログにエラー・コンテキスト情報が表示されます。

参照

- 「-os オプション」256 ページ
- 「-dl オプション」248 ページ
- 「-ot オプション」257 ページ
- 「-on オプション」255 ページ
- 「-v オプション」265 ページ

-on オプション

機能

Mobile Link 同期サーバのメッセージ・ログ・ファイルの最大サイズを指定します。その後、そのファイルは拡張子 .old を持つ名前に変更され、新しいファイルが作成されます。

構文

dbmlsrv9 -c "*connection-string***" -on** *size* [**k** | **m**]…

説明

size には、出力ログの最大ファイル・サイズを、バイト単位で指定します。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス k、m を使用します。最小のサイズ制限は 10 KB です。

ログ・ファイルが指定されたサイズに達すると、Mobile Link 同期サーバは出力ファイルを拡張子 .old を持つ名前に変更し、元の名前で新しいファイルを開始します。

注意
.old ファイルがすでに存在する場合は、そのファイルが上書きされます。古いログ・ファイルを削除しないようにするには、代わりに -os オプションを使用します。

このオプションは、-os オプションと同時に使用できません。

参照

- 「-o オプション」254 ページ
- 「-ot オプション」257 ページ

- 「-on オプション」255 ページ
- 「-os オプション」256 ページ
- 「-v オプション」265 ページ

-oq オプション

機能

Windows で、起動エラーの発生時にエラー・ダイアログが表示されないようにします。

構文

dbmlsrv9 -c "*connection-string***" -oq** …

説明

デフォルトでは、起動エラーが発生すると、Mobile Link 同期サーバにメッセージ・ボックスが表示されます。-oq オプションを指定すると、このダイアログは表示されません。

-os オプション

機能

Mobile Link 同期サーバのメッセージ・ログ・ファイルの最大サイズを設定し、その後新しい名前を持つ新しいログ・ファイルを作成し使用します。

構文

dbmlsrv9 -c "*connection-string***" -os** *size* [**k** | **m**]…

説明

size には、出力メッセージのログを取るファイルの最大サイズを指定します。デフォルトの単位はバイトです。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス k、m を使用します。最小のサイズ制限は 10 KB です。

Mobile Link 同期サーバは現在のファイル・サイズを確認してから、ファイルに出力メッセージのログを取ります。ログ・メッセージによって、ファイル・サイズが指定したサイズより大きくなると、Mobile Link 同期サーバはメッセージ・ログ・ファイルの名前を *yymmddxx.mls* に変更します。*xx* は 00 ～ 99 の数字で、*yymmdd* は現在の年月日を表します。

このオプションを使用すると、古いメッセージ・ログ・ファイルを削除して、ディスク領域を解放できます。常に最新の出力が -o または -ot で指定したファイルに追加されます。

このオプションは、-on オプションと一緒に使用することはできません。

注意
このオプションを使用すると、ログ・ファイルが無制限に作成されます。この状況を回避するには、-o または -on を使用します。

参照

- 「-o オプション」254 ページ
- 「-on オプション」255 ページ
- 「-ot オプション」257 ページ
- 「-v オプション」265 ページ

-ot オプション

機能

最初に Mobile Link 同期サーバのメッセージ・ログ・ファイルをトランケートしてから、そのファイルに出力メッセージのログを取ります。

構文

dbmlsrv9 -c "*connection-string***" -ot** *logfilename* …

説明

メッセージ・ログ・ファイルをトランケートし、そのファイルに出力メッセージを追加します。デフォルトでは、画面に出力を送信します。

参照

- 「-on オプション」255 ページ
- 「-os オプション」256 ページ
- 「-v オプション」265 ページ
- 「-o オプション」254 ページ

-ps オプション

機能

接続ごとに、キャッシュする準備文の最大数を設定します。

構文

dbmlsrv9 -c "*connection-string***" -ps** *num* …

説明

準備文キャッシュに保持される ODBC 準備文の最大数を制御します。

キャッシュ準備文はパフォーマンスを向上させますが、リソースを消費します。統合データベースのタイプによっては準備文の数を制限するものもあるので、このオプションはその制限に従って設定してください。

-q オプション

機能

Mobile Link を起動時に最小化ウィンドウで実行するように指示します。

構文

dbmlsrv9 -c "*connection-string***" -q** …

説明

Mobile Link 同期サーバのウィンドウを最小化します。

-r オプション

機能

デッドロックの最大リトライ回数を設定します。

構文

dbmlsrv9 -c "*connection-string***" -r** *retries* …

説明

デフォルトでは、Mobile Link 同期サーバはデッドロックされたアップロードを最大 10 回リトライします。デッドロックは解消できる保証がないため、デッドロックが解除されない場合は同期が失敗します。このオプションで、リトライの制限回数を任意に設定できます。サーバによるデッドロック・トランザクションのリトライを停止するには、**−r 0** を指定します。この設定の上限は、2 の 32 乗から 1 を引いた値です。

-rd オプション

機能

デッドロック・リトライ間の最大遅延時間を設定します。

構文

dbmlsrv9 -c "*connection-string***" -rd** *delay* …

説明

アップロード・トランザクションがデッドロックされると、Mobile Link 同期サーバは不特定の時間待機してから、そのトランザクションをリトライします。遅延時間がランダムなため、その後の試行が成功する可能性が高くなります。このオプションを使用すると、最大遅延

時間を秒単位で指定できます。値を 0 にするとリトライが即座に行われますが、より大きな値を指定したほうがリトライの成功率が高くなります。デフォルトの最大遅延値は 30 です。

-s オプション

機能

一度にフェッチ、挿入、更新、または削除されるローの最大数を設定します。

構文

dbmlsrv9 -c "*connection-string***" -s** *count* …

説明

count に Mobile Link 同期サーバと統合データベースの間で転送されるローの最大数を設定します。

このオプションを設定する場合は、ODBC プリフェッチ・オプションで指定したロー数以上の値に設定します。デフォルト値は 10 です。

一度にフェッチされるローの数は、ログ・ファイルで **rowset size** として参照できます。

*注意：*転送されるローの実際の数は、ODBC データ・ソースかデータベース・クライアント・ソフトウェアまたはその両方に影響されます。

-sl dnet オプション

機能

.NET Common Language Runtime (CLR) オプションを設定し、起動時に CLR を強制的にロードします。

構文

dbmlsrv9 -c "*connection-string***" -sl dnet** *options* …

説明

.NET CLR に直接渡すオプションを設定します。オプションは次のとおりです。

オプション	説明
-D*name*=*value*	環境変数を設定する。次に例を示します。 `-Dsynchtype=far -Dextra_rows=yes` 詳細については、.NET フレームワークのクラス System.Environment を参照してください。
-MLAutoLoadPath=*path*	基本アセンブリのロケーションを設定する。プライベート・アセンブリでのみ有効です。アセンブリのロケーションを Mobile Link に指示するには、このオプションまたは -MLDomConfigFile を使用します。両方同時には使用できません。-MLAutoLoadPath を使用する場合、イベント・スクリプトでドメインを指定できません。デフォルトは現在のディレクトリです。
-MLDomConfigFile=*file*	基本アセンブリのロケーションを設定する。共有アセンブリがある場合、すべてのアセンブリをディレクトリにロードしない場合、またはその他の理由で MLAutoLoadPath を使用できない場合に使用します。アセンブリのロケーションを Mobile Link に指示するには、-MLDomConfigFile または -MLAutoLoadPath を使用します。両方同時には使用できません。
-MLStartClasses= *classnames*	サーバの起動時に、ユーザ定義の起動クラスをリスト内の順序でロードしインスタンス化する。
-clrConGC	CLR での同時ガーベジ・コレクションを有効にする。
-**clrFlavor=**(**wks** | **svr**)	ロードする .NET CLR の種類。サーバは **svr** で、ワークステーションは **wks** です。デフォルトでは、**wks** がロードされます。

オプション	説明
-clrVersion=*version*	ロードする .NET CLR のバージョン。プレフィクス **v** が必要です。たとえば、**v1.0.3705** と指定すると、ディレクトリ *¥WINNT¥Microsoft.NET¥Framework¥v1.0.3705* がロードされます。

このオプション・リストを表示するには、次のコマンドを使用します。

```
dbmlsrv9 -sl dnet (?)
```

参照

- 『Mobile Link 管理ガイド』> 「.NET での同期スクリプトの作成」

-sl java オプション

機能

Java 仮想マシンのオプションを設定し、起動時に仮想マシンを強制的にロードします。

構文

dbmlsrv9 -c "*connection-string***" -sl java (** *options* **)** …

説明

Java 仮想マシンに直接渡す -jrepath とその他のオプションを設定します。オプションは次のとおりです。

オプション	説明
{ **-hotspot** | **-server** | **-classic** }	使用する Java VM のデフォルトのオプションを上書きする。
{ **-cp** | **-classpath**} *location***;**…	クラスの検索先となる一連のディレクトリまたは jar ファイルを指定する。-cp または -classpath は、中カッコまたは丸カッコで囲んでください。
-D*name*=*value*	システム・プロパティを設定する。次に例を示します。 `-Dsynchtype=far -Dextra_rows=yes`

オプション	説明
-DMLStartClasses=*class*, …	サーバの起動時に、ユーザ定義の起動クラスをリスト内の順序でロードしインスタンス化する。
-jrepath *path*	デフォルトの JRE パスを上書きする。これは、*Sybase¥shared* ディレクトリ内の *sun¥jre142* ディレクトリにあります。
-verbose [**:class** |**:gc** | **:jni**]	冗長出力を有効にする。
-X *vm-option*	ファイル *Xusage.txt* の説明に従って VM 固有のオプションを設定する。このファイルは、デフォルトで *Sybase¥Shared¥Sun¥jre142¥bin¥hotspot* にインストールされます。

オプションはカッコで囲んでください。これは、丸カッコでも、上記の構文に示すように中カッコ { } でもかまいません。

このオプション・リストを表示するには、次のコマンドを使用します。

```
dbmlsrv9 -sl java (?)
```

使用できる Java オプションのリストを表示するには、次のように入力します。

```
java
```

説明

-jrepath オプションは、Windows でのみ使用できます。UNIX で特定の JRE をロードする場合は、LD_LIBRARY_PATH (AIX の場合は LIBPATH、HP-UX の場合は SHLIB_PATH) を設定して、JRE を含むディレクトリを指定します。ディレクトリは、すべての Adaptive Server Anywhere インストール・ディレクトリの前に指定します。

参照

- 『Mobile Link 管理ガイド』> 「Java による同期スクリプトの作成」

例

たとえば、次の dbmlsrv9 コマンド・ラインの一部では、システム・アサートを有効にする Java 仮想マシン・オプションを設定します。

```
dbmlsrv9 -sl java {-cp ;¥myclasses; -esa} ...
```

次の dbmlsrv9 コマンド・ラインの一部は、LDAP_SERVER システム・プロパティを定義します。

```
dbmlsrv9 -sl java { -cp ;¥myclasses; -
DLDAP_SERVER=huron-ldap } ...
```

-t オプション

機能

Mobile Link が発行するすべての ODBC 呼び出しを含むファイルを作成します。

構文

dbmlsrv9 -c "*connection-string***" -t** *ODBC-output-file …*

説明

このオプションを使用すると、Mobile Link が発行するすべての ODBC 呼び出しを含むファイルを作成できます。UNIX では、Adaptive Server Anywhere ドライバをドライバ・マネージャとして使用している場合、この機能は無視されます。この機能は、何が呼び出され、渡され、取り出されたかをトレースするのに有用です。このオプションはパフォーマンスに重大な影響を与えるので、運用環境では使用しないでください。

ファイルが大きくならないようにするには、「-tt オプション」263 ページを使用します。

参照

- 「-tt オプション」263 ページ

-tt オプション

機能

Mobile Link が発行する ODBC 呼び出しのログをファイルに取ります。ファイルがすでに存在する場合は、最初にファイルを削除します。

構文

dbmlsrv9 -c "*connection-string***" -tt** *ODBC-output-file …*

説明

このオプションは、Mobile Link が発行するすべての ODBC 呼び出しを含むファイルを作成するために使用します。UNIX では、Adaptive Server Anywhere ドライバをドライバ・マネージャとして使用している場合、この機能は無視されます。この機能は、何が呼び出され、渡

され、取り出されたかをトレースするのに有用です。このオプションはパフォーマンスに重大な影響を与えるので、運用環境では使用しないでください。

参照

- 「-t オプション」263 ページ

-u オプション

機能

アップロード・キャッシュ・サイズを設定します。

構文

dbmlsrv9 -c "*connection-string***" -u** *size*[**k** | **m** | **g**] …

説明

処理中のアップロード・ストリームのキャッシュ用に予約するメモリ容量を、バイト単位で指定します。単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれサフィックス k、m、または g を使用してください。クライアントが大きなストリームをアップロードする場合や、多くのクライアントが一度に同期する場合、またはその両方の場合は、この値を増やすことを検討してください。推奨サイズは、アップロード・ストリームの予測される最大サイズと、ワーカ・スレッド数を乗算したものです。デフォルト値は 500 KB です。

参照

- 「-bc オプション」244 ページ

-ud オプション

機能

Mobile Link をデーモンとして実行するように指示します。

構文

dbmlsrv9 -c "*connection-string***" -ud** …

説明

UNIX プラットフォーム専用です。

参照

- 『Mobile Link 管理ガイド』> 「現在のセッション以外での Mobile Link の実行」

-us オプション

機能　アップロードするローがテーブルにないときに、Mobile Link 同期サーバがアップロード・スクリプトを呼び出さないようにします。

構文　**dbmlsrv9 -c "***connection-string***" -us** …

説明　このオプションは、dbmlsync のトランザクション・レベルのアップロードを使用している場合に特に便利です。

参照　『Mobile Link クライアント』> 「-tu オプション」

-v オプション

機能　メッセージ・ログ・ファイルにログを取り、同期ウィンドウに表示する情報を指定できます。

構文　**dbmlsrv9 -c "***connection-string***" -v**[*levels*] …

説明　このオプションは、dbmlsync のトランザクション・レベルのアップロードを使用している場合に特に便利です。

このオプションは、メッセージ・ログ・ファイルに書き込まれるメッセージのタイプを制御します。

−v を単独で指定すると、Mobile Link 同期サーバは、各同期について最小量の情報を書き込みます。

levels の値は次のとおりです。たとえば -vnrsu など、一度に 1 つまたは複数のオプションを使用できます。

- **+** 冗長性を高めるすべてのログ・オプションを on にする。
- **c** 呼び出された各同期スクリプトの内容を表示する。このレベルには、s の機能が含まれます。
- **f** 最初の読み込みエラーを表示する。このオプションは、負荷分散デバイスがサーバの活性を確認するとき、データを送信しない接続を確立し同期が失敗したために発生したエラーのログを取ります。

TCP/IP 接続の場合は、TCP/IP の **ignore** オプションを使用するほうがよいでしょう。詳細については、「-x オプション」269 ページを参照してください。

- **h** 同期中にアップロードされたリモート・スキーマを表示する。
- **n** ロー・カウントの要約を表示する。
- **p** 進行オフセットを表示する。
- **r** アップロードまたはダウンロードされた各ローのカラム値を表示する。
- **s** 呼び出された各同期スクリプトの名前を表示する。
- **t** ODBC 標準フォーマットのスクリプトから生成された、変換された SQL を表示する。このレベルには、c の機能が含まれます。次の例は、Adaptive Server Anywhere での文の自動変換を示します。

```
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' )
```

次の例は、同じ文の Microsoft SQLServer での変換を示します。

```
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** 未定義のテーブル・スクリプトを表示する。これは、経験の浅いユーザが同期処理を理解する上で役立ちます。

-w オプション

機能 ワーカ・スレッド数を設定します。

構文 **dbmlsrv9 -c "***connection-string***" -w** *count* …

説明

各ワーカ・スレッドでは、同期要求を一度に 1 つ受け入れます。各ワーカ・スレッドはネットワーク・プロトコルと関連付けられています。複数のプロトコルが定義されている場合、ワーカ・スレッドは各プロトコルに均等に分割されます。

ワーカ・スレッドはそれぞれ、統合データベースへの接続を 1 つ使用します。Mobile Link 同期サーバは、接続をさらに 1 つ管理用に開きます。したがって、Mobile Link 同期サーバから統合データベースへの接続の最小数は、*count* + 1 となります。

ワーカ・スレッド数は Mobile Link 同期スループットに大きく影響するので、特定の同期設定に最適なワーカ・スレッド数を判別するためのテストを実行する必要があります。ワーカ・スレッド数によって同時にアクティブにできる同期の数が決まります。残りの同期はキューイングされてワーカ・スレッドが使用可能になるのを待機します。したがって、ワーカ・スレッドを追加するとスループットが向上しますが、アクティブな同期の間で競合が発生する確率も高くなります。ワーカ・スレッドを追加していくと、ある時点で、複数の同期を同時に行うメリットよりも競合が発生する確率の方が大きくなり、スループットが低下します。

詳細については、Mobile Link のパフォーマンスに関するホワイトペーパ (英文) http://my.sybase.com/detail?id=1009664 と 『Mobile Link 管理ガイド』> 「Mobile Link のパフォーマンス」を参照してください。

このオプションの設定値は -wu オプションのデフォルト設定にもなります。-wu オプションは、同時にアップロードできるスレッド数を制限するためのオプションです。特に、処理速度が遅いコンピュータ上のリモート・データベース、または Mobile Link サーバとの通信速度が遅い接続など、ダウンロードを行うワーカ・スレッドの最適数が、アップロードを行うワーカ・スレッドの最適数より多い場合は、このオプションが役立ちます。処理速度が遅い同期クライアント (Palm デバイスまたはダイヤルアップ接続のコンピュータなど) に対するテストでは、ワーカ・スレッド数を大きく設定し (-w を使用)、アップロードを同時に適用できるワーカ・スレッド数を小さく設定する (-wu を使用) ことで、最高のスループットを達成できました。通常、-wu の最適数は統合データベースによって異なり、リモート・データベースの処理速度またはネットワーク速度とはあまり関係ありません。し

たがって、-w を使用してスレッド数を増やす場合、-wu を使用して同時にアップロードできるスレッド数を制限できます。詳細については、「-wu オプション」268 ページを参照してください。

ワーカ・スレッド数のデフォルトは 5 です。

-wu オプション

機能

アップロードを同時に適用できるワーカ・スレッドの最大数を設定します。

構文

dbmlsrv9 -c "*connection-string***" -wu** *count* …

説明

-wu オプションを使用して、アップロードを同時に適用できるワーカ・スレッド数を制限します。制限値に達すると、アップロードの適用の準備が完了しているワーカ・スレッドは、別のワーカ・スレッドがアップロードを終了するまで待機します。ワーカ・スレッドが余っていれば、アップロードを受け取るか、ダウンロードすることができます。

統合データベースで発生する競合の最も一般的な原因は、アップロードを同時に適用するワーカ・スレッドが多すぎることです。ネットワーク接続の速度が遅い場合、またはクライアント・デバイスの処理速度が遅い場合に、この問題が発生します。たとえば、広域無線ネットワーク上で作業するときや Palm デバイスを使用する場合は、スレッド数を増やす (-w を使用) 一方で、同時にアップロードを適用できるスレッド数を制限できます。

次の例を考えます。LAN と PC 上のリモート・データベースを使用するパイロット設定では、アップロード専用同期とダウンロード専用同期の両方に対して、ワーカ・スレッドの最適数は約 10 であり、それが統合データベースの CPU 使用率 100% に相当します。ワーカ・スレッドの数が少ないと、スループットが低下し、統合データベースの CPU 使用率が低下します。ワーカ・スレッドの数が多くても、統合データベースの処理速度はワーカ数が 10 の場合と同じなので、スループットは向上しません。Mobile Link ワーカ・スレッド数を 10 に指定してダイヤルアップ・ネットワークに切り替えると、スループットが低下し、統合データベースの CPU 使用率が低下します。ワーカ・スレッド数を増やす (-w を使用) 一方、アップロードを同時に適用で

きる数を 10 に制限 (-wu を使用) することによって、LAN で達成できる値に近いスループット (および統合データベースの CPU 使用率) を達成できます。

デフォルトでは、すべてのワーカ・スレッドで同時にアップロードを適用できます。使用されるワーカ・スレッドの数は -w オプションで設定します。デフォルトは 5 です。

-x オプション

機能

Mobile Link クライアントのネットワーク・プロトコルとプロトコル・オプションを設定します。Mobile Link 同期サーバは、これらのプロトコルとパラメータを使用して同期要求を受信します。

構文

dbmlsrv9 -c "*connection-string***"**
　　-x *protocol*[*protocol-options*]…

protocol-options : **(** *keyword*=*value*;... **)**

説明

クライアント・アプリケーションとの通信に使用する通信プロトコルを指定します。デフォルトは TCPIP でポート 2439 を使用します。

Ultra Light ユーザに対する注意
Ultra Light Java アプリケーションを使用し、TLS セキュリティを使用している場合は、-x の構文が多少異なります。詳細については、『Ultra Light 静的型 Java ユーザーズ・ガイド』> 「トランスポート・レイヤ・セキュリティ」を参照してください。

許可されている *protocol* の値は次のとおりです。

- **tcpip**　TCP/IP を介したアプリケーションからの接続を受け入れます。
- **http**　標準の Web プロトコルを介した接続を受け入れます。クライアント・アプリケーションは HTTP のバージョンを選択でき、Mobile Link 同期サーバは接続ごとにバージョンを調整します。

- **https**　安全なトランザクションを処理する HTTP の変形プロトコルを介した接続を受け入れます。HTTPS ストリームは、RSA 暗号化を使用して HTTP over SSL/TLS を実装します。また、他の HTTPS サーバと互換性があります。

- **https_fips**　HTTPS プロトコルと、FIPS が承認している暗号化アルゴリズムを使用して、接続を受け入れます。HTTPS_FIPS では、別の FIPS 140-2 承認ソフトウェアを使用します。rsa_tls を使用しているサーバは、rsa_tls_fips を使用しているクライアントと互換性があり、rsa_tls_fips を使用しているサーバは、rsa_tls を使用しているクライアントと互換性があります。rsa_tls_fips は、Windows 上の Adaptive Server Anywhere データベースでのみ使用できます。

オプションで、*option=value* の形式でネットワーク・プロトコル・オプションを指定することもできます。個々の複数のオプションは、セミコロンで区切ってください。指定するオプションは、選択するプロトコルによって異なります。

- **TCP/IP オプション**　**tcpip** プロトコルを指定する場合は、オプションで次のプロトコル・オプションを指定できます。

  - **backlog=number-of-connections**　リモート接続の最大数。この値を超えて新しい同期要求が行われると Mobile Link によって拒否されるため、クライアント側で同期が失敗します。バックログのサイズを指定することで、サーバがビジーなときにクライアントが同期を待ち続けることを回避できます。バックログのサイズが指定されていない場合、クライアントは、バックログのサイズにかかわらず同期を試行します。

  - **host=hostname**　Mobile Link 同期サーバが受信に使用するホスト名または IP 番号。デフォルト値は **localhost** です。

  - **ignore=hostname**　接続を確立する場合に、Mobile Link 同期サーバが無視するホスト名または IP 番号。このオプションを使用すると、最も可能性が低いレベルにある負荷分散装置からの要求を無視することができ、Mobile Link サーバ・ログおよび Mobile Link モニタの出力ファイルへの過剰な出力を回避できます。無視するホストは複数指定できま

す。たとえば、`-x tcpip(ignore=lb1;ignore=123.45.67.89)` の形式で指定します。

- **liveness_timeout=n**　クライアントとの最後の通信から、Mobile Link が同期をアボートするまでの時間 (秒単位)。値 0 はタイムアウトがないことを意味します。クライアントのダウンロード確認が off (デフォルト) に設定されている場合だけ、このオプションは有効です。デフォルトは 120 秒。

- **port=portnumber**　Mobile Link 同期サーバが受信に使用するソケット・ポート番号。デフォルトのポートは 2439 です。これは、Mobile Link 同期サーバの IANA 登録ポート番号です。

  security=cipher(keyword=value;…)　この接続を介して行われるすべての通信は、トランスポート・レイヤ・セキュリティを使用して暗号化および認証されます。暗号化は **ecc_tls**、**rsa_tls**、**rsa_tls_fips** のいずれかで、これらは楕円曲線暗号化、RSA 暗号化、FIPS 承認の RSA 暗号化にそれぞれ対応します。rsa_tls_fips は、Certicom の FIPS 140-2 承認ソフトウェアという別のソフトウェアを使用しますが、https (バージョン 9.0.2 以降) を使用するクライアントと互換性があります。rsa_tls_fips は、Windows 上の Adaptive Server Anywhere クライアントでのみ使用できます。

  下位互換性を保つために、**ecc_tls** を **certicom_tls** と指定することもできます。

  セキュリティ・パラメータは、**certificate** (サーバ認証で使用される証明書のパスとファイル名) および **certificate_password** です。選択した暗号パッケージ・プログラムと一致する証明書を使用してください。

  詳細については、「トランスポート・レイヤ・セキュリティを使用する Mobile Link 同期サーバの起動」219 ページを参照してください。

別途ライセンスを取得できるオプションが必要

トランスポート・レイヤ・セキュリティを使用するには、別途ライセンスを取得できる SQL Anywhere Studio セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。

このコンポーネントの注文方法については、『SQL Anywhere Studio の紹介』> 「別途ライセンスが入手可能なコンポーネント」を参照してください。

- **HTTP オプション** **http** プロトコルを指定する場合は、オプションで次のプロトコル・オプションを指定できます。
  - **backlog=number-of-connections** リモート接続の最大数。この値を超えて新しい同期要求が行われると Mobile Link によって拒否されるため、クライアント側で同期が失敗します。バックログのサイズを指定することで、サーバがビジーなときにクライアントが同期を待ち続けることを回避できます。バックログのサイズが指定されていない場合、クライアントは、バックログのサイズにかかわらず同期を試行します。
  - **buffer_size=number** Mobile Link サーバから送信される HTTP メッセージの本文の最大サイズ (バイト単位)。このオプションを変更すると、HTTP メッセージの送信に割り当てられるメモリ量が減少または増加します。デフォルトは 65,535 バイトです。
  - **contd_timeout=seconds** 同期を中止する前に、部分的に完了した同期の次の部分を受信するまで Mobile Link 同期サーバが待機する秒数。クライアントが接続を続行しないことが待機時間で示されている場合は Mobile Link のワーカ・スレッドを解放するように、このオプションを調整できます。デフォルト値は 30 秒です。
  - **host=hostname** Mobile Link 同期サーバが受信に使用するホスト名または IP 番号。デフォルト値は **localhost** です。

- **port=portnumber**　Mobile Link 同期サーバが受信に使用するソケット・ポート番号。ポート番号は 10 進数で、Mobile Link 同期サーバがモニタするように設定されているポートと一致させます。デフォルトのポートは 80 です。

- **unknown_timeout=seconds**　同期を中止する前に、新しい接続の HTTP ヘッダを受信するまで Mobile Link 同期サーバが待機する秒数。ネットワーク障害が発生したことが待機時間で示されている場合は Mobile Link のワーカ・スレッドを解放するように、このオプションを調整できます。デフォルト値は 30 秒です。

- **url_suffix=suffix**　各 HTTP 要求の 1 行目の URL に追加するサフィックス。このパラメータを使用して、特定のクライアントが確実に目的のサーバへ接続するようにできます。値が一致しないと、同期は失敗します。

- **version=http-version**　Mobile Link 同期サーバは、クライアントが使用している HTTP のバージョンを自動的に検出します。このパラメータは、クライアントが使用しているバージョンをサーバが検出できない場合に使用される、HTTP のデフォルト・バージョンを指定する文字列です。**1.0** または **1.1** を選択できます。デフォルト値は **1.1** です。

- **HTTPS または HTTPS_FIPS オプション**　https 通信ストリームでは、トランスポート・レイヤ・セキュリティで RSA デジタル証明書を使用します。https_fips ストリームでは、別の FIPS 140-2 承認ソフトウェアを使用しますが、https と互換性があります。

  詳細については、「トランスポート・レイヤ・セキュリティを使用する Mobile Link 同期サーバの起動」219 ページを参照してください。

別途ライセンスを取得できるオプションが必要

トランスポート・レイヤ・セキュリティを使用するには、別途ライセンスを取得できる SQL Anywhere Studio セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。

このコンポーネントの注文方法については、『SQL Anywhere Studio の紹介』> 「別途ライセンスが入手可能なコンポーネント」を参照してください。

https プロトコルを指定する場合は、オプションで次のプロトコル・オプションを指定できます。

- **backlog=number-of-connections**　リモート接続の最大数。この値を超えて新しい同期要求が行われると Mobile Link によって拒否されるため、クライアント側で同期が失敗します。バックログのサイズを指定することで、サーバがビジーなときにクライアントが同期を待ち続けることを回避できます。バックログのサイズが指定されていない場合、クライアントは、バックログのサイズにかかわらず同期を試行します。

  buffer_size=number　Mobile Link サーバから送信される HTTPS メッセージの本文の最大サイズ (バイト単位)。このオプションを変更すると、HTTPS メッセージの送信に割り当てられるメモリ量が減少または増加します。デフォルトは 65,535 バイトです。

- **contd_timeout=seconds**　同期を中止する前に、部分的に完了した同期の次の部分を受信するまで Mobile Link 同期サーバが待機する秒数。クライアントが接続を続行しないことが待機時間で示されている場合は Mobile Link のワーカ・スレッドを解放するように、このオプションを調整できます。デフォルト値は 30 秒です。

- **host=hostname**　Mobile Link 同期サーバが受信に使用するホスト名または IP 番号。デフォルト値は **localhost** です。

- **port=portnumber**　Mobile Link 同期サーバが受信に使用するソケット・ポート番号。ポート番号は 10 進数で、Mobile Link 同期サーバがモニタするように設定されているポートと一致させます。デフォルトのポートは 443 です。

- **certificate**　サーバ認証で使用される証明書のパスとファイル名。これは RSA 証明書にしてください。

- **certificate_password**　証明書のパスワードを指定するオプションのパラメータ。

  セキュリティの詳細については、『Mobile Link 管理ガイド』> 「Mobile Link トランスポート・レイヤ・セキュリティ」を参照してください。

- **unknown_timeout=seconds**　同期を中止する前に、新しい接続の HTTP ヘッダを受信するまで Mobile Link 同期サーバが待機する秒数。ネットワーク障害が発生したことが待機時間で示されている場合は Mobile Link のワーカ・スレッドを解放するように、このオプションを調整できます。デフォルト値は 30 秒です。

- **url_suffix=suffix**　各 HTTP 要求の 1 行目の URL に追加するサフィックス。このパラメータを使用して、特定のクライアントが確実に目的のサーバへ接続するようにできます。値が一致しないと、同期は失敗します。

- **version=http-version**　Mobile Link 同期サーバは、クライアントが使用している HTTP のバージョンを自動的に検出します。このパラメータは、クライアントが使用しているバージョンをサーバが検出できない場合に使用される、HTTP のデフォルト・バージョンを指定する文字列です。**1.0** または **1.1** を選択できます。デフォルト値は **1.1** です。

例

次のコマンド・ラインは、バックログのサイズを 10 接続に設定します。

```
dbmlsrv9 -c dsn=asa90sample;uid=DBA;pwd=SQL -x
http(backlog=10)
```

-za オプション

機能

簡単なスナップショットを使った同期を実行する文ベースのスクリプトを生成します。

構文

dbmlsrv9 -c "*connection-string***" -za**

説明

次のスクリプトを生成します。

- upload_insert

- upload_update
- upload_delete
- download_cursor

このオプションは、アクティブなスクリプト、つまり現在の同期に対して使用されるスクリプトを生成します。生成されたスクリプトは、保存され、同じスクリプト・バージョンを使用する後続の同期に対して使用されます。-za オプションは通常、短時間のデモに使用されます。独自のスクリプトを作成する手始めとしてスクリプトを生成するには、dbmlsrv9 -ze オプションの方がより便利です。

スクリプトを生成するには、クライアントがカラム名を送信することも指定してください。同期を開始するときに、この指定を行います。Adaptive Server Anywhere リモートについては、『Mobile Link クライアント』> 「SendColumnNames (scn) 拡張オプション」を参照してください。Ultra Light リモートについては、『Mobile Link クライアント』> 「Send Column Names 同期パラメータ」を参照してください。

生成されたスクリプトは、クライアントから送信されたテーブル名とカラム名を使用して、1 対 1 のスナップショット同期を実行します。統合データベースにリモートと異なるテーブル名またはカラム名がある場合は、これらのスクリプトをアクティブにすると、同期中にエラーが発生します。

注意：
存在しないスクリプト・バージョンで初めてリモートが同期するときに、スクリプトが生成されます。指定したスクリプト・バージョンが存在する場合は、-za は機能しません。つまり、-za を使用して、同じスクリプト・バージョンの複数のスクリプトを、一度に 1 つのテーブルに対して 1 つずつ生成することはできません。-za を使用する場合は、すべてのテーブルとパブリケーションに対するスクリプトを一度に生成してください。

参照

- 「-ze オプション」277 ページ

例

次の dbmlsrv9 コマンドは、スクリプトの自動生成を有効にします。dbmlsync コマンドは、必要な SendColumnNames オプションを設定します。

```
dbmlsrv9 -c "dsn=YourDBDSN" -za
 dbmlsync -c dsn=dsn_remote -e "SendColumnNames=ON"
```

-ze オプション

機能

簡単なスナップショットを使った同期を実行するサンプル・スクリプトを生成します (スクリプトがアクティブな場合)。

構文

dbmlsrv9 -c "*connection-string***" -ze**

説明

次のスクリプトを生成します。

- example_upload_insert
- example_upload_update
- example_upload_delete
- example_download_cursor

これらのスクリプトをもとに独自のスクリプトを作成できます。

スクリプトを生成するには、クライアントがカラム名を送信することも指定してください。同期を開始するときに、この指定を行います。Adaptive Server Anywhere リモートについては、『Mobile Link クライアント』> 「SendColumnNames (scn) 拡張オプション」を参照してください。Ultra Light リモートについては、『Mobile Link クライアント』> 「Send Column Names 同期パラメータ」を参照してください。

生成されたスクリプトは、クライアントから送信されたテーブル名とカラム名を使用して、1 対 1 のスナップショット同期を実行します。統合データベースにリモートと異なるテーブル名またはカラム名がある場合は、これらのスクリプトをアクティブにすると、同期中にエラーが発生します。

注意：
スクリプトが生成されるのは、リモートが最初に同期するときと、指定のスクリプト・バージョンが存在しないときだけです。それ以外の場合、-ze は機能しません。

参照

- 「-za オプション」275 ページ

-zp オプション

機能

競合検出にどのタイムスタンプ値を使用するかを調整します。

構文

dbmlsrv9 -c "*connection-string***" -zp**

説明

このオプションを使用すると、統合データベースとリモート・データベース間にタイムスタンプの競合がある場合に、最小精度よりも高い精度を持つタイムスタンプ値を競合検出に使用できます。リモート・データベース上の更新されたタイムスタンプによって、次の同期で競合が発生する場合があるため、このオプションは、統合データベースのタイムスタンプがリモート・データベースのタイムスタンプよりも精度が高い場合に有用です。このオプションを指定すると、Mobile Link はそのような競合を無視します。精度が不一致で -zp が使用されていない場合は、同期ごとにスキーマが異なるテーブル別の警告がログに書き込まれるので、-zp オプションの使用を推奨します。可能であればリモート・データベースのタイムスタンプの精度を調整するようユーザに通知する、同期ごとの警告もさらに追加されます。

-zs オプション

機能

dbmlstop 用の Mobile Link サーバの名前を指定します。

構文

dbmlsrv9 -c "*connection-string***" -zs** *name*

説明

指定する名前には、英数字を使用できますが、その他の文字は使用できません。

-zs オプションを使用して起動した Mobile Link 同期サーバの停止に dbmlstop を使用するときは、サーバ名を dbmlstop のコマンド・ラインで指定する必要があります。たとえば、`dbmlstop myMLserver` と指定します。Mobile Link 同期サーバがインストールされているコンピュータからしか、シャットダウンは開始できません。

参照

- 「Mobile Link 停止ユーティリティ」606 ページ

-zt オプション

機能

Mobile Link 同期サーバを実行するのに使用されるプロセッサの最大数を指定します。

構文

dbmlsrv9 -c "*connection-string***" -zt** *number*

説明

一部の ODBC ドライバでは、このオプションが必須です。また、プロセッサ・リソースを厳密に制御できます。

このオプションは、Windows オペレーティング・システムでしか使用できません。デフォルトは、コンピュータに搭載されているプロセッサの数です。

-zu オプション

機能

authenticate_user スクリプトが未定義の場合に、ユーザの自動的な追加を制御します。

構文

dbmlsrv9 -c "*connection-string***" -zu**{ + | - } …

説明

このオプションを -zu+ として指定すると、認識されなかった Mobile Link ユーザ名が最初の同期時に ml_user テーブルに追加されます。-zu- を引数に指定するか、まったく指定しない場合は、認識されないユーザ名を同期できません。

参照

- 『Mobile Link クライアント』> 「Mobile Link ユーザの認証」
- 「Mobile Link ユーザ認証ユーティリティ」608 ページ
- 「authenticate_user 接続イベント」420 ページ

-zw オプション

機能

表示する警告メッセージのレベルを制御します。

構文

dbmlsrv9 -c "*connection-string***" -zw** *levels*

説明

Mobile Link には、5 つのレベルの警告メッセージがあります。

レベル	説明
0	すべての警告メッセージを表示しない
1	サーバ・レベルと高い ODBC レベル：Mobile Link 同期サーバが起動するときに警告メッセージを表示
2	同期レベルとユーザ・レベル：同期が開始するときに警告メッセージを表示
3	スキーマ・レベル：Mobile Link 同期サーバがクライアント・スキーマを処理するときに警告メッセージを表示
4	スクリプト・レベルと低い ODBC レベル：Mobile Link 同期サーバがスクリプトをフェッチ、準備、または実行するときに警告メッセージを表示
5	テーブル・レベルまたはロー・レベル：Mobile Link 同期サーバがアップロードまたはダウンロードでテーブル操作を実行するときに警告メッセージを表示

レポートする警告メッセージのレベルを指定する場合は、複数のレベルをカンマで区切るか、2 つのドットで範囲を指定できます。たとえば、**-zw 1..3,5** は、**-zw 1,2,3,5** と同じです。

メッセージのレポートはパフォーマンスにほとんど影響しません。レベル数が高いほど、多くのメッセージが生成される傾向があります。

同じコマンド・ラインで -zw を 2 回以上使用すると、Mobile Link は最後のインスタンスのみを認識します。-zw、-zwd、-zwe の設定が競合する場合は、-zwe、-zwd、-zw の優先順位で処理されます。

デフォルトは **1,2,3,4,5** です。この場合、すべてのレベルの警告メッセージが表示されます。

-zwd オプション

機能 特定の警告コードを無効にします。

構文 **dbmlsrv9 -c "***connection-string***" -zwd** *code,…*

説明

特定の警告コードを無効にすると、同じレベルの他のコードがレポートされる場合でも、その警告コードはレポートされません。

警告メッセージ・コードの完全なリストについては、『ASA エラー・メッセージ』> 「Mobile Link 同期サーバの警告メッセージ」を参照してください。

同じコマンド・ラインで -zwd を 2 回以上使用すると、Mobile Link は設定を累積します。-zw、-zwd、-zwe の設定が競合する場合は、-zwe、-zwd、-zw の優先順位で処理されます。

-zwe オプション

機能

特定の警告コードを有効にします。

構文

dbmlsrv9 -c "*connection-string***" -zwe** *code,…*

説明

特定の警告コードを有効にすると、-zw で同じレベルの他のコードを無効にしてある場合でも、その警告コードがレポートされます。

警告メッセージ・コードの完全なリストについては、『ASA エラー・メッセージ』> 「Mobile Link 同期サーバの警告メッセージ」を参照してください。

同じコマンド・ラインで -zwe を 2 回以上使用すると、Mobile Link は設定を累積します。-zw、-zwd、-zwe の設定が競合する場合は、-zwe、-zwd、-zw の優先順位で処理されます。

第 2 部　Mobile Link スクリプト論理

第 2 部では、Mobile Link スクリプト論理の使用方法について説明します。

第 12 章

同期スクリプトの作成

この章の内容

同期スクリプトを作成して統合データベースに格納することで、同期処理を制御できます。

SQL、Java、または .NET でスクリプトを作成できます。この章の内容はすべての種類のスクリプトに適用されますが、主に SQL で同期スクリプトを作成する方法について説明します。

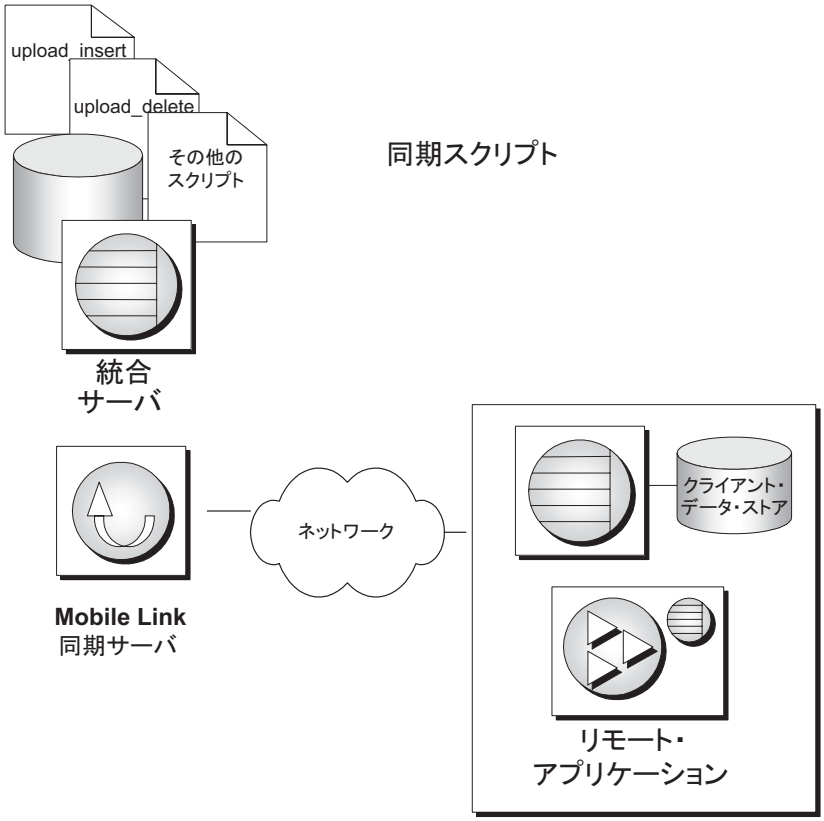
スクリプト作成の詳細については、以下の項目を参照してください。

- 「同期イベント」401 ページ
- 「同期の方法」57 ページ
- 「Java による同期スクリプトの作成」321 ページ
- 「.NET での同期スクリプトの作成」353 ページ

同期スクリプトの概要

「**Mobile Link 同期論理**」は、統合データベースに保管されているスクリプト (個別の文またはストアド・プロシージャ・コール) で構成されます。同期中は、Mobile Link 同期サーバがスクリプトを読み込み、統合データベースに対してスクリプトを実行します。スクリプトによって、同期処理中のさまざまな時点でタスクを実行することができます。統合データベースにスクリプトを追加するには、Sybase Central を使用します。また、ストアド・プロシージャも使用できます。

同期処理には、複数の手順があります。各手順は、ユニークな「イベント名」で識別されます。これらのイベントのいずれかに対応するスクリプトを作成することによって、同期処理を制御します。スクリプトは、特定のイベントで特定の動作を行う必要がある場合にのみ作成します。Mobile Link 同期サーバは、イベントが発生するとそれに対



応するスクリプトを実行します。特定のイベントに対してスクリプトを定義していない場合、Mobile Link 同期サーバは単に次の手順に進みます。

たとえば、begin_upload_rows というイベントを考えてみます。スクリプトを作成して、このイベントに関連付けることができます。Mobile Link 同期サーバは、このスクリプトを、必要となった時点で初めて読み込み、同期のアップロード・フェーズ中に実行します。スクリプトを作成しなかった場合、Mobile Link 同期サーバは直ちに次の手順、つまり、アップロードされたローを処理する手順に移ります。

「**テーブル・スクリプト**」と呼ばれるスクリプトは、イベントだけでなく、リモート・データベースの特定のテーブルにも対応します。Mobile Link 同期サーバは、いくつかのタスクをテーブル単位で実行します (たとえばローのダウンロード)。同じイベントでもアプリケーション・テーブルが異なれば、複数のスクリプトに対応することができます。または、いくつかのアプリケーション・テーブルに対しては複数のスクリプトを定義し、他のアプリケーション・テーブルに対しては何も定義しないこともできます。

イベントの概要については、「同期処理」18 ページを参照してください。

作成可能な各スクリプトの説明については、「同期イベント」401 ページを参照してください。

SQL、Java、または .NET でスクリプトを作成できます。この章の内容はすべての種類のスクリプトに適用されますが、主に SQL で同期スクリプトを作成する方法について説明します。

SQL、Java、.NET の説明と比較については、「同期論理の作成オプション」31 ページを参照してください。

.NET でのスクリプト作成については、「.NET での同期スクリプトの作成」353 ページを参照してください。

Java でのスクリプト作成については、「Java による同期スクリプトの作成」321 ページを参照してください。

同期スクリプトを実装する方法については、「同期の方法」57 ページを参照してください。

簡単な同期スクリプト

Mobile Link には利用可能なイベントがたくさんありますが、その各イベントに対してスクリプトを作成する必要はありません。簡単な同期モデルの場合、必要なスクリプトはわずかです。

テーブルから各リモート・データベースにすべてのローをダウンロードすると、CustDB サンプル・アプリケーションの ULProduct テーブルと同期します。この場合、リモート・データベースでの追加は許可されません。この簡単な形の同期は、1 つのスクリプトで実装できます。この例では、1 つのイベントだけに関連するスクリプトを使用します。

各同期中にダウンロードするローを制御する Mobile Link イベントは、download_cursor イベントと呼ばれます。カーソル・スクリプトには、SELECT 文が必要です。Mobile Link 同期サーバは、このクエリを使用してカーソルを定義します。download_cursor スクリプトの場合、カーソルによって、リモート・データベース内の特定の 1 テーブルにダウンロードするローが選択されます。

CustDB サンプル・アプリケーションには、このサンプル・アプリケーションにある ULProduct テーブルに対応する download_cursor スクリプトが 1 つあり、次のクエリから構成されています。

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

このクエリは結果セットを生成します。クライアントには、この結果セットを構成するローがダウンロードされます。この場合は、テーブルのすべてのローがダウンロードされます。

Mobile Link 同期サーバでは、ULProduct アプリケーション・テーブルへローを送信することを認識しています。これは、このスクリプトが download_cursor イベントと ULProduct テーブルの両方に対応するような方法で統合データベースに格納されているからです。Sybase Central ではこのような対応付けが可能です。

注意
この例では、クエリによって同じ ULProduct という名前の統合テーブルからデータが選択されます。この名前が同じである必要はありません。クエリを書き換えることで、統合データベース内の任意の単一または複数のテーブルのデータを ULProduct アプリケーション・テーブルにダウンロードするようにできます。

より複雑な同期スクリプトを作成することもできます。たとえば、最近修正されたローだけをダウンロードするスクリプトや、リモート・データベースごとに異なる情報を提供するスクリプトを作成できます。

スクリプトの自動生成

dbmlsrv9 -za オプションを使用して、デフォルトの同期スクリプトを生成できます。同期スクリプトは、クライアントから送信されたテーブル名とカラム名を使用して、リモート・データベースと統合データベースのスナップショット同期を実行します。

Adaptive Server Anywhere クライアントでこの機能を使用するには、SendColumnNames 拡張オプションを ON にして、dbmlsync がアップロード・ヘッダとともにカラム名を送信するように設定します。Ultra Light クライアントでこの機能を使用するには、send_column_names パラメータを ul_true に設定します。

-za を指定すると、存在しないスクリプト・バージョンで初めてリモートが同期するときにスクリプトが生成されます。指定したスクリプト・バージョンが存在する場合は、-za は機能しません。つまり、-za を使用して、同じスクリプト・バージョンのスクリプトを、一度に 1 テーブルずつ生成することはできません。-za を使用する場合は、すべてのテーブルとパブリケーションに対するスクリプトを一度に生成してください。

詳細については、「-za オプション」275 ページを参照してください。

例

-za スイッチを使用して Mobile Link 同期サーバを起動します。たとえば、次のように入力します。

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

dbmlsync を実行して、SendColumnNames 拡張オプションを ON に設定します。たとえば、次のように入力します。

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

パブリケーションに指定されたすべてのテーブルに対してスクリプトが生成されます。同期では、これらの自動生成スクリプトが、クライアントと統合データベース間のデータのアップロードとダウンロードを制御します。次の表は、emp テーブルで使用するサンプル・スクリプトのリストです。

スクリプト名	スクリプトの内容
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`

サンプル・スクリプトの生成

dbmlsrv9 -ze オプションを使用して、サンプル同期スクリプトを生成できます。サンプル同期スクリプトは、クライアントが送信したテーブル名とカラム名を使用して、統合データベースとリモート・データベースとのスナップショットを使った同期を実行できますが、使用不可に設定されています。統合データベースのテーブル名またはカラム名が異なっていると、サンプル同期スクリプトを有効にすることによって、同期中にエラーが発生します。

Adaptive Server Anywhere クライアントでこの機能を使用するには、SendColumnNames 拡張オプションを ON にして、dbmlsync がアップロード・ヘッダとともにカラム名を送信するように設定します。Ultra Light クライアントでこの機能を使用するには、send_column_names パラメータを ul_true に設定します。

-ze オプションは、サンプル・スクリプト example_upload_insert、example_upload_update、example_upload_delete、example_download_cursor を生成します。

詳細については、「-ze オプション」277 ページを参照してください。

例

次の例は、Adaptive Server Anywhere リモート・データベース用のスクリプトを生成します。

コマンド・プロンプトで次のコマンドを入力します。

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

コマンド・プロンプトで次のコマンドを入力します。

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

この例では、同期定義で指定したすべてのテーブルに対してサンプル・スクリプトが生成されます。スクリプトは、同期定義で指定したそれぞれのテーブルに対して生成されます。次の表は、emp テーブルで使用するサンプル・スクリプトのリストです。

スクリプト名	スクリプト
example_upload_insert	`INSERT INTO emp (emp_id,emp_name) VALUES (?,?)`
example_upload_update	`UPDATE emp SET emp_name=? WHERE emp_id=?`
example_upload_delete	`DELETE FROM emp WHERE emp_id=?`
example_download_cursor	`SELECT emp_id, emp_name FROM emp`

サンプル・スクリプトは、文で指定した条件を満たすすべてのレコードを同期サブスクリプション内のテーブルから選択してアップロードします。したがって、たとえば emp の upload_insert スクリプトの場合には emp からすべてのレコードが挿入されます。サンプル・スクリプトは、同期サブスクリプションで指定したリモート・データベース内のテーブルごとに生成されます。Mobile Link 同期サーバは、ス

ナップショットを使った同期に必要な完全なスクリプトを生成します。同期記述が処理されると、すぐにスクリプトが追加されます。スクリプトの生成後に、同期はアボートされます。

Ultra Light 用のサンプル・スクリプト

Ultra Light アプリケーションを構築すると、Ultra Light ジェネレータによって、Ultra Light リファレンス・データベースに example_download_cursor スクリプトが自動的に挿入されます。ダウンロード用サンプル・スクリプトのアクションは、リモート・データベースにある対応するテーブルのすべてのローをダウンロードするというものです。スクリプトは、select リストを適切な順序で指定しています。

サンプル・スクリプトは ml_scripts テーブルに挿入されますが、これを download_cursor イベントに関連付けるエントリを ml_table_script テーブルに挿入しないかぎり、使用されません。

少なくとも、ダウンロード・カーソル用のサンプル・スクリプトからは、リモート・データベースが予期するカラム順がわかります。

スクリプトと同期処理

各スクリプトは、同期処理の特定のイベントに対応します。特定の動作を行う必要がある場合にのみ、スクリプトを作成します。不要なイベントは、定義しないでおくことができます。

同期処理を大きく 2 つに分けると、アップロードされた情報の処理と、ダウンロードするためのローの準備があります。

Mobile Link 同期サーバは、各スクリプトを、必要となった時点で初めて読み込み、準備します。以後は、イベントが呼び出されるたびにスクリプトが実行されます。

イベントの順序

Mobile Link イベントの完全な順序については、『Mobile Link 管理ガイド』> 「Mobile Link イベントの概要」を参照してください。

アップロード・ストリームの処理の詳細については、「ローをアップロードするスクリプトの作成」305 ページを参照してください。

ダウンロード・ストリームの処理の詳細については、「ローをダウンロードするスクリプトの作成」308 ページを参照してください。

注意

- Mobile Link テクノロジによって、複数のクライアントを一度に同期させることができます。この場合、各クライアントは別々の接続を使用して統合データベースにアクセスします。
- 1 つの接続で複数の同期要求を処理できるので、begin_connection イベントと end_connection イベントは特定の同期に依存しません。これらのスクリプトには、パラメータがありません。これらは、接続レベルのスクリプトの例です。
- イベントの中には、各同期に対して 1 回だけ呼び出され、1 つのパラメータを持つものがあります。このパラメータは、同期処理中の Mobile Link クライアントをユニークに識別するユーザ名です。これらも、接続レベルのスクリプトの例です。
- 各テーブルが同期されるたびに 1 回ずつ呼び出されるイベントもあります。これらのイベントに対応するスクリプトは、テーブル・レベルのスクリプトと呼ばれます。これらのスクリプト

には、2 つのパラメータがあります。パラメータの 1 つは、同期関数の呼び出しで提供されるユーザ名で、もう 1 つは、同期されるリモート・データベース内のテーブル名です。

各テーブルは専用のテーブル・スクリプトを持つことができますが、いくつかのテーブルで共有されるテーブル・レベルのスクリプトを作成することもできます。

- begin_synchronization など、一部のイベントは接続レベルとテーブル・レベルの両方で発生します。これらのイベントに対しては、接続スクリプトとテーブル・スクリプトの両方を作成することができます。
- 同期処理がどのように複数のトランザクションに分散されるかについては、COMMIT 文が参考になります。
- エラーは、同期処理のあらゆる時点で発生する可能性がある特殊なイベントです。エラーは、次のスクリプトを使用して処理します。

```
handle_error( error_code, error_message,
user_name, table_name )
```

各スクリプトやそのパラメータなどのリファレンス情報については、「同期イベント」401 ページを参照してください。

スクリプトの種類

同期スクリプトは、接続全体または指定したテーブルに適用できます。

- **接続スクリプト**　接続専用または同期専用の、どのリモート・テーブルにも依存しないアクションを実行します。より複雑な同期スキームを実行する際は、このスクリプトを他のスクリプトと組み合わせて使用します。
- **テーブル・スクリプト**　1 つの同期と特定の 1 つのリモート・テーブル専用のアクションを実行します。より複雑な同期スキームを実行する際は、このスクリプトを他のスクリプトと組み合わせて使用します。

接続スクリプト

接続レベルのスクリプトは、特定のテーブルに関連付けられていない高いレベルのイベントを制御します。これらのイベントは、各同期の処理中に必要な全般的なタスクを実行するときに使用します。

接続スクリプトは、接続と切断を中心としてアクションを制御するほか、アップロード処理やダウンロード処理の開始と終了などの、同期レベルのイベントでのアクションを制御します。一部の接続スクリプトには、関連するテーブル・スクリプトがあります。これらの接続スクリプトは、テーブルが同期されているかどうかに関係なく、いつでも呼び出されます。

接続レベルのスクリプトは、特定のイベントで特定のアクションを実行する必要がある場合にのみ作成します。少数のイベント用のスクリプトだけを作成すれば済む場合もあります。あらゆるイベントに対する Mobile Link 同期サーバのデフォルト・アクションは、何もアクションを実行しない設定になっています。簡単な同期スキームの中には、接続スクリプトが必要でないものもあります。

テーブル・スクリプト

テーブル・スクリプトによって、ローのアップロードの開始や終了、競合の解決、ダウンロードするローの選択など、特定のテーブルの同期に関する特定のイベントでのアクションを実行できます。

テーブルの同期スクリプトは、統合データベースのあらゆるテーブルまたはテーブルの組み合わせを参照できます。この機能を使用して、1 つまたは複数の統合テーブルに格納されたデータを特定のリモート・テーブルに入れたり、1 つのリモート・テーブルからアップロードされたデータを統合データベースの複数のテーブルに格納したりできます。

テーブル名は一致しなくてもよい

リモート・データベースでのテーブル名と統合データベースでのテーブル名は、同じである必要はありません。Mobile Link 同期サーバは、ml_table システム・テーブルでリモート・テーブル名を検索し、テーブルに対応するスクリプトを特定します。

スクリプトのパラメータ

同期スクリプトのほとんどは、Mobile Link 同期サーバからパラメータを受け取ります。スクリプト内で疑問符を置くことによって、これらのパラメータをスクリプト内で使用できます。

次に、スクリプトで使用される共通パラメータの一部を示します。

- **last_download_timestamp**　成功した最後の同期中の、ダウンロード・フェーズの直前に統合データベースから取得した値です。現在の Mobile Link のユーザが同期を行ったことがない場合や同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

  詳細については、「タイムスタンプベースの同期」60 ページを参照してください。

- **ml_username**　このパラメータの値は、Mobile Link クライアントをユニークに識別する文字列です。Mobile Link 同期サーバで同期を開始するとき、各クライアントをこの名前で識別します。このパラメータは、接続レベルのほとんどのスクリプト、テーブル・レベルのすべてのスクリプト、いくつかのカーソル・スクリプトで使用できます。

  ユーザ名は、リモート・データベースでのテーブルの分割に使用できます。

- **table**　リモート・データベースのテーブルを識別するパラメータです。統合データベースには、同じ名前のテーブルがある場合とない場合があります。このパラメータは、テーブル・スクリプトでのみ使用します。

パラメータを使用するには、SQL スクリプトで各パラメータに対して 1 つ疑問符を置きます。オプションのパラメータもあります。Mobile Link 同期サーバが、各疑問符をパラメータ値に置き換えます。パラメータ値への置き換えは、スクリプト内でパラメータが定義されている順序に従って行われます。

各スクリプトやそのパラメータなどのリファレンス情報については、「同期イベント」401 ページを参照してください。

スクリプト・バージョン

スクリプトは、「**スクリプト・バージョン**」と呼ばれるグループに分けられます。バージョンを指定することによって、アップロード・ストリームの処理やダウンロード・ストリームの準備に使用する同期スクリプト・セットを Mobile Link クライアントで選択できます。

統合データベースにスクリプト・バージョンを追加する方法については、「スクリプト・バージョンの追加」299 ページを参照してください。

スクリプト・バージョンの使用方法

スクリプト・バージョンを使用すると、スクリプトを異なる環境で実行されるスクリプト・セットに編成することができます。この機能によって柔軟性が生まれ、特に次のような場合に便利です。

- **カスタマイズ**　さまざまなリモート・ユーザからの情報を処理するために、異なるスクリプト・セットを使用します。たとえば、組織のマネージャが自分が持っているデータベースを同期させるときに使用するスクリプト・セットを、他の人が使用するスクリプト・セットとは別に作成することができます。これと同じ機能を 1 つのスクリプト・セットで実現することもできますが、より複雑なスクリプトになってしまいます。

- **アプリケーションのアップグレード**　データベース・アプリケーションをアップグレードする場合、新しいスクリプトが必要になることがあります。これは、新しいバージョンのアプリケーションではデータ処理方法が異なる場合があるからです。リモート・データベースが変更される場合はほとんど、新しいスクリプトが必要になります。通常は、すべてのユーザを同時にアップグレードすることはできません。Mobile Link クライアントは、同期中に新しいスクリプト・セットの使用を要求することができます。古いスクリプトと新しいスクリプトの両方がサーバ上で共存できるため、使用するアプリケーションのバージョンに関わらず、すべてのユーザが同期できます。

- **複数のアプリケーション**　1 つの Mobile Link 同期サーバが 2 つの完全に異なるアプリケーションを同期しなければならない場合があります。たとえば、販売アプリケーションを使用する従業員がいる一方で、在庫管理用に設計されたアプリケーションが必要な従業員もいます。2 つのアプリケーションで異なるデー

タ・セットが必要な場合は、各アプリケーションにつき 1 つのバージョンとなるよう、2 つのバージョンの同期スクリプトを作成できます。

- **スクリプト・バージョンのプロパティの設定**　.NET または Java 同期論理のクラスから、参照可能なスクリプト・バージョンのプロパティを設定できます。詳細については、『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

バージョン名の割り当て

スクリプト・バージョン名は、文字列です。統合データベースにスクリプトを追加するときに、この名前を指定します。たとえば、ml_add_connection_script ストアド・プロシージャと ml_add_table_script ストアド・プロシージャを使用してスクリプトを追加する場合、スクリプト・バージョン名が 1 つ目のパラメータになります。また、Sybase Central を使用してスクリプトを追加する場合は、バージョン名を入力するように指示されます。

スクリプト・バージョンには、**ml_sis_1** または **ml_qa_1** という名前は使用できません。これらの名前は、Mobile Link によって内部的に使用されています。また、スクリプト・バージョンの名前は、**ml_** で始めないでください。

デフォルトのスクリプト・バージョン

リモート・サイトでスクリプト・バージョンが指定されなかった場合、Mobile Link 同期サーバでは ml_script_version テーブルに定義された 1 つ目のバージョンが使用されます。スクリプト・バージョンが定義されていない場合、同期は失敗します。

スクリプト・バージョンの追加

すべてのスクリプトはスクリプト・バージョンに関連付けられています。接続スクリプトを追加するには、まず統合データベースにバージョン名を追加してください。

詳細については、「スクリプト・バージョン」298 ページを参照してください。

❖ データベースにスクリプト・バージョンを追加するには、次の手順に従います (Sybase Central の場合)。

1. Sybase Central で [Mobile Link 同期] を右クリックして、統合データベースに接続します。
2. [バージョン] フォルダを開きます。
3. [バージョンを追加] をダブルクリックして、ウィザードの指示に従います。

❖ データベースからスクリプト・バージョンを削除するには、次の手順に従います (Sybase Central の場合)。

1. Sybase Central で [Mobile Link 同期] を右クリックして、統合データベースに接続します。
2. [バージョン] フォルダを開きます。
3. バージョン名を右クリックして、[削除] を選択します。
4. [削除の確認] ダイアログが表示されます。[はい] をクリックします。

❖ データベースにスクリプト・バージョンを追加するには、次の手順に従います (ストアド・プロシージャの場合)。

- スクリプト・バージョンは、接続スクリプトまたはテーブル・スクリプトを追加するのと同じ操作で追加できます。

詳細については、『Mobile Link 管理ガイド』> 「スクリプトを追加または削除するためのストアド・プロシージャ」を参照してください。

統合データベースでのスクリプトの追加と削除

スクリプトを作成したら、統合データベース内の Mobile Link システム・テーブルにそのスクリプトを追加してください。追加するには、ストアド・プロシージャまたは Sybase Central のウィザードを使用できます。

Mobile Link システム・テーブルの詳細については、「Mobile Link システム・テーブル」619 ページを参照してください。

注意 : SQL スクリプトは、統合データベースに格納されます。Java スクリプトと .NET スクリプトの場合、スクリプトのロケーションは統合データベースに格納されています。ただし、スクリプトまたはスクリプトのロケーションの追加方法と削除方法は、ほぼ同じです。

スクリプトの追加または削除

Sybase Central のウィザードを使用して同期スクリプトを追加できます。接続スクリプトとテーブル・スクリプトでは手順が異なります。テーブル・スクリプトはリモート・データベース内のテーブルに対応するため、リモート・データベース・テーブルの名前を統合データベースに追加しないと、テーブル・スクリプトを追加できません。

Sybase Central を使用している場合は、データベースに同期バージョンを追加してから、個々のスクリプトを追加してください。詳細については、「スクリプト・バージョンの追加」299 ページを参照してください。

❖ 接続スクリプトを追加または削除するには、次の手順に従います (Sybase Central の場合)。

1. Sybase Central で [Mobile Link 同期] を右クリックして、統合データベースに接続します。
2. [接続スクリプト] を開きます。
3. 接続スクリプトを追加するには、[接続スクリプトを追加] をダブルクリックして、ウィザードの指示に従います。

または

接続スクリプトを削除するには、スクリプト名を右クリックして、[削除] を選択します。[削除の確認] ダイアログが表示されます。[はい] をクリックします。

❖ 同期テーブルのリストでリモート・テーブルを追加または削除するには、次の手順に従います (Sybase Central の場合)。

1. Sybase Central で [Mobile Link 同期] を右クリックして、統合データベースに接続します。
2. [同期テーブル] を開きます。
3. リモート・テーブルを同期テーブルのリストに追加するには、[同期テーブルの追加] をダブルクリックします。同期スクリプトを作成するリモート・データベースのテーブル名を入力します。一致する名前を持つテーブルが統合データベースにあれば、ウィザードにショートカットが表示されます。

   または

   リモート・テーブルを同期テーブルのリストから削除するには、テーブル名を右クリックして、[削除] を選択します。[削除の確認] ダイアログが表示されます。[はい] をクリックします。

❖ データベースでテーブル・スクリプトを追加または削除するには、次の手順に従います (Sybase Central の場合)。

1. Sybase Central で [Mobile Link 同期] を右クリックして、統合データベースに接続します。
2. [同期テーブル] を開きます。
3. スクリプトを追加するテーブルを選択します。
4. テーブル・スクリプトを追加するには、[テーブル・スクリプトを追加] を右クリックして、ウィザードの指示に従います。

または

テーブル・スクリプトを削除するには、スクリプト名を右クリックして、[削除] を選択します。[削除の確認] ダイアログが表示されます。[はい] をクリックします。

❖ **すべてのタイプのスクリプトを追加または削除するには、次の手順に従います (ストアド・プロシージャの場合)。**

- 統合データベースの作成時に Mobile Link システム・テーブルとともにインストールされるストアド・プロシージャを使用して、スクリプトを統合データベースに追加、または統合データベースから削除することができます。

  スクリプトを追加または削除できるストアド・プロシージャについては、『Mobile Link 管理ガイド』> 「スクリプトを追加または削除するためのストアド・プロシージャ」を参照してください。

スクリプトの直接挿入

ほとんどの場合、ストアド・プロシージャまたは Sybase Central を使用してスクリプトをシステム・テーブルに挿入することをおすすめします。ただし、INSERT 文を使用してスクリプトを直接挿入することが必要な場合もあります。たとえば、旧バージョンの DBMS では、長さの制限があって、ストアド・プロシージャを使用するのが難しい場合があります。

Mobile Link システム・テーブルの完全な説明については、「Mobile Link システム・テーブル」619 ページを参照してください。

スクリプトを直接挿入するのに必要な INSERT 文のフォーマットは、ストアド・プロシージャ ml_add_connection_script と ml_add_table_script のソース・コードにあります。これらのストアド・プロシージャのソース・コードは、Mobile Link 設定スクリプトにあります。サポートされている各 RDBMS 用に個別の設定スクリプトがあります。設定スクリプトは次のとおりです。

統合データベース	設定ファイル
Adaptive Server Anywhere	*scripts¥syncasa.sql*
Oracle	*MobiLink¥setup¥syncora.sql*
IBM DB2 UDB	*MobiLink¥setup¥syncdb2long.sql*
Microsoft SQL Server	*MobiLink¥setup¥syncmss.sql*
Adaptive Server Enterprise バージョン 12.5 以降	*MobiLink¥setup¥syncase125.sql*
Adaptive Server Enterprise バージョン 12.5 より前	*MobiLink¥setup¥syncase.sql*

ローをアップロードするスクリプトの作成

リモート・データベースの情報を統合データベースにアップロードするには、アップロード・スクリプトを定義します。リモート・データベースで更新、挿入、または削除するローを処理する個別のスクリプトを作成します。簡単な実装で対応するアクション (更新、挿入、削除) を統合データベースで実行できます。

Mobile Link 同期サーバは、データを 1 つのトランザクションでアップロードします。アップロード処理の説明については、『Mobile Link 管理ガイド』> 「アップロード中のイベント」を参照してください。

.NET 同期論理でローをアップロードする方法については、「ローのアップロード」371 ページを参照してください。

注意

- 各リモート・テーブルの begin_upload スクリプトと end_upload スクリプトには、更新される個々のローには依存しない論理が保持されます。
- アップロード・ストリームは、1 ローずつの挿入、更新、削除から構成されます。通常、これらのアクションは、upload_insert、upload_update、upload_delete の各スクリプトを使用して実行されます。
- Adaptive Server Anywhere クライアント用のアップロードの準備において、dbmlsync ユーティリティは、正常に行われた最後の同期よりも後に書き込まれたすべてのトランザクション・ログにアクセスする必要があります。詳細については、『Mobile Link クライアント』> 「トランザクション・ログ・ファイル」を参照してください。

upload_insert スクリプトの作成

Mobile Link 同期サーバは、アップロード・ストリームの処理中にこのイベントを使用して、リモート・データベースに挿入されたローを処理します。次の INSERT 文は、upload_insert 文の使用方法を示しています。

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

詳細については、『Mobile Link 管理ガイド』> 「upload_insert テーブル・イベント」を参照してください。

upload_update スクリプトの作成

Mobile Link 同期サーバは、アップロード・ストリームの処理中にこのイベントを使用して、リモート・データベースで更新されたローを処理します。次の UPDATE 文は、upload_update 文の使用方法を示しています。

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

詳細については、『Mobile Link 管理ガイド』> 「upload_update テーブル・イベント」を参照してください。

upload_delete スクリプトの作成

Mobile Link 同期サーバは、アップロード・ストリームの処理中にこのイベントを使用して、リモート・データベースから削除されたローを処理します。次の文は、upload_delete 文の使用方法を示しています。

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

詳細については、『Mobile Link 管理ガイド』> 「upload_delete テーブル・イベント」を参照してください。

upload_fetch スクリプトの作成

upload_fetch スクリプトは、統合データベースのテーブルにカーソルを定義する SELECT 文です。このカーソルは、リモート・データベースから更新されたものとして受信したローの古い値と、統合データベースにある値を比較するために使用します。これによって、upload_fetch スクリプトは更新の処理中に競合を識別します。

同期テーブルが、次のように定義されている場合を考えてみます。

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

この場合、このテーブルに対して考えられる upload_fetch スクリプトは次のようになります。

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

詳細については、『Mobile Link 管理ガイド』> 「upload_fetch テーブル・イベント」を参照してください。

Mobile Link 同期サーバが、統合データベース内の競合のチェック対象となる 1 つのローを正確に識別するには、upload_fetch スクリプト内にクエリの WHERE 句が必要となります。

ローをダウンロードするスクリプトの作成

ダウンロード・トランザクション時に各テーブルの処理に使用できるスクリプトは、2 つあります。挿入と更新を実行する download_cursor スクリプトと、削除を実行する download_delete_cursor スクリプトです。

これらのスクリプトは、SELECT 文か、結果セットを返すプロシージャの呼び出しのどちらかです。Mobile Link 同期サーバは、スクリプトの結果セットをリモート・データベースにダウンロードします。Mobile Link クライアントは自動的に、download_cursor スクリプトの結果セットに基づいてローを挿入または更新し、download_delete_cursor イベントに基づいてローを削除します。

ストアド・プロシージャの使用の詳細については、「ストアド・プロシージャ・コールからの結果セットのダウンロード」97 ページを参照してください。

Mobile Link 同期サーバは、データを 1 つのトランザクションでダウンロードします。ダウンロード処理の説明については、『Mobile Link 管理ガイド』> 「ダウンロード中のイベント」を参照してください。

注意

- アップロード・ストリームと同様、ダウンロード・ストリームも接続イベントで開始、終了します。他のイベントは、テーブル・レベルのイベントです。
- SendDownloadAck 設定を ON に変更した場合は、クライアントからダウンロード確認を受け取らないと、統合データベースでダウンロード・トランザクション全体がロールバックされます (デフォルトでは、SendDownloadAck は OFF に設定されています)。

  詳細については、『Mobile Link クライアント』> 「SendDownloadACK (sa) 拡張オプション」または 『Mobile Link クライアント』> 「Send Download Acknowledgement 同期パラメータ」を参照してください。
- 各リモート・テーブルの begin_download スクリプトと end_download スクリプトには、更新される個々のローには依存しない論理が保持されます。

- ダウンロード・ストリームでは、挿入と更新が区別されません。download_cursor イベントに対応するスクリプトは、ダウンロードされるローを定義する SELECT 文です。クライアントは、ローが存在するかどうかを調べ、適切な挿入または更新オペレーションを実行します。

- 参照整合性違反を避けるために必要であれば、ダウンロード処理の最後にクライアントは自動的にローを削除します。

  詳細については、「参照整合性と同期」27 ページを参照してください。

download_cursor スクリプトの作成

統合データベースからリモート・データベースに情報をダウンロードするには、download_cursor スクリプトを作成します。このスクリプトは、変更をダウンロードするリモート・データベースの各テーブルに 1 つ作成してください。他のスクリプトを使用するとダウンロード処理をカスタマイズできますが、それらは必要ありません。

- 各 download_cursor スクリプトには、SELECT 文か、SELECT 文を含むプロシージャの呼び出しが必要です。Mobile Link 同期サーバは、SELECT 文を使用して統合データベース内でカーソルを定義します。

- スクリプトでは、対応するリモート・データベース内のテーブルのカラムに対応するすべてのカラムを選択します。統合データベース内のカラムは、対応するリモート・データベースのカラムとは異なる名前にできますが、互換性のある型にしてください。

- カラムは、対応するカラムがリモート・データベース内で定義されている順序に従って選択します。この順序は、リファレンス・データベースのカラム順と同じです。

例

次のスクリプトは、従業員情報を格納しているリモート・テーブルの download_cursor スクリプトとして機能します。Mobile Link 同期サーバは、この SQL 文を使用してダウンロード・カーソルを定義します。このスクリプトによって、すべての従業員についての情報がダウンロードされます。

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

Mobile Link 同期サーバは、スクリプトへ特定のパラメータを渡します。これらのパラメータを使用するには、SQL 文に疑問符を入れます。Mobile Link 同期サーバは、パラメータ値に置き換えてから、統合データベースに対して文を実行します。次のスクリプトは、これらのパラメータの使用方法を示しています。

```
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 = ?' )
```

この例では、Mobile Link 同期サーバによって疑問符が download_cursor スクリプトに対するパラメータ値に置き換えられます。

注意

- ローの値は、単一のテーブルまたは複数のテーブル間のジョインから選択できます。
- スクリプト自体にリモート・テーブルの名前を入れる必要はありません。リモート・テーブル名は、統合データベースのテーブル名と同じである必要はありません。リモート・テーブル名は、*ml_table* テーブル内のエントリによって示されます。Sybase Central では、リモート・テーブルがそのスクリプトとともにリストされます。
- リモート・テーブルのローには、*emp_id*、*emp_fname*、*emp_lname* の値を入れてください。リモート・カラムは、名前が異なっていてもかまいませんが、上記の順にしてください。リモート・データベース内のカラムは、リファレンス・データベース内のカラムと同じ順になります。

Ultra Light のヒント

サンプル・スクリプトでは、リファレンス・データベースで定義されている順序に従ってカラムがリストされます。カラム順は、example_download_cursor スクリプトと example_upload_cursor スクリプトを見て確認してください。

- カーソル・スクリプトでは、リモート・データベースで定義されている順序に従ってカラムを選択してください。統合データベースでカラム名やテーブル構造が異なる場合、リモート・データベース (リファレンス・データベースと同等) に対して正しい順序でカラムを選択してください。カラムは、SELECT 文内の順序に基づいてリモート・データベース内のカラムに割り当てられます。

- Ultra Light アプリケーションを構築する場合は、Ultra Light アプリケーションの各テーブルに対して Ultra Light ジェネレータがサンプル・ダウンロード・スクリプトを作成します。そして、これらのサンプル・スクリプトをリファレンス・データベースに挿入します。サンプル・スクリプトでは、統合データベースにアプリケーションと同じテーブルが含まれていることを仮定しています。統合データベースの設計が異なる場合は、サンプル・スクリプトを修正してください。その場合でも、これらのスクリプトを基にすることができます。

参照

- 「download_cursor テーブル・イベント」464 ページ
- 「リモート・データベース間でローを分割する」65 ページ
- 「download_delete_cursor スクリプトの作成」311 ページ

download_delete_cursor スクリプトの作成

リモート・データベースからローを削除するには、download_delete_cursor スクリプトを作成します。このスクリプトは、同期中に削除するローを含む、リモート・データベースの各テーブルに対して 1 つ作成してください。

統合データベースからのローの削除のみを実行し、リモート・データベースからローを消去することはできません。削除されたローのプライマリ・キーを download_delete_cursor で選択できるようにするため、そのローのプライマリ・キーを追跡する必要があります。こうするには、以下の 2 つの一般的な方法があります。

- **論理的削除** 統合データベースのローを物理的に削除しないでください。その代わり、ローが有効であるかどうかを追跡するステータス・カラムを使用します。こうすると、download_delete_cursor を簡略化できます。ただし、ステータス・カラムを認識して使用できるように、download_cursor とその他のアプリケーションを修正しなければならない場合があります。削除の時刻を保持する最終変更カラムが存在し、各リモートの最終ダウンロード時刻の追跡を行っている場合は、すべてのリモートのダウンロード時刻が削除の時刻よりも早ければ、ローを物理的に削除できます。

- **シャドー・テーブル** 削除に関する追跡を実行する各テーブルには、テーブルのプライマリ・キーを保持するカラムとタイムスタンプを保持するカラムの 2 つを含むシャドー・テーブルを作成します。ローが削除されたときにプライマリ・キーとタイムスタンプをシャドー・テーブルに挿入するトリガを作成します。こうすると、download_delete_cursor はこのシャドー・テーブルから選択できるようになります。論理的削除と同様に、すべてのリモート・データベースによってローがダウンロードされれば、シャドー・テーブルからローを削除できます。

Mobile Link 同期サーバは、統合データベースからプライマリ・キー値を選択し、それらをリモート・データベースに渡すことによって、リモート・データベースのローを削除します。値がリモート・データベース内のプライマリ・キーの値と一致する場合、そのローを削除します。

- 各 download_delete_cursor スクリプトには、SELECT 文か、結果セットを返すストアド・プロシージャの呼び出しが必要です。Mobile Link 同期サーバは、SELECT 文を使用して統合データベース内でカーソルを定義します。

- SELECT 文では、リモート・データベース内のテーブルのプライマリ・キー・カラムに対応するすべてのカラムを選択します。統合データベース内のカラムは、対応するリモート・データベースのカラムとは異なる名前にできますが、互換性のある型にしてください。

- 値は、対応するカラムがリモート・データベース内で定義されている順序に従って選択します。この順序は、テーブルの作成に使用される CREATE TABLE 文のカラム順と同じですが、プライマリ・キーを定義する文中のカラム順とは異なります。

各 download_delete_cursor スクリプトでは、対応するリモート・テーブルのプライマリ・キーに存在するすべてのカラム値を選択する必要がありますが、その他のすべてのカラムを選択することも可能です。この機能は、古いクライアントとの互換性を保つためだけにあります。追加のカラムを選択すると、データベース・エンジンが検索しなければならないデータが増え、効率が悪くなります。古いクライアント以外については、Mobile Link 同期サーバが追加の値を即座に破棄します。追加の値は、古いクライアントにのみダウンロードされます。

テーブルから全ローを削除

Mobile Link はすべての NULL を含むローの download_delete_cursor を検出すると、リモート・データベース内のデータをすべて削除します。download_delete_cursor 内の NULL の数は、プライマリ・キー・カラムの数またはテーブル内のカラムの総数です。

たとえば、次の download_delete_cursor SQL スクリプトは、2 つのプライマリ・キー・カラムがあるテーブル内のすべてのローを削除します。この例は、Adaptive Server Anywhere、Adaptive Server Enterprise、Microsoft SQL Server のデータベースに適用されます。

```
SELECT NULL, NULL
```

IBM DB2 と Oracle の統合データベースでは、ダミー・テーブルを指定して NULL を選択してください。IBM DB2 7.1 では、次の構文を使用できます。

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

Oracle 統合データベースでは、次の構文を使用できます。

```
SELECT NULL FROM DUAL
```

例

次の例は、従業員情報を格納しているリモート・テーブルに対する download_delete_cursor スクリプトです。Mobile Link 同期サーバは、この SQL 文を使用して削除カーソルを定義します。このスクリプトは、スクリプトの実行時に統合データベースとリモート・データベースの両方に存在するすべての従業員の情報を削除します。

```
SELECT emp_id
 FROM employee
```

download_delete_cursor は、パラメータ last_download と ml_username を受け入れます。次のスクリプトは、各パラメータを使用して選択範囲を絞る方法を示しています。

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

上記の例は、従業員数の多い組織では効率的ではない場合があるので注意してください。リモート・データベースにあるローだけを選択すると、削除処理をより効率的に行えます。たとえば、最近新しくマネージャになった人だけを選択すると、ロー数を制限することができます。また、クライアント・アプリケーションにロー自体を削除させるという方法もあります。この方法は、不必要なローをルールによって識別できる場合のみ可能です。たとえば、有効期限を示すタイムスタンプがローに含まれている場合があります。次回の同期時にこれらの削除がアップロードされるのを停止するには、STOP SYNCHRONIZATION DELETE 文を使用してからローを削除します。他の削除を通常の方法で同期する場合は、この後ですぐに START SYNCHRONIZATION DELETE を実行してください。

参照

特に効率良くローを削除するには、すべての Mobile Link クライアントに組み込まれている参照整合性検査を使用します。詳細については、「参照整合性と同期」27 ページ参照してください。

download_delete_cursor スクリプトの使用の詳細については、以下を参照してください。

- 『Mobile Link 管理ガイド』> 「download_delete_cursor テーブル・イベント」
- 『Mobile Link クライアント』> 「削除同期の一時停止」

- 『ASA SQL リファレンス・マニュアル』> 「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」
- 「削除の処理」91 ページ
- 『Mobile Link 管理ガイド』> 「download_cursor テーブル・イベント」
- 「リモート・データベース間でローを分割する」65 ページ
- 「スナップショットを使った同期」62 ページ

エラーを処理するスクリプトの作成

Mobile Link 同期サーバがスクリプト内のオペレーションを実行しているとき、そのオペレーションに失敗すると、同期スクリプトのエラーが発生します。DBMS は、エラーの内容を示す SQLCODE を Mobile Link 同期サーバに返します。統合データベースの各 DBMS は、独自の SQLCODE 値を持っています。

エラーが発生すると、handle_error イベントが呼び出されます。エラーを処理するには、このイベントに対応する接続スクリプトを定義してください。このスクリプトには、Mobile Link 同期サーバから、エラーの性質とコンテキストについての情報を提供するいくつかのパラメータが渡され、エラーへの対処方法を指示する出力値が要求されます。

エラー処理アクション

エラー処理スクリプト内で指定できるアクションには次のものがあります。

- 別のテーブルにエラーのログを取る
- Mobile Link 同期サーバに、エラーを無視して続行、同期のロールバック、同期をロールバックして Mobile Link 同期サーバを停止、のいずれかを指示する
- 電子メール・メッセージを送信する

詳細については、『Mobile Link 管理ガイド』> 「handle_error 接続イベント」を参照してください。

エラーのレポート

エラーによって同期処理の正常な進行が中断されるので、エラーとその解決方法の記録を継続的に作成することは困難です。report_error スクリプトは、このタスクを実行する手段を提供します。Mobile Link 同期サーバはエラーが発生すると常に、このスクリプトを実行します。handle_error スクリプトが定義されている場合は、レポート・スクリプトに先立って即座にそれが実行されます。

report_error スクリプトのパラメータは、handle_error スクリプトのパラメータと同じです。report_error スクリプトではアクション・コードを変更できない点だけが異なります。アクション・コードの値は、handle_error スクリプトによって返されます。したがって、エラー処理問題のデバッグにこのスクリプトを使用できます。

このスクリプトは通常、後で参照できるように、時間や日付などといった他のデータとともに値をテーブルに記録する INSERT 文から成ります。このデータが失われないようにするために、Mobile Link 同期サーバは常にこのスクリプトを別のトランザクションで実行し、スクリプトが完了すると直ちに変更を自動的にコミットします。

詳細については、『Mobile Link 管理ガイド』> 「report_error 接続イベント」を参照してください。

例

次の report_error スクリプトは 1 つの INSERT 文で構成されています。このスクリプトは、現在の日付と時刻とともにスクリプトのパラメータをテーブルに追加します。スクリプトはこの変更をコミットしません。通常、Mobile Link 同期サーバが自動的に行ってくれるからです。

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

1 つの SQL 文で複数のエラーが処理される場合

ODBC では、1 つの SQL 文につき複数のエラーの処理が可能で、DBMS にはこの機能を利用しているものがあります。たとえば、Microsoft SQL Server では、1 文で 2 つのエラーが発生することがあります。1 番目は実際のエラーで、通常 2 番目は現在の文が終了した理由を説明する情報メッセージです。

1 つの SQL 文で複数のエラーが発生した場合、1 エラーにつき 1 つの handle_error スクリプトが呼び出されます。Mobile Link 同期サーバは、最も厳しいアクション・コード (つまり、一番大きな番号) を使用して、どのアクションをとるかを決定します。同じことが handle_error スクリプトにも適用されます。

handle_error スクリプト自体で SQL エラーが発生した場合は、デフォルトのアクション・コード (3000) と見なされます。

スクリプト構文のテスト

同期スクリプトの開発時に、Sybase Central を使用してスクリプトの構文エラーをテストすることができます。

スクリプトのテストは、リモート・サイトを用意しなくても行えます。テスト中は、データがデータベースに追加されたり、データベースからダウンロードされたりすることはありません。同期されるデータ自体の妥当性はテストされません。

❖ 同期スクリプトをテストするには、次の手順に従います。

1. Sybase Central を起動して、Mobile Link 同期からデータベースに接続します。
2. 左ウィンドウ枠で、データベース名をクリックします。
3. [同期テーブル] フォルダまたは [接続スクリプト] フォルダを右クリックして、ポップアップ・メニューから [スクリプトのテスト] を選択します。

   [スクリプトのテスト] ウィンドウが表示されます。

4　[テスト] をクリックします。プロンプトが表示されたら、パラメータを入力します。テストの結果はウィンドウに表示されます。

テスト結果には、どのスクリプトがどの順序で実行されたかを示すリストが含まれます。また、テスト中に発見された構文エラーやデータ型エラーもリストされます。

第 13 章

Java による同期スクリプトの作成

この章の内容

Mobile Link 同期サーバの動作を制御するには、同期スクリプトを作成します。これらのスクリプトの実装には、SQL、.NET または Java を使用できます。この章では、Java を使用して同期スクリプトを実装する方法について説明します。

Java 同期スクリプトを使用するチュートリアルについては、『Mobile Link チュートリアル』> 「チュートリアル : Adaptive Server Anywhere での Java 同期論理」を参照してください。

SQL、Java、.NET の説明と比較については、「同期論理の作成オプション」31 ページを参照してください。

スクリプト作成については、「同期スクリプトの作成」285 ページを参照してください。

.NET でのスクリプト作成については、「.NET での同期スクリプトの作成」353 ページを参照してください。

概要

Mobile Link では同期スクリプトを Java で作成することができます。Java 同期論理には、SQL 論理と同じ機能を持たせることができます。Mobile Link 同期サーバは、Mobile Link イベントの発生時に SQL スクリプトにアクセスできるのと同様に、Java メソッドを呼び出すことができます。Mobile Link には、Java メソッドとして SQL 文字列が返されます。

この項では、Java 同期論理を設定、開発、実行する方法について説明します。また、サンプル・アプリケーションと Mobile Link Java API も含まれています。

Java 同期スクリプトを使用するチュートリアルについては、『Mobile Link チュートリアル』> 「チュートリアル：Adaptive Server Anywhere での Java 同期論理」を参照してください。

Java 同期論理の設定

SQL Anywhere Studio をインストールすると、インストーラによって Mobile Link Java API クラスのロケーションが自動的に設定されます。これらのクラスは、Mobile Link 同期サーバの起動時にクラスパスに自動的に組み込まれます。Mobile Link Java API クラスは、SQL Anywhere Studio のインストール・ディレクトリにある *java¥mlscript.jar* にインストールされます。Mobile Link では、ASANYSH9 環境変数を使用して共有コンポーネント・パスを判断します。

詳細については、『ASA データベース管理ガイド』> 「ASANYSH9 環境変数」を参照してください。

❖ Java を使用して同期スクリプトを実装するには、次の手順に従います。

1. 1 つまたは複数の独自クラスを作成します。必要な同期スクリプトごとに、メソッドを作成します。これらのメソッドは、パブリックにしてください。クラスは、パッケージ内ではパブリックにしてください。

   メソッドの詳細については、「メソッド」329 ページを参照してください。

   静的でないメソッドを持つ各クラスには、パブリック・コンストラクタが必要です。Mobile Link 同期サーバは、各クラスのメソッドが初めて呼び出されるときに、そのクラスを自動的にインスタンス化します。

   コンストラクタの詳細については、「コンストラクタ」328 ページを参照してください。

2. クラスをコンパイルするときは、JAR ファイル *java¥mlscript.jar* をインクルードしてください。

   次に例を示します。

```
javac MyClass.java -classpath
%ASANY9%¥java¥mlscript.jar
```

3 統合データベース内で、各同期スクリプトについて、呼び出すパッケージ、クラス、メソッドの名前を指定します。スクリプトのバージョンごとに、クラスを 1 つずつ使用します。

この情報を Mobile Link システム・テーブルに追加するには、ml_add_java_connection_script ストアド・プロシージャまたは ml_add_java_table_script ストアド・プロシージャを使用するのが最も簡単な方法です。

たとえば、次の SQL 文は、Adaptive Server Anywhere データベース内で実行すると、スクリプト・バージョン ver1 に対して、authenticate_user 接続レベル・イベントが発生するたびに myPackage.myClass.myMethod を実行するように指定します。指定されるメソッドはパブリック Java メソッドの完全修飾名で、大文字と小文字が区別されます。

```
call ml_add_java_connection_script( 'ver1',
 'authenicate_user', 'myPackage.myClass.myMethod' )
```

スクリプト追加の詳細については、以下の項目を参照してください。

- 「スクリプトを追加または削除するためのストアド・プロシージャ」594 ページ
- 『Mobile Link 管理ガイド』> 「ml_add_java_connection_script」
- 『Mobile Link 管理ガイド』> 「ml_add_java_table_script」

4 Mobile Link サーバがクラスをロードするよう指定します。Java 同期論理の設定の中で最も重要な部分は、Java クラスの検索場所を仮想マシンに対して指定することです。このようにするには、次の 2 つの方法があります。

- クラスを検索するディレクトリまたは jar ファイルを指定するには、dbmlsrv9 -sl java -cp オプションを使用します。たとえば、コマンド・ラインで次のように入力します。

  ```
  dbmlsrv9 -c "dsn=consolidated1" -sl java (-cp
  %classpath%;c:¥local¥Java¥myclasses.jar)
  ```

Mobile Link 同期サーバは、一連のディレクトリまたは jar ファイルに Mobile Link Java API クラスのロケーション (java\mlscript.jar) を自動的に追加します。また、-sl java オプションは、Java VM がサーバの起動時にロードされるよう指定します。

使用可能な Java オプションの詳細については、『Mobile Link 管理ガイド』> 「-sl java オプション」を参照してください。

- 明示的にクラスパスを設定します。ユーザ定義クラスのクラスパスを設定するには、次のような文を使用します。

```
SET
classpath=%classpath%;c:¥local¥Java¥myclasses.j
ar
```

システムのクラスパスに Java 同期論理のクラスが含まれている場合、Mobile Link 同期サーバのコマンド・ラインを変更する必要はありません。

-sl java オプションを使用すると、サーバ起動時に Java 仮想マシンを強制的にロードできます。このオプションを使用しない場合は、Java メソッドが最初に実行されるときに Java 仮想マシンが起動します。

使用可能な Java オプションの詳細については、『Mobile Link 管理ガイド』> 「-sl java オプション」を参照してください。

5　UNIX で特定の JRE をロードする場合は、LD_LIBRARY_PATH (AIX の場合は LIBPATH、HP-UX の場合は SHLIB_PATH) を設定して、JRE を含むディレクトリを指定します。ディレクトリは、すべての Adaptive Server Anywhere インストール・ディレクトリの前に指定します。

Java 同期論理の作成

Java 同期論理の作成は、他の Java コードの作成と違いはありません。必要となるのは、Mobile Link イベントの知識、Java に関する若干の知識、Mobile Link Java API の知識です。これ以降の項を参考にして、役立つ同期論理を作成してください。

Java 同期論理を使用すると、ステータス情報を管理し、アップロード・イベントとダウンロード・イベント関連の論理を実装できます。たとえば、Java で作成された begin_synchronization スクリプトを使用すると、Mobile Link ユーザ名を変数に格納できます。同期処理中に後で呼び出されるスクリプトは、この変数にアクセスできます。また、Java は、コミットの実行前または実行後に統合データベースのローにアクセスするために使用できます。

Java を使用すると、統合データベースへの依存度が減少します。また、統合データベースを新バージョンにアップグレードしたり、別のデータベース管理システムに切り替えたりする場合も、動作に与える影響が少なくて済みます。

API の完全な説明については、「Mobile Link Java API リファレンス」344 ページを参照してください。

クラス・インスタンス

Mobile Link 同期サーバは、クラスを接続レベルでインスタンス化します。静的でない Java メソッドをあるイベントに対して作成した場合、そのイベントに達すると、現在の接続でクラスが構成されていなければ、Mobile Link 同期サーバが自動的にクラスを構成します。

詳細については、「コンストラクタ」362 ページを参照してください。

1 つのスクリプト・バージョンの接続レベル・イベントまたはテーブルレベル・イベントに直接関連するすべてのメソッドは、*同じクラスに属している必要があります。*

データベース接続ごとに、インスタンス化されたクラスはその接続が終了するまで持続します。したがって、連続する複数の同期セッションに、引き続き同じインスタンスを使用できます。そのため、パブリック変数またはプライベート変数内の情報は、明示的にクリアしないかぎり、同じ接続で発生するすべての同期を通して持続します。

また、静的なクラスや変数も使用できます。この場合、値はすべての接続を通して使用できます。

統合データベースへの接続が終了した場合にのみ、Mobile Link 同期サーバはクラス・インスタンスを自動的に削除します。

トランザクション

Java メソッドには、トランザクションに関する通常のルールが適用されます。データベース・トランザクションの開始と継続期間は、同期処理に重要になります。トランザクションの開始と終了は、Mobile Link 同期サーバのみが行います。Java メソッド内の同期接続でトランザクションを明示的にコミットまたはロールバックすると、同期処理の整合性違反になり、エラーが発生することがあります。

これらのルールは、Mobile Link 同期サーバによって作成されるデータベース接続、特に、メソッドから返される SQL 文にのみ適用されます。クラスで他のデータベース接続を作成する場合は、自由に管理できます。

SQL データ型と Java データ型

次の表は、SQL データ型とそれに対応する Java データ型を示します。

SQL データ型	対応する Java データ型
VARCHAR	java.lang.String
CHAR	java.lang.String
INTEGER	Int または Integer
BINARY	byte[ ]

SQL データ型	対応する Java データ型
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

このパッケージが存在しない場合は、Mobile Link 同期サーバがクラスパスに自動的に追加します。

ただし、クラスをコンパイルするときは、SQL Anywhere Studio のインストール・ディレクトリから java\mlscript.jar のパスを追加する必要があります。

コンストラクタ

クラスのコンストラクタは、次の 2 つのシグニチャのどちらかを持ちます。

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

または

```
public MyScriptClass ( )
```

渡される同期コンテキストは、Mobile Link 同期サーバが現在のユーザの同期に使用している接続です。

DBConnectionContext クラスの getConnection メソッドは、Mobile Link が現在のユーザの同期に使用しているのと同じデータベース接続を返します。この接続で文を実行することはできますが、トランザクションのコミットやロールバックは行わないでください。トランザクションは Mobile Link 同期サーバによって管理されます。

Mobile Link 同期サーバは、最初のシグニチャを持つコンストラクタを使用しようとします。引数のないコンストラクタが使用されるのは、最初のシグニチャを持つコンストラクタが存在しない場合のみです。

詳細については、「DBConnectionContext インタフェース」344 ページを参照してください。

メソッド

通常は、同期イベントごとにメソッドを 1 つずつ実装します。これらのメソッドは、パブリックにしてください。プライベート・メソッドの場合、Mobile Link 同期サーバでは使用できず、その存在を認識できません。

統合データベース内の ml_script テーブルで指定されている名前と一致していれば、メソッド名は重要ではありません。ただし、このマニュアルの例では、メソッド名は Mobile Link イベント名と同じです。これは、Java コードを読みやすくするためです。

メソッドのシグニチャは、そのイベント用スクリプトのシグニチャと一致していなければなりません。ただし、パラメータ・リストの最後にパラメータ値が必要でない場合は、リストをトランケートできます。実際には、パラメータを渡すとオーバヘッドが生じる可能性があるため、必要なパラメータのみを受け入れてください。

ただし、メソッドはオーバロードできません。ml_script システム・テーブルには、クラスごとに 1 つのメソッド・プロトタイプのみが格納されます。

戻り値

Mobile Link アップロードまたはダウンロードに対して呼び出されるメソッドは、有効な SQL 言語の文を返さなければなりません。これらのメソッドの戻り型は、java.lang.String にしてください。他の戻り型は使用できません。

他のすべてのスクリプトの戻り型は、java.lang.String または void にします。他の型は使用できません。戻り型が null ではなく文字列の場合、Mobile Link 同期サーバはその文字列に有効な SQL 文が含まれているとみなして、この文を通常の SQL 言語による同期スクリプトと

同様に統合データベース内で実行します。通常、メソッドは文字列を返しますが、返却時にデータベースに対して SQL 文を実行しない場合は null を返すことができます。

Java クラスのデバッグ

Mobile Link は、Java コードのデバッグ時に役立つ各種の情報と機能を提供します。この項では、これらの情報の格納場所と機能の活用方法について説明します。

Mobile Link 同期サーバのログ・ファイル内の情報

Mobile Link 同期サーバは、各種の関連情報を出力ログ・ファイルに書き込みます。同期サーバのログ・ファイルには、次の情報が書き込まれます。

- Java Runtime Environment。-jrepath オプションを使用すると、Mobile Link 同期サーバの起動時に、特定の JRE を要求できます。デフォルトは、Adaptive Server Anywhere 9 でインストールされているパスです。
- ロードされている標準 Java クラスのパス。これらのパスを明示的に指定しない場合、Mobile Link 同期サーバはクラスパスに自動的に追加してから、Java 仮想マシンを起動します。
- 呼び出された特定のメソッドの完全指定名。この情報を使用すると、Mobile Link 同期サーバが適切なメソッドを呼び出していることを確認できます。
- Java メソッドで java.lang.System.out または java.lang.System.err に書き込まれた出力。すべて Mobile Link 同期サーバのログ・ファイルにリダイレクトされます。
- dbmlsrv9 コマンド・ラインのオプション。-verbose を使用できます。

  詳細については、『Mobile Link 管理ガイド』> 「-sl java オプション」を参照してください。

Java デバッガの使用

標準 Java デバッガを使用して、Java クラスをデバッグできます。dbmlsrv9 コマンド・ラインで -sl java オプションを使用して、必要なパラメータを指定します。

詳細については、『Mobile Link 管理ガイド』> 「-sl java オプション」を参照してください。

デバッガを指定すると、Java 仮想マシンは一時停止し、Java デバッガからの接続を待機します。

Java からの情報の出力

もう 1 つの方法として、Java.Lang.System.err か Java.Lang.System.out を使用して、Mobile Link 出力ログに情報を出力する文を Java メソッドに追加することもできます。これにより、クラスの進行状況と動作を追跡できます。

パフォーマンスのヒント
この方法で情報を出力すると、モニタ・ツールとして活用できますが、運用環境では推奨いたしません。

これと同じ方法を利用して、任意の同期情報のログを取ったり、スクリプトの使用方法に関する統計情報を収集したりできます。

独自のテスト・ドライバの作成

独自のドライバを作成して、Java クラスをテストできます。このアプローチでは、Java メソッドのアクションが Mobile Link システムの残りの部分から分離されるため便利な場合があります。

Java での Mobile Link サーバ・エラーの処理

ログをスキャンするだけでは不十分な場合は、アプリケーションをプログラムからモニタできます。たとえば、特定のタイプのメッセージを電子メールで送信できます。

ログに出力される各エラー・メッセージまたは各警告メッセージを表すクラスに渡されるメソッドを作成することも可能です。この方法は、Mobile Link 同期サーバをモニタおよび監査するのに役立ちます。

次のコードは、すべての警告メッセージ用に LogListener をインストールし、その情報をファイルに書き込みます。

```
class TestLogListener implements LogListener {
         FileOutputStream    _out_file;

         public TestLogListener(   FileOutputStream
out_file )
         {
             _out_file       = out_file;
         }


        public void messageLogged(  ServerContext   sc,
                                    LogMessage      msg )
         {
             String  type;
             String  user;
             try {
                 if(msg.getType() == LogMessage.ERROR) {
                     type = "ERROR";
                 } else if(msg.getType() ==
LogMessage.WARNING) {
                     type = "WARNING";
                 } else {
                     type = "UNKNOWN!!!";
                 }


                 user = msg.getUser();
                 if( user == null ) {
                     user = "NULL";
                 }
                 _out_file.write(
                         ("Caught msg type=" + type +
                                 " user=" + user +
                              " text=" +msg.getText() +
                                 "¥n").getBytes() );
                 _out_file.flush();
             } catch( Exception e ) {
             // print some error output to the MobiLink
log
                 e.printStackTrace();
             }
         }
     }
```

```
 // This line of code will register TestLogListener to
receive
 // warning messages.  Call this code from anywhere that
has
 // access to the ServerContext such as a class
constructor or
 // synchronization script.  ServerContext
serv_context;
 serv_context.addWarningListener(
       new MyLogListener( ll_out_file ));
```

参照

「ServerContext インタフェース」347 ページの addErrorListener、removeErrorListener、addWarningListener、removeWarningListener

「LogListener インタフェース」346 ページ

「LogMessage クラス」346 ページ

ユーザ定義起動クラス

サーバの起動時に自動的にロードされる起動クラスを定義できます。この機能の目的は、最初の同期の前に Mobile Link サーバが JVM を起動する時点で実行される Java コードを記述できるようにすることです。つまり、ユーザ同期要求の前に、接続の作成またはデータのキャッシュを実行できます。

この操作を行うには、dbmlsrv9 -sl java オプションの DMLStartClasses オプションを使用します。たとえば、次に示すのは dbmlsrv9 コマンド・ラインの一部です。mycl1 と mycl2 が起動クラスとしてロードされます。

```
-sl java(-
DMLStartClasses=com.test.mycl1,com.test.mycl2)
```

クラスはリスト内の順序でロードされます。同じクラスがリストに 2 回以上指定されている場合は、複数のインスタンスが作成されます。

起動クラスはすべてパブリックにしてください。また、引数を受け付けないか、ianywhere.ml.script.ServerContext 型の引数を 1 つ受け入れるパブリック・コンストラクタが必要です。

ロードされた起動クラスの名前は、「Java 起動クラス *classname* がロードされました。」というメッセージとともに Mobile Link ログに出力されます。

Java 仮想マシンのオプションの詳細については、『Mobile Link 管理ガイド』> 「-sl java オプション」を参照してください。

サーバ起動時に構築される起動クラスを表示する方法については、「getStartClassInstances」348 ページ を参照してください。

例

次に示すのは、テンプレート起動クラスです。これは、イベントを処理してデータベース接続を確立するデーモン・スレッドを起動します (すべての起動クラスがスレッドの作成を必要とするわけではありませんが、スレッドを作成する場合はデーモン・スレッドでなければなりません)。

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

 public class StartTemplate extends
    Thread implements ShutdownListener {
 //==================================================
   ServerContext    _sc;
   Connection       _conn;
   boolean          _exit_loop;

   public StartTemplate( ServerContext sc )
   //=============================================
                          throws SQLException
   {
     // perform setup first so that an exception will
     // cause MobiLink startup to fail
  _sc        = sc;
     // create a connection for use later
     _conn       = _sc.makeConnection();
     _exit_loop = false;
     setDaemon( true );
     start();
   }
```

```
 public void run()
  //===============
  {
    _sc.addShutdownListener( this );
    // we can't throw any exceptions through run()
    try {
       handlerLoop();
       _conn.close();
       _conn = null;
    } catch( Exception e ) {
       // print some error output to the MobiLink log
       e.printStackTrace();
       // we will die so we don't need to be notified
       // of shutdown
       _sc.removeShutdownListener( this );
       // ask server to shutdown so that this fatal
       // error will be fixed
       _sc.shutdown();
    }
    // shortly after return this thread will no longer
    // exist
    return;
  }


public void shutdownPerformed( ServerContext sc )
  //==============================================
  // stop our event handler loop
  {
    try {
       // wait max 10 seconds for thread to die
  join( 10*1000);
    } catch( Exception e ) {
       // print some error output to the MobiLink log
       e.printStackTrace();
   }
  }

 private void handlerLoop()
  //===============throws InterruptedException
  {
      while( !_exit_loop ) {
        // Handle events in this loop. Sleep not
```

```
                // needed, we should block on event queue.
                sleep( 1*1000 );
            }
        }
    }
```

Java 同期の例

Java 同期論理は Mobile Link や共通 Java クラスと連動し、Mobile Link 同期サーバを使用したアプリケーションの配備に柔軟性を提供します。次の項では、単純な例を使用して、この広範囲な機能について説明します。

注意：
この項では、基本の Java 同期論理がわかる簡単な例を提供します。一般に、カスタム・ユーザ認証メカニズムを使用する理由には、既存の DBMS ユーザ認証スキームと統合したり、最小パスワード長やパスワード期限などのカスタム機能を提供することが挙げられます。Mobile Link にも組み込みのメカニズムが用意されています。

Mobile Link 認証の詳細については、『Mobile Link クライアント』> 「ユーザ認証メカニズムの選択」を参照してください。

概要

この項では、Java 同期論理の実例を挙げて説明します。このクラスを使用したり独自のクラスを作成したりする前に、次のチェックリストを使用してすべてを満たしているかどうかを確認してから、クラスをコンパイルしてください。

- 擬似コードなどを使用する機能の計画。
- データベース・テーブルとカラムのマップの作成。
- Mobile Link システム・テーブルで、Java 同期メソッドの言語タイプとロケーションを指定していることを確認して、統合データベースを設定。

  詳細については、「Java 同期論理の設定」323 ページを参照してください。
- Java クラスの実行中に呼び出される関連 Java クラスのリストの作成。

- Mobile Link 同期サーバのクラスパスに含まれる Java クラスのロケーションの確保。

プラン

この例の Java 同期論理は、例の処理に必要な機能を指定する関連 Java ファイルとクラスを指しています。この例は、クラス CustEmpScripts の作成方法を示します。また、接続用同期コンテキストの設定方法も示します。最後に、次の用途を持つ Java メソッドを提供します。

- Mobile Link ユーザを認証する。
- 各データベース・テーブル用のカーソルを使用して、ダウンロード操作とアップロード操作を実行する。

スキーマ

同期対象となるテーブルは emp と cust です。emp テーブルには、3 つのカラム emp_id、emp_name、manager があります。cust テーブルには、cust_id、cust_name、emp_id という 3 つのカラムがあります。各テーブルのすべてのカラムが同期されます。統合データベースからリモート・データベースへのマッピングにより、テーブル名とカラム名はどちらのデータベースでも同じです。さらに、監査テーブルが統合データベースに追加されます。

Java クラス・ファイル

この例で使用するファイルは、*Samples¥MobiLink¥JavaAuthentication* ディレクトリにあります。

Java 同期スクリプトの作成

設定

次のようにして Java 同期論理を設定します。import 文は、Java 仮想マシンに対して、必要なファイルのロケーションを指定します。public class 文はクラスを宣言します。

```
// 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;
```

CustEmpScripts コンストラクタは、authenticateUser メソッドの準備文をすべて設定します。また、メンバ・データも設定します。

```
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;
        }
    }
```

finalize メソッドは、end_connection が呼び出されなければ JDBC リソースをクリーンアップします。

```
protected void finalize()
      throws SQLException, Throwable
  {
      super.finalize();
      freeJDBCResources();
  }
```

freeJDBCResources メソッドは、割り付けられたメモリを解放し、ここで監査接続を終了します。これは、管理プロシージャです。

```
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;
         }
```

endConnection メソッドは、不要になった時点でリソースをクリーンアップします。これも管理プロシージャです。

```
   public void endConnection()
        throws SQLException
    {
        freeJDBCResources();
     }
```

次の authenticateUser メソッドは、すべてのユーザ・ログインを承認し、データベース・テーブルにユーザ情報のログを取ります。ユーザが ml_user テーブルにない場合は、そのログが login_added に書き込まれます。ユーザ ID が ml_user 内で見つかると、そのログが login_audit に書き込まれます。実際のシステムでは user_password を無視することはありませんが、この例では単純にするためにすべてのユーザを承認しています。実行したデータベース操作のいずれかが失敗して例外が発生すると、プロシージャは SQLException を発行します。

```
   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;
    }
```

次のメソッドは、SQL コードを使用して、データベース・テーブル上でカーソルとして動作します。これらはカーソル・スクリプトであるため、SQL 文字列を返します。

```
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" );
     }
 }
```

このコードは、次のコマンドを使用してコンパイルします。

```
javac -cp %asany9%¥java¥mlscript.jar CustEmpScripts.jar
```

こうすると、クラスパスの CustEmpScripts.class のロケーションを使用して Mobile Link 同期サーバを実行できます。次に、コマンド・ラインの一部を示します。

```
dbmlsrv9 ... -sl java (-cp <class_location>)
```

Mobile Link Java API リファレンス

この項では、Mobile Link Java のインタフェースとクラス、これらに関連するメソッド、コンストラクタについて説明します。

DBConnectionContext インタフェース

現在のデータベース接続に関する情報を取得し、アクセスするためのインタフェース。このインタフェースは、スクリプトを含むクラスのコンストラクタに渡されます。コンテキストがバックグラウンド・スレッドに必要な場合や、接続期間を超えて必要な場合は、ServerContext を使用してください。

警告
DBConnectionContext インスタンスは、Java コードに呼び出すスレッド以外で使用しないでください。

---

getConnection メソッド

public java.sql.Connection **getConnection( )**
throws java.sql.SQLException

既存の接続 java.sql.Connection を JDBC 接続として返します。この接続は、Mobile Link が SQL スクリプトの実行に使用するものと同じです。

この接続は、Mobile Link サーバのこの接続での使用に影響する方法でコミット、クローズ、または変更しないでください。返される接続は、基本となる Mobile Link 接続の期間中にのみ有効です。接続に対して end_connection イベントが呼び出された後は、その接続を使用しないでください。

既存の接続を JDBC 接続としてバインドするときにエラーが発生すると、java.sql.SQLException が発行されます。

フル・アクセス権を持つサーバ接続が必要な場合、ServerContext.makeConnection() を使用してください。

getProperties メソッド

public java.util.Properties **getProperties( )**

この接続のスクリプト・バージョンに基づいて、この接続のプロパティを返します。プロパティは、ml_property テーブルに格納されます。

詳細については、『Mobile Link 管理ガイド』> 「ml_property」および『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

getServerContext メソッド

public ServerContext **getServerContext( )**

この Mobile Link サーバの ServerContext を返します。

getVersion メソッド

public java.util.Properties **getVersion( )**

スクリプト・バージョンの名前を返します。

詳細については、『Mobile Link 管理ガイド』> 「ml_property」および『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

InOutByteArray インタフェース

SQL スクリプトに渡される in/out パラメータの機能を有効にするために、メソッドに渡されます。

getValue メソッド

public byte[ ] **getValue( )**

このバイト配列パラメータの値を返します。

setValue メソッド

public void **setValue(** byte[ ] *new_value* **)**

このバイト配列パラメータの値を設定します。パラメータ *new_value* (このバイト配列が取る値) が 1 つあります。

InOutInteger インタフェース

SQL スクリプトに渡される in/out パラメータの機能を有効にするために、メソッドに渡されます。

getValue メソッド

public int **getValue( )**

この整数パラメータの値を返します。

setValue メソッド

public void **setValue(** int *new_value* **)**

この整数パラメータの値を設定します。パラメータ *new_value* (この整数が取る値) が 1 つあります。

InOutString インタフェース

SQL スクリプトに渡される in/out パラメータの機能を有効にするために、メソッドに渡されます。

getValue メソッド

public java.lang.String **getValue( )**

この文字列パラメータの値を返します。

setValue メソッド

public void **setValue(** int *new_value* **)**

この整数パラメータの値を設定します。パラメータ *new_value* (この文字列が取る値) が 1 つあります。

LogListener インタフェース

ログに出力されるメッセージを取得するためのリスナ・インタフェースです。

messageLogged メソッド

public void **messageLogged(**
ServerContext *sc*
LogMessage *message* **)**

メッセージがログに出力されたときに呼び出されます。パラメータは、メッセージを出力しているサーバのコンテキストである *sc* と、Mobile Link ログに送信された LogMessage である *message* の 2 つがあります。

LogMessage クラス

ログ・メッセージに関連付けられたデータを保持します。

java.lang.Object を拡張します。

定数

int **ERROR**

int **WARNING**

getType メソッド

public int **getType( )**

このメッセージ・タイプのアクセサ。

このメッセージのタイプを返します。タイプは ERROR または WARNING です。

getUser メソッド

public java.lang.String **getUser( )**

このメッセージ・ユーザのアクセサ。メッセージのユーザが存在しない場合、ユーザは NULL です。

このメッセージに対応するユーザを返します。

getText メソッド

public java.lang.String **getText( )**

このメッセージ・テキストのアクセサ。

このメッセージの本文を返します。

ServerContext インタフェース

Mobile Link サーバの継続期間中に存在する、すべてのコンテキストのインスタンス化。このコンテキストを静的なデータとして保持し、バックグラウンド・スレッドで使用できます。Mobile Link によって起動される Java 仮想マシンの継続期間中は有効です。

ServerContext インスタンスにアクセスするには、DBConnectionContext.getServerContext メソッドを使用します。

addShutdownListener

public void **addShutdownListener(** ShutdownListener *sl* **)**

サーバ・コンテキストが破壊される前に通知を受信する指定の ShutdownListener を追加します。シャットダウン時に、メソッド ShutdownListener.shutdownPerformed (ianywhere.ml.script.ServerContext) が呼び出されます。パラメータ *sl* (シャットダウン時に ShutdownListener に通知するように指定) が 1 つあります。

removeShutdownListener

public void **removeShutdownListener(** ShutdownListener *sl* **)**

サーバ・コンテキストが破壊される前に通知を受信するリスナのリストから、指定の ShutdownListener を削除します。パラメータ *sl* (シャットダウン時にリスナに通知されないように指定) が 1 つあります。

shutdown

public void **shutdown( )**

サーバを強制的に停止します。

getStartClassInstances

public java.lang.Object[ ] **getStartClassInstances( )**

サーバ起動時に構築された起動クラスの配列を取得します。起動クラスがない場合、配列の長さは 0 です。

ユーザ定義起動クラスの詳細については、「ユーザ定義起動クラス」333 ページを参照してください。

次に、getStartClassInstances() の例を示します。

```
Object objs[] = sc.getStartClassInstances();
 int I;
 for( I=0; i<objs.length; I+=1 ) {
    if( objs[i] instanceof MyClass ) {
        //use class
    }
 }
```

getProperties

public java.util.Properties **getProperties(**
java.lang.String *component_name*
java.lang.String *prop_set_name* **)**

スクリプト・バージョンに関連する一連のプロパティを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。

詳細については、『Mobile Link 管理ガイド』> 「ml_property」および『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

getPropertiesByVersion

public java.util.Properties **getPropertiesByVersion(** java.lang.String *script_version* **)**

スクリプト・バージョンに関連する一連のプロパティを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。component_name が ScriptVersion の場合、スクリプト・バージョンは prop_set_name カラムに格納されます。

詳細については、『Mobile Link 管理ガイド』> 「ml_property」および『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

getPropertySetNames

public java.util.Properties **getPropertySetNames(** java.lang.String *component_name* **)**

指定したコンポーネントのプロパティ・セット名のリストを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。

詳細については、『Mobile Link 管理ガイド』> 「ml_property」および『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

makeConnection メソッド

public java.sql.Connection **makeConnection( )** throws java.sql.SQLException

新しいサーバ接続を作成します。新しい接続を開くときにエラーが発生すると、このメソッドは java.sql.SQLException を発行します。

addErrorListener メソッド

public void **addErrorListener(** LogListener *ll* **)**

エラーが出力されたときに通知を受信するために、指定された LogListener を追加します。

ll は、通知を受信する LogListener です。

次のメソッドが呼び出されます。
LogListener.messageLogged(ianywhere.ml.script.ServerContext、ianywhere.ml.script.LogMessage)

removeErrorListener メソッド

public void **removeErrorListener(** LogListener *l* **)**

エラーが出力されたときに通知を受信するリスナのリストから、指定した LogListener を削除します。

l は、通知を受信しないようにする LogListener です。

addWarningListener メソッド

public void **addWarningListener(** LogListener *l* **)**

警告が出力されたときに通知を受信するために、指定された LogListener を追加します。

l は、通知を受信する LogListener です。

次のメソッドが呼び出されます。
LogListener.messageLogged(ianywhere.ml.script.ServerContext、ianywhere.ml.script.LogMessage)

removeWarningListener メソッド

public void **removeWarningListener(** LogListener *l* **)**

警告が出力されたときに通知を受信するリスナのリストから、指定した LogListener を削除します。

l は、通知を受信しないようにする LogListener です。

ServerException クラス

サーバで同期の進行を妨げるエラー状態が存在することを示すために発行されます。この例外が発行されると、Mobile Link サーバはシャットダウンされます。

ServerException コンストラクタ

public **ServerException( )**

詳細メッセージのない ServerException を構成します。

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

指定した詳細メッセージを含む ServerException を構成します。パラメータ *s* (詳細メッセージを指定) が 1 つあります。

ShutdownListener インタフェース

サーバのシャットダウンを取得するリスナ・インタフェースです。このインタフェースを使用して、サーバが終了する前に、リソース、スレッド、接続などがすべてクリーンアップされるようにします。

shutdownPerformed メソッド

public void **shutdownPerformed(** ServerContext *sc***)**

サーバのシャットダウンによって ServerContext が破壊される前に起動されます。パラメータ sc (シャットダウンされるサーバのコンテキスト) が 1 つあります。

SynchronizationException クラス

現在の同期の完了を妨げるエラー状態が存在することを示すために発行されます。この例外が発行されると、Mobile Link サーバは強制的にロールバックされます。

SynchronizationException コンストラクタ

public **SynchronizationException( )**

詳細メッセージのない SynchronizationException を構成します。

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

指定した詳細メッセージを含む SynchronizationException を構成します。パラメータ s (詳細メッセージを指定) が 1 つあります。

第 14 章

.NET での同期スクリプトの作成

この章の内容

Mobile Link 同期サーバの動作を制御するには、同期スクリプトを作成します。これらのスクリプトの実装には、SQL、Java または .NET を使用できます。この章では、.NET を使用して同期スクリプトを実装する方法について説明します。

.NET 同期スクリプトを使用するチュートリアルについては、『Mobile Link チュートリアル』> 「チュートリアル : Adaptive Server Anywhere での .NET 同期論理」を参照してください。

スクリプト作成については、「同期スクリプトの作成」285 ページを参照してください。

SQL、Java、.NET の説明と比較については、「同期論理の作成オプション」31 ページを参照してください。

Java でのスクリプト作成については、「Java による同期スクリプトの作成」321 ページを参照してください。

概要

Microsoft .NET は、Web サービスとアプリケーションを構築、配備、実行するためのプラットフォームです。

Mobile Link は、同期スクリプトを作成するためのプログラミング言語として Visual Studio .NET をサポートしています。Mobile Link スクリプトを .NET で作成する場合は、有効な .NET アセンブリを作成できる言語であれば、どれでも使用できます。特に、以下の言語がテスト済みで、文書化されています。

- C#
- Visual Basic .NET
- C++

.NET 同期論理には、Java 同期論理と同じ機能を持たせることができます。Mobile Link 同期サーバは、Mobile Link イベントの発生時に .NET メソッドを呼び出すことが可能です。Mobile Link には SQL 文字列が返されます。

この項では、C#、Visual Basic .NET、C++ 用の .NET 同期論理を設定、開発、実行する方法について説明します。また、サンプル・アプリケーションと Mobile Link .NET API リファレンスも含まれています。

.NET 同期スクリプトを使用するチュートリアルについては、『Mobile Link チュートリアル』> 「チュートリアル：Adaptive Server Anywhere での .NET 同期論理」を参照してください。

.NET 同期論理の設定

同期スクリプトを .NET で実装するときの最も重要な点は、アセンブリに含まれるパッケージ、クラス、メソッドの場所を Mobile Link に指示することです。

❖ .NET を使用して同期スクリプトを実装するには、次の手順に従います。

1. 1 つまたは複数の独自クラスを作成します。必要な同期イベントごとに、メソッドを作成します。これらのメソッドは、パブリックにしてください。

   メソッドの詳細については、「メソッド」363 ページを参照してください。

   静的でないメソッドを持つ各クラスには、パブリック・コンストラクタが必要です。Mobile Link 同期サーバは、各クラスのメソッドが接続のために初めて呼び出されるときに、そのクラスを自動的にインスタンス化します。

   コンストラクタの詳細については、「コンストラクタ」362 ページを参照してください。

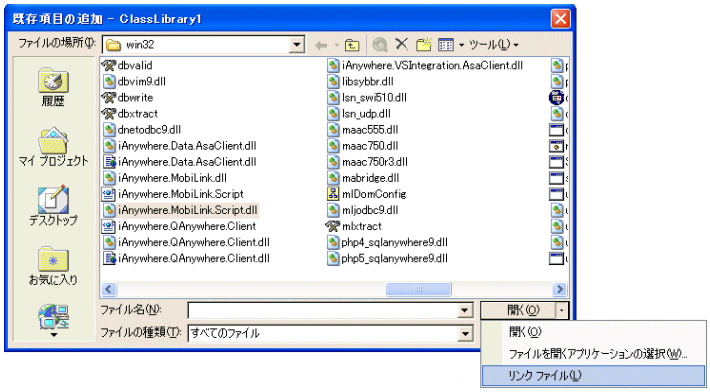
2. 1 つまたは複数のアセンブリを作成します。コンパイル中、独自の .NET メソッドで利用するために、Mobile Link API クラスのリポジトリを含む *iAnywhere.MobiLink.Script.dll* を参照します。*iAnywhere.MobiLink.Script.dll* は、SQL Anywhere インストール環境の *win32* サブディレクトリにあります。

   Mobile Link の .NET API の詳細については、「Mobile Link .NET API リファレンス」381 ページを参照してください。

   コマンド・ラインで、または Visual Studio .NET や他の .NET 開発環境を使用して、クラスをコンパイルできます。

   - Visual Studio .NET からコンパイルするには、次の手順に従います。

1. VS.NET の [プロジェクト] メニューから、[既存項目の追加] を選択します。
2. [開く] ドロップダウン・リストから [リンクファイル] を選択し、*iAnywhere.MobiLink.Script.dll* を選択します。



警告：

Visual Studio .NET の場合は、常に上記の [リンク ファイル] を使用します。*iAnywhere.MobiLink.Script.dll* を参照するために、[参照の追加] オプションを使用しないでください。このオプションは、クラス・アセンブリとして同じ物理ディレクトリに *iAnywhere.MobiLink.Script.dll* を複製するため、Mobile Link 同期サーバにとって問題となります。

---

3. アセンブリを構築するには、[ビルド] メニューを使用します。

- コマンド・ラインからコンパイルするには、次の手順に従います。

次のようにして、コマンド・ラインから *iAnywhere.MobiLink.Script.dll* を参照できます。

```
csc /out:c:¥out.dll /target:library /
reference:c:¥iAnywhere.MobiLink.Script.dll
sync_v1.cs
```

上記の例では、*iAnywhere.MobiLink.Script.dll* が *c:¥* にコピーされたことを想定しています。

API の完全な説明については、「Mobile Link .NET API リファレンス」381 ページを参照してください。

3 統合データベース内の Mobile Link システム・テーブルで、各同期スクリプトについて、呼び出すパッケージ、クラス、メソッドの名前を指定します。スクリプトのバージョンごとに、クラスを 1 つずつ使用します。

この情報を Mobile Link システム・テーブルに追加するには、ml_add_dnet_connection_script ストアド・プロシージャまたは ml_add_dnet_table_script ストアド・プロシージャを使用するのが最も簡単な方法です。

たとえば、次の文は、Adaptive Server Anywhere データベース内で実行すると、authenticate_user 接続レベル・イベントが発生するたびに myNamespace.myClass.myMethod を実行するように指定します。

```
call ml_add_dnet_connection_script( 'version1',
'authenicate_user', 'myNamespace.myClass.myMethod' )
```

注意：
完全に修飾されたメソッド名では、大文字と小文字が区別されます。

このプロシージャ・コールの結果として、ml_script システム・テーブルの script_language カラムに、**dnet** という語が含まれている必要があります。script カラムの文字列には、SQL で実装されるスクリプトの文が含まれています。この文字列には、代わりにパブリックな .NET メソッドの修飾名を入れます。

詳細については、『Mobile Link 管理ガイド』> 「ml_add_dnet_connection_script」および『Mobile Link 管理ガイド』> 「ml_add_dnet_table_script」を参照してください。

この情報を追加するには Sybase Central を使用する方法もあります。

4 アセンブリをロードするよう Mobile Link サーバに指示し、CLR を起動します。dbmlsrv9 コマンド・ラインでオプションを使用して、これらのアセンブリの場所を Mobile Link に指示します。2 つのオプションから選択できます。

- **-sl dnet (-MLAutoLoadPath) を使用** このオプションは、アプリケーションのベース・ディレクトリへの特定のパスを設定し、そのディレクトリ内のすべてのプライベート・アセンブリをロードします。ほとんどの場合、このオプションを使用してください。たとえば、*c:¥* に保存されたすべてのアセンブリをロードするには、次を入力します。

```
dbmlsrv9 -c "dsn=consolidated1" -sl dnet(-
MLAutoLoadPath=c:¥)
```

  -MLAutoLoadPath オプションを使用すると、イベント・スクリプトの完全に修飾されたメソッド名を入力するときに、ドメインを指定できません。

  アセンブリのロードの詳細については、「アセンブリのロード」373 ページを参照してください。dbmlsrv9 オプションの -sl dnet の詳細については、『Mobile Link 管理ガイド』> 「-sl dnet オプション」を参照してください。

- **-sl dnet (-MLDomConfigFile) を使用** このオプションを使用するには、ドメインとアセンブリの設定を含む設定ファイルが必要です。このオプションを使用するのは、共有アセンブリがある場合、ディレクトリのすべてのアセンブリをロードする必要がない場合、またはその他の理由で設定ファイルを使用する必要がある場合です。

共有アセンブリのロードの詳細については、「アセンブリのロード」373 ページを参照してください。dbmlsrv9 オプションの -sl dnet の詳細については、『Mobile Link 管理ガイド』> 「-sl dnet オプション」を参照してください。

注意：

-MLAutoLoadPath オプションまたは -MLDomConfigFile オプションを使用できますが、両方は使用できません。

.NET 同期論理の作成

.NET 同期論理の作成は、他の .NET コードの作成と違いはありません。ユーザが必要なのは、Mobile Link イベントの知識と Mobile Link .NET API の知識です。これ以降の項を参考にして、役立つ同期論理を作成してください。

API の完全な説明については、「Mobile Link .NET API リファレンス」381 ページを参照してください。

.NET 同期論理は、ステータス情報の管理と、アップロード・イベントとダウンロード・イベント関連の論理の実装に使用できます。たとえば、.NET で作成された begin_synchronization スクリプトを使用すると、Mobile Link ユーザ名を変数に格納できます。同期処理中に後で呼び出されるスクリプトは、この変数にアクセスできます。また、.NET は、コミットの実行前または実行後に統合データベースのローにアクセスするために使用できます。

.NET を使用すると、統合データベースへの依存度も減少します。また、統合データベースを新バージョンにアップグレードしたり、別のデータベース管理システムに切り替えたりする場合も、動作に与える影響が少なくて済みます。

クラス・インスタンス

Mobile Link 同期サーバは、クラスを接続レベルでインスタンス化します。静的でない .NET メソッドをあるイベントに対して作成した場合、そのイベントに達すると、現在のデータベース接続でクラスが構成されていなければ、Mobile Link 同期サーバが自動的にクラスを構成します。

詳細については、「コンストラクタ」362 ページを参照してください。

1 つのスクリプト・バージョンの接続レベル・イベントまたはテーブルレベル・イベントに直接関連するすべてのメソッドは、*同じクラスに属している必要があります。*

データベース接続ごとに、インスタンス化されたクラスはその接続が終了するまで持続します。したがって、連続する複数の同期セッションに、引き続き同じインスタンスを使用できます。そのため、パブリック変数またはプライベート変数内の情報は、明示的にクリアしないかぎり、同じ接続で発生するすべての同期を通して持続します。

また、静的なクラスや変数も使用できます。この場合、値は同じドメインのすべての接続を通して使用できます。

統合データベースへの接続が終了した場合にのみ、Mobile Link 同期サーバはクラス・インスタンスを自動的に削除します。

トランザクション

.NET メソッドには、トランザクションに関する通常のルールが適用されます。データベース・トランザクションの開始と継続期間は、同期処理に重要になります。トランザクションの開始と終了は、Mobile Link 同期サーバのみが行います。.NET メソッド内の同期接続でトランザクションを明示的にコミットまたはロールバックすると、同期処理の整合性違反になり、エラーが発生することがあります。

これらのルールは、Mobile Link 同期サーバによって作成されるデータベース接続、特に、メソッドから返される SQL 文にのみ適用されます。クラスで他のデータベース接続を作成する場合は、自由に管理できます。

SQL データ型と .NET データ型

次の表は、Mobile Link スクリプト・パラメータの SQL データ型とそれに対応する .NET データ型を示します。

SQL データ型	.NET データ型
VARCHAR	string
CHAR	string
INTEGER	int
BINARY	byte [ ]

SQL データ型	.NET データ型
TIMESTAMP	DateTime
INOUT INTEGER	ref int
INOUT VARCHAR	ref string
INOUT CHAR	ref string
INOUT BYTEARRAY	ref byte [ ]

コンストラクタ

クラスのコンストラクタは、次の 2 つのシグニチャのどちらかを持ちます。

```
public ExampleClass(
iAnywhere.MobiLink.Script.DBConnectionContext cc )
```

または

```
public ExampleClass()
```

渡される同期コンテキストは、Mobile Link 同期サーバが現在のユーザの同期に使用している接続です。

DBConnectionContext クラスの getConnection メソッドは、Mobile Link が現在のユーザの同期に使用しているのと同じデータベース接続を返します。この接続で文を実行することはできますが、トランザクションのコミットやロールバックは行わないでください。トランザクションは Mobile Link 同期サーバによって管理されます。

Mobile Link 同期サーバは、最初のシグニチャを持つコンストラクタを使用しようとします。void コンストラクタが使用されるのは、最初のシグニチャを持つコンストラクタが存在しない場合のみです。

DBConnectionContext クラスの詳細については、「DBConnectionContext インタフェース」383 ページを参照してください。

メソッド

通常は、同期イベントごとにメソッドを 1 つずつ実装します。これらのメソッドは、パブリックにしてください。プライベート・メソッドの場合、Mobile Link 同期サーバでは使用できず、その存在を認識できません。

統合データベース内の ml_script テーブルで指定されている名前と一致していれば、メソッド名は重要ではありません。ただし、このマニュアルの例では、メソッド名は Mobile Link イベント名と同じです。これは、.NET コードを読みやすくするためです。

メソッドのシグニチャは、そのイベント用スクリプトのシグニチャと一致していなければなりません。ただし、パラメータ・リストの最後にパラメータ値が必要でない場合は、リストをトランケートできます。実際には、パラメータを渡すとオーバヘッドが生じる可能性があるため、必要なパラメータのみを受け入れてください。

ただし、メソッドはオーバロードできません。ml_script システム・テーブルには、クラスごとにメソッド・プロトタイプが 1 つだけ格納されます。

戻り値

Mobile Link アップロードまたはダウンロードに対して呼び出されるメソッドは、有効な SQL 言語の文を返さなければなりません。これらのメソッドの戻り値は String 型にします。他の戻り型は使用できません。

他のすべてのスクリプトの戻り型は、string または void にします。他の型は使用できません。戻り型が null ではなく文字列の場合、Mobile Link 同期サーバはその文字列に有効な SQL 文が含まれているとみなして、この文を通常の SQL 言語による同期スクリプトと同様に統合データベース内で実行します。通常、メソッドは文字列を返しますが、返却時にデータベースに対して SQL 文を実行しない場合は null を返すことができます。

ユーザ定義起動クラス

サーバの起動時に自動的にロードされる起動クラスを定義できます。この機能の目的は、最初の同期の前に Mobile Link サーバが CLR を起動する時点で実行される .NET コードを記述できるようにすることです。つまり、ユーザ同期要求の前に、接続の作成またはデータのキャッシュを実行できます。

この操作を行うには、dbmlsrv9 -sl dnet オプションの MLStartClasses オプションを使用します。たとえば、次に示すのは dbmlsrv9 コマンド・ラインの一部です。mycl1 と mycl2 が起動クラスとしてロードされます。

```
-sl dnet(-MLStartClasses=com.test.mycl1,com.test.mycl2)
```

クラスはリスト内の順序でロードされます。同じクラスがリストに 2 回以上指定されている場合は、複数のインスタンスが作成されます。

すべての起動クラスはパブリックでなければなりません。また、引数を 1 つも受け付けないか、または MobiLink.Script.ServerContext データ型の引数を 1 つ受け付けるパブリック・コンストラクタが必要です。

ロードされた起動クラスの名前は、「.NET 起動クラス *classname* がロードされました。」というメッセージとともに Mobile Link ログに出力されます。

.NET CLR の詳細については、『Mobile Link 管理ガイド』> 「-sl dnet オプション」を参照してください。

サーバ起動時に構築される起動クラスを表示する場合は、「GetStartClassInstances メソッド」392 ページを参照してください。

例

次に示すのは、テンプレート起動クラスです。これは、イベントを処理してデータベース接続を確立するデーモン・スレッドを起動します (すべての起動クラスがスレッドの作成を必要とするわけではありませんが、スレッドを作成する場合はデーモン・スレッドでなければなりません)。

```
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 からの情報の出力

System.Console を使用して、Mobile Link ログに情報を出力する文を .NET メソッドに追加することもできます。これにより、クラスの進行状況と動作を追跡できます。

パフォーマンスのヒント
この方法で情報を出力すると、モニタ・ツールとして活用できますが、運用環境ではおすすめしません。

これと同じ方法を利用して、任意の同期情報のログを取ったり、スクリプトの使用方法に関する統計情報を収集したりできます。

.NET での Mobile Link サーバ・エラーの処理

ログをスキャンするだけでは不十分な場合は、アプリケーションをプログラムからモニタできます。たとえば、特定のタイプのメッセージを電子メールで送信できます。

ログに出力される各エラー・メッセージまたは各警告メッセージを表すクラスに渡されるメソッドを作成することも可能です。この方法は、Mobile Link 同期サーバをモニタおよび監査するのに役立ちます。

次のコードは、すべてのエラー・メッセージ用に Listener をインストールし、その情報を StreamWriter に出力します。

```
class TestLogListener {
     public TestLogListener( StreamWriter output_file  )
      {
          _output_file    = output_file;
      }
      public void errCallback( ServerContext sc,
LogMessage lm )
      {
          string type;
          string user;
          if( lm.Type==LogMessage.MessageType.ERROR ) {
              type = "ERROR";
          } else if(
lm.Type==LogMessage.MessageType.WARNING ) {
              type = "WARNING";
          } else {
              type = "INVALID TYPE!!";
          }
          if( lm.User == null ) {
              user = "null";
          } else {
              user = lm.User;
          }
```

```
        _output_file.WriteLine( "Caught msg type=" +
type +
                                " user=" + user +
                                " text=" + lm.Text );
         _output_file.Flush();
     }
     StreamWriter    _output_file;
 }

// Two lines that registers the TestLogListener Call
this code
 // from anywhere that has access to the ServerContext
such as
 // a class constructor or synchronization script.
 // ServerContext serv_context;
 TestLogListener etll = new
TestLogListener(log_listener_file);
 serv_context.ErrorListener += new
LogCallback(etll.errCallback);
```

参照

「LogCallback デリゲート」391 ページ

「ServerContext インタフェース」392 ページの ErrorListener と WarningListener

「LogMessage クラス」391 ページ

「MessageType 列挙」392 ページ

.NET 同期論理のデバッグ

次の手順では、Microsoft Visual Studio .NET を使用して .NET スクリプトをデバッグする方法を説明します。

❖ .NET スクリプトをデバッグするには、次の手順に従います。

1 次のようにして、デバッグ情報をオンに設定してコードをコンパイルします。

- csc コマンド・ラインで、フラグ **/debug+** を設定します。

または

- Microsoft Visual Studio .NET の設定を使用してデバッグ出力を設定します。
  - [ファイル] メニューから [ビルド] ー [構成マネージャ] を選択します。[アクティブ ソリューション構成] ドロップダウン・リストから、[デバッグ] を選択します。
  - アセンブリを構築します。

2 ソース・ファイルを含む Visual Studio .NET の実行中のインスタンスを閉じます。

次の手順では、新しい Visual Studio .NET インスタンスを起動して、Mobile Link 同期サーバと使用している .NET 同期スクリプトをデバッグします。

3 コマンド・ライン・オプションを使用して Visual Studio .NET を起動し、Mobile Link 同期サーバをデバッグします。

- コマンド・プロンプトで、Visual Studio .NET インストール環境の *Common7¥IDE* サブディレクトリに移動します。
- /debugexe オプションを使用して、devenv (Visual Studio .NET IDE) を起動します。

  たとえば、次のコマンド・ラインを入力して、Mobile Link 同期サーバをデバッグします。接続文字列と .NET アセンブリをロードするオプションを含めて、dbmlsrv9 オプションを指定してください。

  ```
  devenv /debugexe %asany9%¥win32¥dbmlsrv9.exe -c
  ...
  ```

  この結果、Visual Studio .NET が起動し、[ソリューション エクスプローラ] ウィンドウに dbmlsrv9.exe が表示されます。

4 .NET コードをデバッグするために Microsoft Visual Studio を次のように設定します。

- Visual Studio の [ソリューション エクスプローラ] ウィンドウで dbmlsrv9.exe を右クリックし、[プロパティ] を選択します。
- **[デバッガのタイプ]** を **[自動]** から **[混合]** または **[マネージのみ]** に変更します。

  これによって、Visual Studio .NET は、.NET 同期スクリプトのみをデバッグします。

5 関連する .NET ソース・ファイルを開き、ブレーク・ポイントを設定します。

注意 : dbmlsrv9 ソリューションでソース・ファイルを個別に開きます。元のソリューションやプロジェクト・ファイルは開かないでください。

6 [デバッグ] メニューまたは [F5] キーを使用して Mobile Link を起動します。

プロンプトが表示されたら、*dbmlsrv9.sln* を適切なロケーションに保存します。

[シンボル情報なし] ダイアログが表示された場合は、[OK] をクリックしてデバッグを続けます。デバッグしているのは、Mobile Link が呼び出す管理対象の .NET 同期スクリプトであり、Mobile Link 同期サーバ本体ではありません。

7 同期を実行します。この結果、ブレークポイントのあるコードが Mobile Link によって実行されます。

.NET の同期の方法

この項では、一般的な .NET の同期タスクに取り組む場合に使用できる方法について説明します。

ローのアップロード

upload_insert、upload_update、upload_delete の各イベントを使用して、統合データベースにアップロードされたローを処理することができます。

これらのイベントの詳細については、以下の項目を参照してください。

- 「upload_insert テーブル・イベント」572 ページ
- 「upload_update テーブル・イベント」587 ページ
- 「upload_delete テーブル・イベント」568 ページ

これらのイベントに渡される可変のロー数は、対応する .NET 関数にパラメータとして渡されることはありません。テンポラリ・テーブルにこれらのローを挿入し、後続のイベントで結果セットを処理するという、特別な処理を行います。

❖ 統合データベースにアップロードされたローを処理するには、次の手順に従います。

1. アップロードされた値をテンポラリ・テーブルに挿入します。

   たとえば、upload_insert イベントに対して次の関数を使用します。

```
public static string UploadInsert(/* no parameters
*/)
  {
    return("INSERT INTO tempULCustomer(cust_id,
cust_name) values (?,?)");
  }
```

2 後続のスクリプトを使用してテンポラリ・テーブルから値を取得し、ローを処理します。

end_upload イベントに対して次の関数を登録します。

```
public void EndUpload(string user, string table)
 {
    // declare variables to store your row values
    string _custid;
    string _custname;

    // create the command
    DBCommand stmt = currConn.CreateCommand();
    stmt.CommandText = "select * from
tempULCustomer";

   // store the result set in a DBRowReader instance
    DBRowReader rset = stmt.ExecuteReader();

    // traverse the result set
    object[] rvalues = rset.NextRow();

    while( rvalues != null )
    {
       _custid = (string)rvalues[0];
       _custname = (string)rvalues[1];

      //... your logic to process the rows

       rvalues = rset.NextRow();
    }

    rset.Close();
    stmt.Close();
 }
```

共有アセンブリのロード

この項では、.NET アセンブリをロードするためのオプションと共有アセンブリをロードする処理について説明します。

アセンブリのロード

.NET アセンブリは、タイプ、メタデータ、プログラム・コードのパッケージです。.NET アプリケーションでは、すべてのコードがアセンブリになければなりません。アセンブリ・ファイルの拡張子は *.dll* または *.exe* です。

アセンブリには、2 種類あります。

- **プライベート・アセンブリ**　ファイル・システム内のファイル。
- **共有アセンブリ**　グローバル・アセンブリ・キャッシュにインストールされるアセンブリ。

Mobile Link では、クラスを含むアセンブリが指定されないと、クラスをロードしてそのクラスのメソッドを呼び出すことができません。指定する必要があるのは、Mobile Link が直接呼び出すアセンブリのみです。このアセンブリによって、その他の必要なアセンブリが呼び出されます。

たとえば、Mobile Link が MyAssembly を呼び出すと、MyAssembly が UtilityAssembly と NetworkingUtilsAssembly を呼び出します。この場合は、Mobile Link が MyAssembly だけを探すように設定します。

Mobile Link では、次の 2 つの方法でアセンブリをロードできます。

- **-sl dnet (-MLAutoLoadPath) を使用**　このオプションは、プライベート・アセンブリに対してのみ有効です。このオプションは、アプリケーションのベース・ディレクトリへのパスを設定し、そのディレクトリ内のすべてのアセンブリをロードします。このオプションの方が簡単に使用でき、ほとんどの場合はこのオプションで十分対応できます。

-MLAutoLoadPath オプションを使用すると、イベント・スクリプトの完全に修飾されたメソッド名を入力するときに、ドメインを指定できません。

-MLAutoLoadPath でパスとディレクトリを指定すると、次のアクションが実行されます。

- このパスがアプリケーション・ベース・パスとして設定される
- 指定したディレクトリで *.dll* または *.exe* の付くすべてのファイルのすべてのクラスがロードされる
- 1 つのアプリケーション・ドメインが作成され、ドメインが指定されていないすべてのユーザ・クラスがそのドメインにロードされる

このオプションでは、グローバル・アセンブリ・キャッシュ内のアセンブリを直接呼び出すことはできません。これらの共有アセンブリを呼び出すには、-MLDomConfigFile を使用します。

- **-sl dnet (-MLDomConfigFile) を使用**　このオプションは、プライベート・アセンブリと共有アセンブリの両方に使用できます。このオプションを使用するには、ドメインとアセンブリの設定を含む設定ファイルが必要です。このオプションは、共有アセンブリがある場合、アプリケーション・ベース・パス内のすべてのアセンブリをロードする必要がない場合、またはその他の理由で設定ファイルを使用する必要がある場合に使用してください。

  このオプションを使用すると、Mobile Link は指定されたドメイン設定ファイル内の設定を読み込みます。ドメイン設定ファイルには、1 つまたは複数の .NET ドメインの設定が入っています。このファイルに複数のドメインが記述されている場合は、1 番目に指定されているドメインがデフォルト・ドメインとして使用されます (デフォルト・ドメインは、指定されたドメインがスクリプトにない場合に使用されます)。

  Mobile Link はアセンブリをロードするときに、まずプライベート・アセンブリとしてロードし、次にグローバル・アセンブリ・キャッシュからアセンブリをロードしようとします。プライ

ベート・アセンブリは、アプリケーション・ベース・ディレクトリにあります。共有アセンブリはグローバル・アセンブリ・キャッシュからロードされます。

-MLDomConfigFile オプションでは、ドメイン設定ファイルで指定されているアセンブリのみが、イベント・スクリプトから直接呼び出せます。

サンプルのドメイン設定ファイル

mlDomConfig.xml というサンプルのドメイン設定ファイルが、Mobile Link とともにインストールされます。独自のファイルを最初から作成するか、要件に合わせてサンプルを編集できます。サンプル・ファイルは、SQL Anywhere Studio の次のパスにあります。

MobiLink¥setup¥dnet¥mlDomConfig.xml

サンプルのドメイン設定ファイル *mlDomConfig.xml* の内容は次のとおりです。

```
<?xml version="1.0" encoding="utf-8"?>
 <config xmlns="iAnywhere.MobiLink.mlDomConfig"
xsi:schemaLocation='iAnywhere.MobiLink.mlDomConfig
mlDomConfig.xsd' xmlns:xsi='http://www.w3.org/2001/
XMLSchema-instance' >
    <domain>
       <name>SampleDomain1</name>
       <appBase>C:¥scriptsDir</appBase>
       <configFile></configFile>
       <assembly name="Assembly1" />
       <assembly name="Assembly2" />
    </domain>
    <domain>
       <name>SampleDomain2</name>
       <appBase>¥Dom2assembly</appBase>

<configFile>¥Dom2assembly¥AssemblyRedirects.config</
configFile>
       <assembly name="Assembly3" />
       <assembly name="Assembly4" />
    </domain>
 </config>
```

次に、*mlDomConfig.xml* の内容について説明します。

- **name** イベント・スクリプトでドメインを指定するときに使用するドメイン名です。
  `"DomainName:Namespace.Class.Method"` というフォーマットのイベント・スクリプトを使用するには、ドメイン設定ファイル内に DomainName という名前のドメインが必要です。

  ドメイン名は 1 つ以上指定してください。

- **appBase** ドメインがアプリケーション・ベース・ディレクトリとして使用するディレクトリです。すべてのプライベート・アセンブリは、このディレクトリに基づいて .NET CLR によってロードされます。appBase は必ず指定してください。

- **configFile** ドメインに使用する .NET アプリケーション設定ファイルです。ここは空白でもかまいません。通常は、この部分を使用してアセンブリのバインドとロードのデフォルトの動作を変更します。アプリケーション設定ファイルの詳細については、.NET のマニュアルを参照してください。

- **assembly** イベント・スクリプト内の型の参照を解決するときに Mobile Link がロードして検索するアセンブリの名前です。アセンブリは 1 つ以上指定してください。アセンブリが複数のドメインで使用されている場合は、ドメインごとに 1 つのアセンブリを指定してください。プライベート・アセンブリの場合は、ドメインのアプリケーション・ベース・ディレクトリになければなりません。

dbmlsrv9 オプションの -sl dnet の詳細については、『Mobile Link 管理ガイド』> 「-sl dnet オプション」を参照してください。

.NET 同期のサンプル

このサンプルは、.NET 同期論理を使用して authenticate_user イベントを処理する方法を表示するように既存のアプリケーションを変更します。このサンプルは、*AuthUser.cs* という名前の authenticate_user 用の C# スクリプトを作成します。このスクリプトは、user_pwd_table というテーブル内でユーザのパスワードを検索し、そのパスワードに基づいてユーザを認証します。

注意：
この項では、基本の .NET 同期論理がわかる簡単な例を提供します。一般に、カスタム・ユーザ認証メカニズムを使用する理由として、既存の DBMS ユーザ認証スキームと統合したり、最小パスワード長やパスワード期限などのカスタム機能を提供することが挙げられます。Mobile Link にも組み込みのデフォルト・メカニズムが用意されています。

Mobile Link 認証の詳細については、『Mobile Link クライアント』> 「ユーザ認証メカニズムの選択」を参照してください。

まず、テーブル user_pwd_table をデータベースに追加します。Interactive SQL で次のコードを実行します。

```
CREATE TABLE user_pwd_table (
   user_name  varchar(128) PRIMARY KEY NOT NULL,
   pwd        varchar(128)
 )
```

次に、ユーザとパスワードをテーブルに追加します。

```
INSERT INTO user_pwd_table VALUES( 'user1', 'myPwd' )
```

.NET アセンブリ用のディレクトリを作成します。次に例を示します。

```
mkdir c:¥mlexample
```

次の内容の *AuthUser.cs* というファイルを作成します。

詳細については、『Mobile Link 管理ガイド』> 「authenticate_user 接続イベント」を参照してください。

```
using System;
 using iAnywhere.MobiLink.Script;

namespace MLExample
 {

   public class AuthClass
   {
      private DBConnection  _conn;



      /// AuthClass constructor.

   public AuthClass( DBConnectionContext cc )
   {
      _conn   = cc.GetConnection();
   }


   /// The DoAuthenticate method handles the
'authenticate_user'
   /// event.
   /// Note: This method does not handle password
changes for
   /// advanced authorization status codes.



  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 parameter for the user name
            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;
        }
        }
    }
```

MLExample.AuthClass.DoAuthenticate メソッドは、authenticate_user イベントを処理します。これはユーザ名とパスワードを受け入れ、検証の成功または失敗を示す認証ステータス・コードを返します。

ファイル *AuthUser.cs* をコンパイルします。コンパイルは、コマンド・ライン上または Visual Studio .NET で実行できます。

たとえば、次のコマンド・ラインは、*AuthUser.cs* をコンパイルして、*example.dll* という名前のアセンブリを *c:¥mlexample* に生成します。*asany9* を自分のインストール・ディレクトリに置き換えます。

```
csc /out:c:¥mlexample¥example.dll /target:library /
reference:¥asany9¥win32¥iAnywhere.MobiLink.Script.dll
AuthUser.cs
```

authenticate_user イベント用の .NET コードを登録します。実行する必要があるメソッドは、ネームスペース MLExample とクラス AuthClass にあります。次の SQL を実行します。

```
call ml_add_dnet_connection_script( 'ex_version',
'authenticate_user',
'MLExample.AuthClass.DoAuthenticate' )
 COMMIT
```

次に、次のオプションで Mobile Link 同期サーバを実行します。このオプションによって、Mobile Link が *c:¥myexample* 内のすべてのアセンブリをロードします。

```
-sl dnet ( -MLAutoLoadPath=c:¥mlexample )
```

これで、ユーザがバージョン ex_version と同期するときに、テーブル user_pwd_table のパスワードで認証されるようになります。

Mobile Link .NET API リファレンス

この項では、Mobile Link .NET のインタフェースとクラス、これらに関連するメソッド、プロパティ、コンストラクタについて説明します。これらのクラスを使用するには、SQL Anywhere Studio のインストール・ディレクトリにあるアセンブリ \win32\iAnywhere.MobiLink.Script.dll を参照します。

この項では C# について説明しますが、VB.NET と C++ にも同様の説明が適用されます。

DBCommand インタフェース

public interface **DBCommand**
Member of **iAnywhere.MobiLink.Script**

SQL 文またはデータベース・コマンドを表します。DBCommand は更新またはクエリを表すことができます。

たとえば、次の C# コードは DBCommand インタフェースを使用して次の 2 つのクエリを実行します。

```
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();
```

次の C# サンプルは DBCommand でパラメータを指定して更新を実行します。

```
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 メソッド

public void **Prepare( )**

CommandText に格納されている SQL 文の実行を準備します。

ExecuteNonQuery() メソッド

public int **ExecuteNonQuery( )**

クエリ以外の文を実行します。データベース内で SQL 文の影響を受けるローの数を返します。

ExecuteReader() メソッド

public DBRowReader **ExecuteReader( )**

結果セットを返すクエリ文を実行します。SQL 文が返す結果を取得するための DBRowReader を返します。

Close() メソッド

public void **Close( )**

現在の SQL 文またはコマンドを終了します。

CommandText プロパティ

public string **CommandText**

値は実行する SQL 文です。

DBParameterCollection Parameters プロパティ

public **DBParameterCollection Parameters**

この DBCommand に対する iAnywhere.MobiLink.Script.DBParameterCollection を取得します。

DBConnection インタフェース

public interface **DBConnection**
Member of **iAnywhere.MobiLink.Script**

Mobile Link ODBC 接続を表します。

このインタフェースにより、ユーザが作成した同期論理で Mobile Link によって確立された ODBC 接続にアクセスできます。

Commit() メソッド

public void **Commit( )**

現在のトランザクションをコミットします。

Rollback() メソッド

public void **Rollback( )**

現在のトランザクションをロールバックします。

Close() メソッド

public void **Close( )**

現在の接続を閉じます。

CreateCommand() メソッド

public DBCommand **CreateCommand( )**

この接続で SQL 文またはコマンドを作成します。新しく生成された DBCommand を返します。

DBConnectionContext インタフェース

public interface **DBConnectionContext**
Member of **iAnywhere.MobiLink.Script**

現在のデータベース接続に関する情報を取得し、アクセスするためのインタフェース。このインタフェースは、スクリプトを含むクラスのコンストラクタに渡されます。コンテキストがバックグラウンド・スレッドに必要な場合や、接続期間を超えて必要な場合は、ServerContext を使用してください。

警告：
DBConnectionContext インスタンスは、.NET コードに呼び出すスレッド以外で使用しないでください。

GetConnection メソッド

public **iAnywhere.MobiLink.Script.DBConnection GetConnection( )**
Member of **iAnywhere.MobiLink.Script.DBConnectionContext**

既存の接続を返します。この接続は、Mobile Link が SQL スクリプトの実行に使用するものと同じです。

この接続は、Mobile Link サーバのこの接続での使用に影響する方法でコミット、終了、または変更しないでください。返される接続は、基本となる Mobile Link 接続の期間中にのみ有効です。接続に対して end_connection イベントが呼び出された後は、その接続を使用しないでください。

フル・アクセス権を持つサーバ接続が必要な場合、ServerContext.makeConnection() を使用してください。

GetServerContext メソッド

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

この Mobile Link サーバの ServerContext を返します。

getProperties メソッド

NameValueCollection **getProperties( )**

この接続のスクリプト・バージョンに基づいて、この接続のプロパティを返します。プロパティは、ml_property テーブルに格納されます。

詳細については、『Mobile Link 管理ガイド』> 「ml_property」および『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

getVersion メソッド

string **getVersion( )**

スクリプト・バージョンの名前を返します。

詳細については、『Mobile Link 管理ガイド』> 「ml_property」および『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

DBParameter クラス

public class **DBParameter**
Member of **iAnywhere.MobiLink.Script**

ODBC バウンド・パラメータを表します。

パラメータを指定してコマンドを実行するには、DBParameter が必要です。すべてのパラメータを指定してからコマンドを実行してください。

たとえば、次の C# コードは DBCommand でパラメータを指定して更新を実行します。

```
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 プロパティ

public **SQLTYPE dbType**

値はこのパラメータの SQLType です。

デフォルト：SQLType.SQL_TYPE_NULL

Direction プロパティ

public **System.Data.ParameterDirection Direction**

値はこのパラメータの入出力方向です。

デフォルト：ParameterDirection.Input

IsNullable プロパティ

public bool **IsNullable**

値は、このパラメータが NULL かどうかを示します。

デフォルト：false

ParameterName プロパティ

public string **ParameterName**

値はこのパラメータの名前です。

デフォルト：null

Precision プロパティ

public uint **Precision**

値はこのパラメータの 10 進数精度です。SQLType.SQL_NUMERIC パラメータと SQLType.SQL_DECIMAL パラメータにのみ使用します。

デフォルト：0

Scale プロパティ

public short **Scale**

値はこのパラメータの解決可能な桁数です。SQLType.SQL_NUMERIC パラメータと SQLType.SQL_DECIMAL パラメータにのみ使用します。

デフォルト：0

Size プロパティ

public uint **Size**

値はこのパラメータのサイズ (バイト数) です。

デフォルト：DbType から推定されます。

Value プロパティ

public object **Value**

値はこのパラメータの値です。

デフォルト：null

DBParameterCollection クラス

public class **DBParameterCollection**
inherits from **IDataParameterCollection**, **IList**, **ICollection**, **IEnumerable**
Member of **iAnywhere.MobiLink.Script**

DBParameters のコレクション。DBCommand で DBParamterCollection を作成した時点では、DBParamterCollection は空です。DBCommand を実行する前に、適切なパラメータを指定してください。

DBParameterCollection( ) メソッド

public **DBParameterCollection( )**

DBParameter の空リストを作成します。

Contains(string parameterName) メソッド

public bool **Contains(string** *parameterName* **)**

指定した名前のパラメータがコレクションに含まれている場合は、true を返します。パラメータ *parameterName* (パラメータの名前) を 1 つ取ります。

IndexOf(string parameterName) メソッド

public int **IndexOf(string** *parameterName* **)**

パラメータのインデックスを返します。指定した名前のパラメータがない場合は、-1 を返します。パラメータ *parameterName* (パラメータの名前) を 1 つ取ります。

RemoveAt(string parameterName) メソッド

public void **RemoveAt(string** *parameterName* **)**

指定した名前のパラメータをコレクションから削除します。パラメータ *parameterName* (パラメータの名前) を 1 つ取ります。

Add(object value) メソッド

public int **Add(object** *value* **)**

指定したパラメータをコレクションに追加します。パラメータ *value* (コレクションに追加する iAnywhere.MobiLink.Script.DBParameter) を 1 つ取ります。コレクションに追加されたパラメータのインデックスを返します。

Clear() メソッド

public void **Clear( )**

すべてのパラメータをコレクションから削除します。

Contains(object value) メソッド

public bool **Contains(object** *value* **)**

指定した iAnywhere.MobiLink.Script.DBParameter がこのコレクションに含まれている場合は、true を返します。パラメータ *value* (iAnywhere.MobiLink.Script.DBParameter) を 1 つ取ります。

IndexOf(object value) メソッド

public int **IndexOf(object** *value* **)**

このコレクションに含まれる指定した iAnywhere.MobiLink.Script.DBParameter のインデックスを返します。パラメータ *value* (iAnywhere.MobiLink.Script.DBParameter) を 1 つ取ります。

Insert(int index, object value) メソッド

public void **Insert(int** *index*, **object** *value* **)**

指定した iAnywhere.MobiLink.Script.DBParameter を、コレクションの指定したインデックス位置に挿入します。2 つのパラメータ *value* (iAnywhere.MobiLink.Script.DBParameter) と *index* (挿入先のインデックス) を取ります。

Remove(object value) メソッド

public void **Remove(object** *value* **)**

指定した iAnywhere.MobiLink.Script.DBParameter をコレクションから削除します。パラメータ *value* (iAnywhere.MobiLink.Script.DBParameter) を 1 つ取ります。

RemoveAt(int index) メソッド

public int **RemoveAt(int index)**

このコレクションから指定したインデックス位置の iAnywhere.MobiLink.Script.DBParameter を削除します。パラメータ *index* (iAnywhere.MobiLink.Script.DBParameter のインデックス) を 1 つ取ります。

CopyTo(Array array, int index) メソッド

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

コレクションの内容を、指定したインデックスで始まる特定の配列にコピーします。2 つのパラメータ *array* (コレクションの内容のコピー先の配列) と *index* (コレクションの内容のコピーを開始する配列内のインデックス) を取ります。

GetEnumerator() メソッド

public IEnumerator **GetEnumerator( )**

コレクションの列挙子を返します。

IsFixedSize プロパティ

public bool **IsFixedSize**

false を返します。

IsReadOnly プロパティ

public bool **IsReadOnly**

false を返します。

Count プロパティ

public int **Count**

コレクション内のパラメータ数。

IsSynchronized プロパティ

public bool **IsSynchronized**

false を返します。

SyncRoot プロパティ

public object **SyncRoot**

コレクションの同期に使用できるオブジェクト。

this[string parameterName] プロパティ

public object **this[string** *parameterName* **]**

コレクション内で指定した名前の iAnywhere.MobiLink.Script.DBParameter を取得または設定します。パラメータ *parameterName* (取得または設定する iAnywhere.MobiLink.Script.DBParameter の名前) を 1 つ取ります。

this[int index] プロパティ

public object **this[int** *index* **]**

コレクション内で指定したインデックスの iAnywhere.MobiLink.Script.DBParameter を取得または設定します。パラメータ *index* (取得または設定する iAnywhere.MobiLink.Script.DBParameter のインデックス) を 1 つ取ります。

DBRowReader インタフェース

public interface **DBRowReader**
Member of **iAnywhere.MobiLink.Script**

データベースから読み込まれるロー・セットを表します。メソッド DBCommand.executeReader( ) を実行すると、DBRowReader が作成されます。

次のサンプルは C# コード・フラグメントです。このコードは、結果セット内で所定の DBRowReader で表されるローの関数を呼び出します。

```
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( ) メソッド

public object[ ] **NextRow( )**

結果セット内の次のローの値を取得して返します。結果セットにローがなくなると、NULL を返します。

「SQLType 列挙」396 ページを参照してください。

Close( ) メソッド

public void **Close( )**

この MLDBRowReader で使用されるリソースをクリーンアップします。Close( ) を呼び出した後は、この MLDBRowReader を再び使用することはできません。

ColumnNames プロパティ

public string[ ] **ColumnNames**

結果セットに含まれるすべてのカラムの名前を取得します。値は、結果セット内のカラム名に相当する文字列の配列です。

ColumnTypes プロパティ

public SQLType[ ] **ColumnTypes**

結果セットに含まれるすべてのカラムの型を取得します。値は、結果セット内のカラムの型に対応する SQLTypes の配列です。

LogCallback デリゲート

public delegate void **LogCallback(**
ServerContext *sc*
LogMessage *message*
)
Member of **iAnywhere.MobiLink.Script**

Mobile Link 同期サーバがメッセージを出力したときに呼び出されます。

LogMessage クラス

public class **LogMessage** : **iAnywhere.MobiLink.Script.LogMessage**
Member of **iAnywhere.MobiLink.Script**

ログに出力されたメッセージに関する情報が含まれています。

Type プロパティ

public LogMessage.MessageType **Type**

このインスタンスが表すログ・メッセージのタイプ。

User プロパティ

public string **User**

このメッセージがログに記録されるユーザ。null の場合があります。

Text プロパティ

public string **Text**

メッセージの本文。

MessageType 列挙

public enum **MessageType**
Member of **iAnywhere.MobiLink.Script.LogMessage**

LogMessage の使用可能な型の列挙。

ERROR フィールド

public **ERROR**

ログのエラー・メッセージ。

WARNING フィールド

public **WARNING**

ログの警告メッセージ。

ServerContext インタフェース

public interface **ServerContext**
Member of **iAnywhere.MobiLink.Script**

Mobile Link サーバの継続期間中に存在する、すべてのコンテキストのインスタンス化。このコンテキストを静的なデータとして保持し、バックグラウンド・スレッドで使用できます。Mobile Link で起動される .NET CLR の継続期間中は有効です。

ServerContext インスタンスにアクセスするには、DBConnectionContext.getServerContext メソッドを使用します。

GetStartClassInstances メソッド

public **object[ ] GetStartClassInstances( )**
Member of **iAnywhere.MobiLink.Script.ServerContext**

サーバ起動時に構築された起動クラスの配列を取得します。起動クラスがない場合、配列の長さは 0 です。

ユーザ定義起動クラスの詳細については、「ユーザ定義起動クラス」364 ページを参照してください。

次に、getStartClassInstances() の例を示します。

```
void FindStartClass( ServerContext sc, string name )
  {
    object[] startClasses =
sc.GetStartClassInstances();

   foreach( object obj in startClasses ) {
      if( obj is MyClass ) {
          // Execute some code.....
      }
    }
  }
```

LogCallback ErrorListener イベント

このイベントは、Mobile Link 同期サーバがエラーを出力したときにトリガされます。

LogCallback WarningListener イベント

このイベントは、Mobile Link 同期サーバが警告を出力したときにトリガされます。

makeConnection メソッド

public **iAnywhere.MobiLink.Script.DBConnection makeConnection( )**
Member of **iAnywhere.MobiLink.Script.ServerContext**

新しいサーバ接続を作成します。

ShutDown メソッド

public void **Shutdown( )**
Member of **iAnywhere.MobiLink.Script.ServerContext**

サーバを強制的に停止します。

ShutdownListener メソッド

public event iAnywhere.MobiLink.Script.ShutdownCallback
ShutdownListener(
iAnwyhere.MobiLink.Script.ServerContext *sc***)**
Member of **iAnywhere.MobiLink.Script.ServerContext**

このイベントはシャットダウン時にトリガされます。次のサンプル・コードは、このイベントの使用方法のサンプルです。

```
ShutdownCallback callback = new ShutdownCallback(
shutdownHandler );
 _sc.ShutdownListener += callback;

public void shutdownHandler( ServerContext sc )
 //==============================================
 {
 _test_out_file.WriteLine( "shutdownPerformed" );
 }
```

getProperties メソッド

NameValueCollection **getProperties(**
string *component_name*
string *prop_set_name* **)**

スクリプト・バージョンに関連する一連のプロパティを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。

詳細については、『Mobile Link 管理ガイド』> 「ml_property」および『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

getPropertiesByVersion メソッド

NameValueCollection **getPropertiesByVersion(** string *script_version* **)**

スクリプト・バージョンに関連する一連のプロパティを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。component_name が ScriptVersion の場合、スクリプト・バージョンは prop_set_name カラムに格納されます。

詳細については、『Mobile Link 管理ガイド』> 「ml_property」および『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

getPropertySetNames メソッド

StringCollection **getPropertySetNames(** string *component_name* **)**

指定したコンポーネントのプロパティ・セット名のリストを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。

詳細については、『Mobile Link 管理ガイド』> 「ml_property」および『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

ServerException クラス

public class **ServerException**
Member of **iAnywhere.MobiLink.Script**

サーバでエラーが発生したため直ちにシャットダウンする必要があることを Mobile Link に通知します。

ServerException コンストラクタ

public **ServerException( )**
Member of **iAnywhere.MobiLink.Script.ServerException**

詳細メッセージのない ServerException を構成します。

public **ServerException(string** *message* **)**
Member of **iAnywhere.MobiLink.Script.ServerException**

指定されたメッセージを持つ ServerException を新規に作成します。パラメータ *message* はこの ServerException のメッセージです。

public **ServerException(string** *message*, **SystemException** *ie* **)**
Member of **iAnywhere.MobiLink.Script.ServerException**

指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む ServerException を新規に作成します。これには 2 つのパラメータがあります。*message* はこの ServerException のメッセージで、*ie* はこの ServerException の原因となった例外です。

ShutdownCallback デリゲート

public sealed delegate **ShutdownCallback : System.MulticastDelegate**
Member of **iAnywhere.MobiLink.Script**

Mobile Link 同期サーバがシャットダウンするときに呼び出されます。Mobile Link サーバのシャットダウン時に呼び出される ServerContext.ShutdownListener イベントで、このデリゲートの実装を登録できます。

SQLType 列挙

public enum **SQLType**
Member of **iAnywhere.MobiLink.Script**

使用可能なすべての ODBC データ型の列挙。

SQL_TYPE_NULL フィールド

public **SQL_TYPE_NULL**

Null データ型。

SQL_UNKNOWN_TYPE フィールド

public **SQL_UNKNOWN_TYPE**

不定のデータ型。

SQL_CHAR フィールド

public **SQL_CHAR**

設定されたサイズの UTF-8 文字配列。.NET 型 String を持ちます。

SQL_NUMERIC フィールド

public **SQL_NUMERIC**

設定されたサイズと精度の数値。.NET 型 decimal を持ちます。

SQL_DECIMAL フィールド

public **SQL_DECIMAL**

設定されたサイズと精度の 10 進数字。.NET 型 decimal を持ちます。

SQL_INTEGER フィールド

public **SQL_INTEGER**

32 ビット整数値。.NET 型 Int32 を持ちます。

SQL_SMALLINT フィールド

public **SQL_SMALLINT**

16 ビット整数値。.NET 型 Int16 を持ちます。

SQL_FLOAT フィールド

public **SQL_FLOAT**

ODBC ドライバで精度が定義された浮動小数点数。.NET 型 Double を持ちます。

SQL_REAL フィールド

public **SQL_REAL**

単精度の浮動小数点数。.NET 型 Single を持ちます。

SQL_DOUBLE フィールド

public **SQL_DOUBLE**

倍精度浮動小数点数。.NET 型 Double を持ちます。

SQL_DATE フィールド

public **SQL_DATE**

日付。.NET 型 DateTime を持ちます。

SQL_DATETIME フィールド

public **SQL_DATETIME**

日付と時刻。.NET 型 DateTime を持ちます。

SQL_TIME フィールド

public **SQL_TIME**

時刻。.NET 型 DateTime を持ちます。

SQL_INTERVAL フィールド

public **SQL_INTERVAL**

時間の間隔。.NET 型 TimeSpan を持ちます。

SQL_TIMESTAMP フィールド

public **SQL_TIMESTAMP**

タイムスタンプ。.NET 型 DateTime を持ちます。

SQL_VARCHAR フィールド

public **SQL_VARCHAR**

ユーザ設定の最大長を持ち、null で終了する UTF-8 文字列。.NET 型 String を持ちます。

SQL_TYPE_DATE フィールド

public **SQL_TYPE_DATE**

日付。.NET 型 DateTime を持ちます。

SQL_TYPE_TIME フィールド

public **SQL_TYPE_TIME**

時刻。.NET 型 DateTime を持ちます。

SQL_TYPE_TIMESTAMP フィールド

public **SQL_TYPE_TIMESTAMP**

タイムスタンプ。.NET 型 DateTime を持ちます。

SQL_DEFAULT フィールド

public **SQL_DEFAULT**

デフォルト型。型を持ちません。

SQL_ARD_TYPE フィールド

public **SQL_ARD_TYPE**

ARD オブジェクト。型を持ちません。

SQL_BIT フィールド

public **SQL_BIT**

シングル・ビット。.NET 型 Boolean を持ちます。

SQL_TINYINT フィールド

public **SQL_TINYINT**

8 ビット整数。.NET SByte 型を持ちます。

SQL_BIGINT フィールド

public **SQL_BIGINT**

64 ビット整数。.NET 型 Int64 を持ちます。

SQL_LONGVARBINARY フィールド

public **SQL_LONGVARBINARY**

ドライバ依存の最大長を持つ可変長バイナリ・データ。.NET 型 byte[ ] を持ちます。

SQL_VARBINARY フィールド

public **SQL_VARBINARY**

ユーザ指定した最大長を持つ可変長バイナリ・データ。.NET 型 byte[ ] を持ちます。

SQL_BINARY フィールド

public **SQL_BINARY**

固定長のバイナリ・データ。.NET 型 byte[ ] を持ちます。

SQL_LONGVARCHAR フィールド

public **SQL_LONGVARCHAR**

ドライバ依存の最大長を持ち、null で終了する UTF-8 文字列。.NET 型 String を持ちます。

SQL_GUID フィールド

public **SQL_GUID**

グローバル・ユニーク ID (UUID とも呼びます)。.NET 型 Guid を持ちます。

SQL_WCHAR フィールド

public **SQL_WCHAR**

固定サイズのユニコード文字配列。.NET 型 String を持ちます。

SQL_WVARCHAR フィールド

public **SQL_WVARCHAR**

ユーザ定義の最大長を持ち、null で終了するユニコード文字列。.NET 型 String を持ちます。

SQL_WLONGVARCHAR フィールド

public **SQL_WLONGVARCHAR**

ドライバ依存の最大長を持ち、null で終了するユニコード文字列。.NET 型 String を持ちます。

SynchronizationException クラス

public class **SynchronizationException**
Member of **iAnywhere.MobiLink.Script**

同期例外が発生したことと、現在の同期をロールバックして再開する必要があることを通知します。

SynchronizationException コンストラクタ

public **SynchronizationException( )**
Member of **iAnywhere.MobiLink.Script.SynchronizationException**

詳細のない SynchronizationException を構成します。

public **SynchronizationException(string** *message* **)**
Member of **iAnywhere.MobiLink.Script.SynchronizationException**

指定されたメッセージを持つ SynchronizationException を新規に作成します。パラメータ *message* はこの ServerException のメッセージです。

public **SynchronizationException(string** *message*, **SystemException** *ie* **)**
Member of **iAnywhere.MobiLink.Script.SynchronizationException**

指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む SynchronizationException を新規に作成します。これには 2 つのパラメータがあります。*message* はこの ServerException のメッセージで、*ie* はこの ServerException の原因となった例外です。

第 15 章

同期イベント

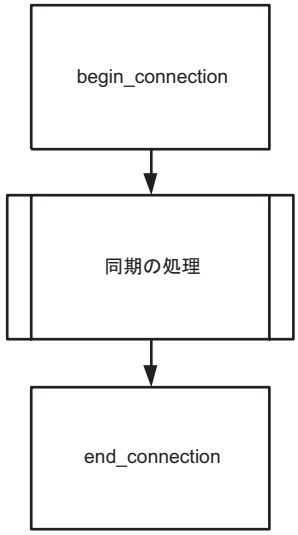
この章の内容

この章では、Mobile Link 同期イベントと、これらのイベントを処理する SQL スクリプト、Java メソッド、または .NET メソッドについて説明します。スクリプトを実装すると、これらのイベントのうち 1 つまたは複数を処理して、Mobile Link 同期サーバの動作を制御できます。

スクリプトの保存については、『Mobile Link 管理ガイド』> 「統合データベースでのスクリプトの追加と削除」を参照してください。

Mobile Link イベントの概要

同期要求が発生し、Mobile Link サーバが、新しい接続を作成することを決定すると、begin_connection イベントが呼び出され、同期が始まります。



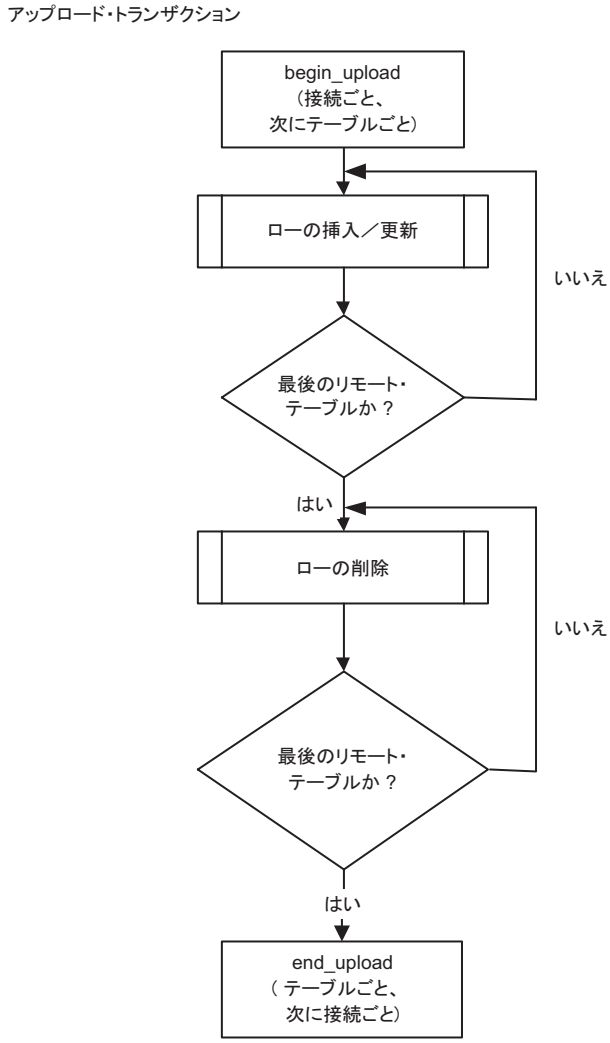
同期に続いて、接続が接続プールに配置され、再度 Mobile Link は現在のスクリプト・バージョンに対する同期要求を待ちます。end_connection イベントが呼び出された後、接続は最終的に接続プールから削除されます。ただし、同じバージョンに対する同期要求を再び受信した場合、Mobile Link はその次に受けた同期要求を同じ接続で処理します。現在の同期に影響するイベントがいくつかあります。

同期の主要段階は、アップロード・トランザクションとダウンロード・トランザクションです。アップロードとダウンロードのトランザクションに含まれるイベントについては、以下で概説します。

アップロード・トランザクション

アップロード・トランザクションは、リモート・データベースからアップロードされた変更を適用します。

begin_upload イベントは、アップロード・トランザクションの開始にマークを付けます。アップロード・トランザクション処理は 2 段階になっています。まず、すべてのリモート・テーブルに対する挿入と更新がアップロードされ、次にすべてのリモート・テーブルに対する削除がアップロードされます。



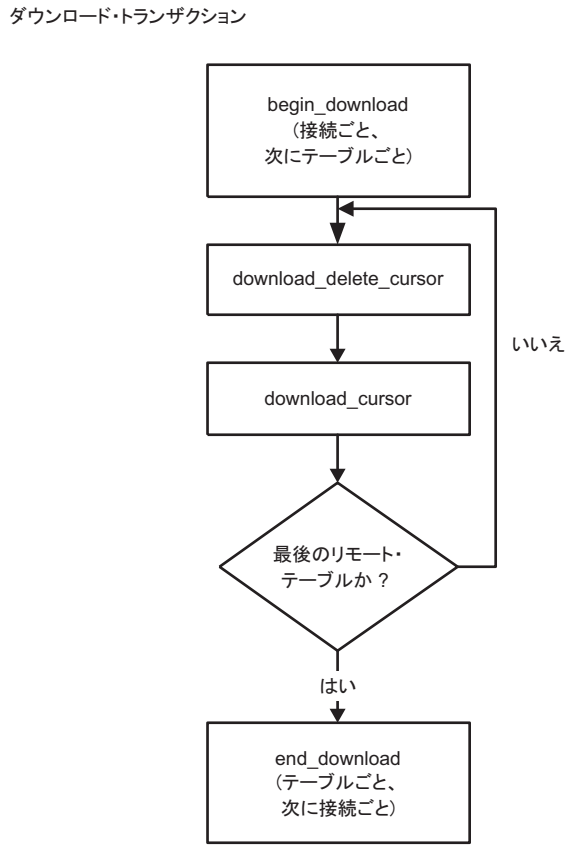
end_upload イベントは、アップロード・トランザクションの終了にマークを付けます。

アップロード中に発生するイベントの詳細については、『Mobile Link 管理ガイド』> 「ローをアップロードするスクリプトの作成」を参照してください。

ダウンロード・トランザクション

ダウンロード・トランザクションは、統合データベースからローをフェッチします。ダウンロード・トランザクションは、begin_download イベントで始まります。

ダウンロード・トランザクション処理は 2 段階になっています。まず最初に各テーブルに対する削除がダウンロードされ、次に更新／挿入ロー (アップサート) がダウンロードされます。end_download イベントによって、ダウンロード・トランザクションが終了します。



ダウンロード中に発生するイベントの詳細については、『Mobile Link 管理ガイド』> 「ローをダウンロードするスクリプトの作成」を参照してください。

次の疑似コードは、イベント (イベントと同名のスクリプト) が呼び出される順序の概要を示します。

擬似コード内のイベントの概要

次の擬似コードは、完全な Mobile Link 同期イベント・モデルを示しています。このモデルは、エラーのない完全な同期 (アップロード専用またはダウンロード専用ではない) を想定しています。

説明

- ほとんどの場合、指定したイベントのスクリプトを定義していないと、デフォルト・アクションは何も実行しません。
- begin_connection イベントと end_connection イベントは、「**接続レベルのイベント**」です。これらのイベントは特定の同期に依存せず、パラメータがありません。
- 各テーブルが同期されるたびに 1 回ずつ呼び出されるイベントもあります。これらのイベントに対応するスクリプトは、「**テーブル・レベルのスクリプト**」と呼ばれます。

  各テーブルは専用のテーブル・スクリプトを持つことができますが、いくつかのテーブルで共有されるテーブル・レベルのスクリプトを作成することもできます。
- begin_synchronization など、一部のイベントは接続レベルとテーブル・レベルの両方で発生します。これらのイベントに対しては、接続スクリプトとテーブル・スクリプトの両方を作成することができます。
- 同期処理がどのように複数のトランザクションに分散されるかについては、COMMIT 文が参考になります。
- データベース・エラーは、同期処理中のどの時点でも発生する可能性があります。データベース・エラーは、handle_error スクリプトまたは handle_odbc_error スクリプトを使用して処理されます。

警告

同期スクリプト、または同期スクリプトから呼び出されるプロシージャやトリガで、暗黙的または明示的なコミットまたはロールバックを実行しないでください。スクリプト内に COMMIT 文または ROLLBACK 文があると、同期手順のトランザクションの性質が変化してしまいます。これらの文を使用すると、障害が発生した場合にデータの整合性を保証できません。

```
-----------------------------------------------------
 Synchronization events in pseudocode.

 Legend:
 - // This is a comment
```

```
 - <name>
     The pseudo code for <name> is listed separately
     in a later section, under a banner:
         ------------------------
         name
         ------------------------
 - VariableName <- value
     Assign the given value to the given variable name.
     Variable names are in mixed case.
 - event_name
     If you have defined a script for the given event
name,
     it will be invoked.
 -------------------------------------------------------



CONNECT to consolidated database
 begin_connection_autocommit
 begin_connection
 COMMIT
 for each synchronization request with
      the same script version {
   <synchronize>
 }
 end_connection
 COMMIT
 DISCONNECT from consolidated database



-------------------------------------------------------
 synchronize
 -------------------------------------------------------

 <authenticate>
 <begin_synchronization>
 <upload>
 <prepare_for_download>
 <download>
 <end_synchronization>
```

```
-----------------------------------------------------
 authenticate
 -----------------------------------------------------

 Status <- 1000
 UseDefaultAuthentication <- TRUE
 if( authenticate_user script is defined ) {
   UseDefaultAuthentication <- FALSE
   TempStatus <- authenticate_user
   if( TempStatus > Status ) {
     Status <- TempStatus
   }
 }


if( authenticate_user_hashed script 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 <= 2000 ) {
   if( authenticate_parameters script is defined )
  {
     TempStatus <- authenticate_parameters
     if( TempStatus > Status ) {
```

```
        Status <- TempStatus
    }
  }
  if( Status >= 3000 ) {
    ROLLBACK
    // Abort the synchronization.
  } else {
    // UserName defaults to MobiLink user name
    // sent from the remote.
    if( modify_user script is defined ) {
    UserName <- modify_user
    // The new value of UserName is later passed to
    // all scripts that expect the MobiLink user name.
    }
    COMMIT
  }
```

```
-------------------------------------------------------
 begin_synchronization
 -------------------------------------------------------

 begin_synchronization   // conection event
 for each table being synchronized {
     begin_synchronization    // call the table level
script
 }
 for each publication being synchronized {
   begin_publication
 }
 COMMIT
```

```
-------------------------------------------------------
 end_synchronization
 -------------------------------------------------------

 for each publication being synchronized {
   if( begin_publication script was called ) {
     end_publication
   }
 }
 for each table being synchronized {
   if( begin_synchronization table script was called ) {
```

```
    end_synchronization // table event
  }
}
end_synchronization     // connection event

for each table being synchronized {
synchronization_statistics // table event
}
synchronization_statistics // connection event
for each table being synchronized {
time_statistics // table event
}
time_statistics // connection event

COMMIT
```

アップロード・ストリームの処理の詳細については、「アップロード中のイベント」410 ページを参照してください。

ダウンロード・ストリームの処理の詳細については、「ダウンロード中のイベント」414 ページを参照してください。

アップロード中のイベント

次の擬似コードは、アップロード・イベントとアップロード・スクリプトがどのように呼び出されるかを示します。

これらのイベントは、完全なイベント・モデルのアップロードのロケーションで発生します。詳細については、「Mobile Link イベントの概要」402 ページを参照してください。

```
-----------------------------------------------------
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
    }
    end_upload

    for each table being synchronized {
      upload_statistics  // table event
    }
      upload_statistics  // connection event

    COMMIT



-------------------------------------------------------
 <upload_inserted_row>
 -------------------------------------------------------
 // NOTES:
 // - Only table scripts for the current table are
involved.
 // - Cursor-based upload scripts, like upload_cursor,
 //   are deprecated, and so are not shown.

 UploadUsingStatements <- (
        upload_insert script is defined
      or upload_update script is defined
      or upload_delete script is defined
      or upload_fetch script is defined
      or upload_new_row_insert script is defined
      or upload_old_row_insert script is defined )


 if( UploadUsingStatements ) {
   ConflictsAreExpected <- (
        upload_new_row_insert script is defined
      or upload_old_row_insert script is defined
      or resolve_conflict script is defined )
```

```
   if( upload_insert script is defined ) {
     upload_insert
   } else if( ConflictsAreExpected
       and upload_update script is not defined
       and upload_insert script is not defined
       and upload_delete script is not defined ) {
       // Forced conflict.
       upload_new_row_insert
       resolve_conflict
   } else {
       // ignore the insert
   }
 } else {
   // ignore the insert
 }



-------------------------------------------------------
 upload_updated_row
 -------------------------------------------------------
 // NOTES:
 // - Only table scripts for the current table are
involved.
 // - Both the old (original) and new rows are uploaded
for
 //   each update.
 // - Cursor-based upload scripts, like upload_cursor,
 //   are deprecated, and so are not shown.

 UploadUsingStatements <- (
        upload_insert script is defined
     or upload_update script is defined
     or upload_delete script is defined
     or upload_fetch script is defined
     or upload_new_row_insert script is defined
     or upload_old_row_insert script is defined )


if( UploadUsingStatements ) {
   ConflictsAreExpected <- (
        upload_new_row_insert script is defined
     or upload_old_row_insert script is defined
     or resolve_conflict script is defined )
   Conflicted <- FALSE
   if( upload_update script is defined ) {
```

```
      if( ConflictsAreExpected
        and upload_fetch script 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 script is not defined
        and upload_insert script is not defined
        and upload_delete script is not defined ) {
        // Forced conflict.
        Conflicted <- TRUE
    }
    if( ConflictsAreExpected and Conflicted ) {
      upload_old_row_insert
      upload_new_row_insert
      resolve_conflict
    }
  } else {
    // ignore the update
  }



-------------------------------------------------------
 upload_deleted_row
 -------------------------------------------------------
 // NOTES:
 // - Only table scripts for the current table are
involved.
 // - Cursor-based upload scripts, like upload_cursor,
 //   are deprecated, and so are not shown.

 UploadUsingStatements <- (
        upload_insert script is defined
      or upload_update script is defined
      or upload_delete script is defined
      or upload_fetch script is defined
      or upload_new_row_insert script is defined
      or upload_old_row_insert script is defined )
```

```
if( UploadUsingStatements ) {
   ConflictsAreExpected <- (
        upload_new_row_insert script is defined
     or upload_old_row_insert script is defined
     or resolve_conflict script is defined )
   if( upload_delete is defined ) {
     upload_delete
   } else if( ConflictsAreExpected
     and upload_update script is not defined
     and upload_insert script is not defined
     and upload_delete script is not defined ) {
     // Forced conflict.
     upload_old_row_insert
     resolve_conflict
   } else {
     // ignore this delete
   }
 } else {
     // ignore this delete
 }
```

ダウンロード中のイベント

次の疑似コードは、ダウンロード・イベント (イベントと同名のスクリプト) が呼び出される順序の概要を示します。

これらのイベントは、「Mobile Link イベントの概要」402 ページに示す完全なイベント・モデルのダウンロードのロケーションで発生します。

```
-----------------------------------------------------
 prepare_for_download
 -----------------------------------------------------

prepare_for_download
modify_last_download_timestamp
if( prepare_for_download script is defined or
modify_last_download_timestamp script is defined ) {
     COMMIT
 }
```

```
-----------------------------------------------------
download
-----------------------------------------------------

begin_download
for each table being synchronized {
  begin_download_deletes
  for each row in download_delete_cursor {
    if( all primary key columns are NULL ) {
      send TRUNCATE to remote
    } else {
      send DELETE to remote
    }
  }
  end_download_deletes
  begin_download_rows
  for each row in download_cursor {
    send INSERT ON EXISTING UPDATE to remote
  }
  end_download_rows
}
modify_next_download_timestamp
end_download



for each table being synchronized {
  download_statistics   // table event
}
  download_statistics   // connection event

COMMIT
```

説明

- ダウンロード通知を要求していてもダウンロードの確認をクライアントから受け取らなかった場合、統合データベースではダウンロード・トランザクション全体がロールバックされます。

  詳細については、Adaptive Server Anywhere リモートの場合は『Mobile Link クライアント』> 「SendDownloadACK (sa) 拡張オプション」を、Ultra Light リモートの場合は『Mobile Link クライアント』> 「Send Download Acknowledgement 同期パラメータ」を参照してください。

- ダウンロード・ストリームでは、挿入と更新が区別されません。download_cursor イベントに対応するスクリプトは、ダウンロードされるローを定義する SELECT 文です。クライアントは、ローが存在するかどうかを調べ、適切な挿入または更新の操作を実行します。
- ダウンロード処理の最後に、クライアントは自動的に参照整合性に違反するローを削除します。

  詳細については、『Mobile Link 管理ガイド』> 「参照整合性と同期」を参照してください。

authenticate_parameters 接続イベント

機能

ユーザ ID やパスワード以外の認証に使用できる、リモートからの非テーブル・データを受信します。非テーブル・データを使用して、各同期を任意にカスタマイズすることもできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

項目	パラメータ	説明
1	auth_status	INTEGER。これは INOUT パラメータです。
2	ml_username	VARCHAR(128)
3...	remote_parameters (1 つ以上)	VARCHAR(128)

説明

Adaptive Server Anywhere のリモートからも Ultra Light のリモートからも、文字列またはパラメータを文字列の形式で送信できます。これにより、ユーザ ID とパスワード以外の認証も可能になります。また、パラメータの値に基づいて同期をカスタマイズでき、それも前同期フェーズで認証中にカスタマイズできます。

Ultra Light リモート・データベースでは、ul_synch_info 構造体の num_auth_parms フィールドと auth_parms フィールドを使用して、パラメータを渡します。num_auth_parms は、パラメータの数で、0 ～ 255 の値になります。auth_parms は、文字列のリストへのポインタです。文字列がプレーン・テキストとして表示されるのを防ぐため、文字列はパスワードと同じ方法で送信されます。num_auth_parms が 0 の場合、auth_parms は NULL に設定します。

次に、Ultra Light でパラメータを渡す例を示します。

```
ul_char * Params[ 3 ] = { UL_TEXT( "parm1" ), UL_TEXT(
"parm2" ), UL_TEXT( "parm3" ) };

 ...
 info.num_auth_parms = 3;
 info.auth_parms = Params;
```

Adaptive Server Anywhere のリモート・データベースでは、カンマで区切られたリストにあるパラメータを dbmlsync -ap オプションを使用して渡します。次に例を示します。

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

Mobile Link 同期サーバは、各同期の開始時にこのイベントを実行します。このイベントは、begin_synchronization と同じトランザクション内で、begin_synchronization イベントの直前に実行されます。

このイベントを使用して、組み込み Mobile Link 認証メカニズムを、カスタム・メカニズムに置き換えられます。使用している DBMS の認証メカニズムを使用したり、Mobile Link 組み込みメカニズムには存在しない機能を実装したりできます。

リモート・パラメータの数が予期される数と一致しないと、エラーの原因となります。たとえば、3 つのパラメータが送信された場合、イベントはパラメータが 5 つあると予測する必要があります。これは、3 つのパラメータに加えて、auth_status と ml_username があるためです。また、パラメータがクライアントから送信されたときにイベントがない場合にもエラーが発生します。

authenticate_user スクリプトまたは authenticate_user_hashed スクリプトが呼び出されてエラーを返すと、このイベントは呼び出されません。

authenticate_parameters イベント用の SQL スクリプトは、ストアド・プロシージャとして実装してください。

参照

- 『Mobile Link クライアント』> 「Mobile Link ユーザの認証」
- 『Mobile Link クライアント』> 「カスタム・ユーザ認証」
- 「authenticate_user 接続イベント」420 ページ
- 「authenticate_user_hashed 接続イベント」425 ページ
- 「begin_synchronization 接続イベント」447 ページ
- 『Mobile Link クライアント』> 「-ap オプション」

- 『Mobile Link クライアント』> 「Authentication Parameters 同期パラメータ」

authenticate_user 接続イベント

機能

カスタム・ユーザ認証を実装します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	auth_status	INTEGER。これは INOUT パラメータです。
2	ml_username	VARCHAR(128)
3	user_password	VARCHAR(128)
4	user_new_password	VARCHAR(128)

デフォルトのアクション

Mobile Link 組み込みユーザ認証メカニズムを使用します。

説明

Mobile Link 同期サーバは、各同期の開始時にこのイベントを実行します。このイベントは、begin_synchronization トランザクションの前にトランザクション内で実行されます。

このイベントを使用して、組み込み Mobile Link 認証メカニズムを、カスタム・メカニズムに置き換えられます。使用している DBMS の認証メカニズムを使用したり、Mobile Link 組み込みメカニズムには存在しない機能 (パスワードの有効期限やパスワードの最小長など) を実装したりできます。

authenticate_user イベントで使用するパラメータは、次のとおりです。

1. **auth_status**　この必須パラメータは、スクリプトに値を提供し、スクリプトから新しい値を受け取ることがある INOUT パラメータです。auth_status パラメータは認証の全体の成功状況を示します。次のいずれかの値に設定されます。

戻り値	Auth_status	説明
V <= 1999	1000	認証に成功しました。
1999 < V <= 2999	2000	認証に成功しました。パスワードの有効期限がもうすぐ切れます。
2999 < V <= 3999	3000	認証に失敗しました。パスワードの有効期限が切れています。
3999 < V <= 4999	4000	認証に失敗しました。
4999 < V <= 5999	5000	認証に失敗しました。ユーザがすでに同期中です。
5999 < V	4000	戻り値が 5999 より大きい場合、その値は 4000 として解釈されます。

2. **ml_username**　このオプションのパラメータは、Mobile Link ユーザ名です。

3. **user_password**　認証に使用するパスワードを示すオプションのパラメータです。ユーザがパスワードを入力しない場合は、NULL です。

4. **user_new_password**　新しいパスワードを示すオプションのパラメータです。ユーザがパスワードを変更しない場合は、NULL です。

authenticate_user イベント用の SQL スクリプトは、ストアド・プロシージャとして実装してください。

2 つの認証スクリプトを両方とも定義し、両方のスクリプトが異なる auth_status コードを返す場合は、大きい方の値が使用されます。

authenticate_user スクリプトは、すべての認証スクリプトと共に、トランザクション内で実行されます。このトランザクションは、常にコミットを実行します。

LDAP サーバ、IMAP サーバ、POP3 サーバを使用する認証を簡素化するために、authenticate_user イベントに使用できる事前に定義されたスクリプトがあります。

詳細については、『Mobile Link クライアント』> 「外部サーバに対する認証」を参照してください。

参照

- 『Mobile Link クライアント』> 「Mobile Link ユーザの認証」
- 『Mobile Link クライアント』> 「カスタム・ユーザ認証」
- 『Mobile Link クライアント』> 「外部サーバに対する認証」
- 「authenticate_user_hashed 接続イベント」425 ページ
- 「authenticate_parameters 接続イベント」417 ページ
- 「modify_user 接続イベント」533 ページ
- 「begin_synchronization 接続イベント」447 ページ

SQL の例

一般的な authenticate_user スクリプトは、ストアド・プロシージャの呼び出しです。呼び出しの中のパラメータ順は、上記の順序と同じでなければなりません。次の例では、ml_add_connection_script を使用して、my_auth というストアド・プロシージャにイベントを割り当てます。

```
call ml_add_connection_script(
    'ver1', 'authenticate_user', 'call my_auth ( ?, ?,
?, ? )'
 )
```

たとえば、次の Adaptive Server Anywhere ストアド・プロシージャは認証にユーザ名のみ使用し、パスワードのチェックは行いません。このプロシージャは、指定されたユーザ名が ULEmployee テーブルにある従業員 ID の 1 つであるかどうかだけを確認します。

```
CREATE PROCEDURE my_auth(in @user_name varchar(128))
 begin
   if exists
   ( select * from ulemployee
     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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、authenticateUser という Java メソッドを authenticate_user イベント用のスクリプトとして登録します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
call ml_add_java_connection_script(
    'ver1', 'authenticate_user',
    'ExamplePackage.ExampleClass.authenticateUser'
 )
```

次に示すのは、サンプルの Java メソッド authenticateUser です。このメソッドは、ユーザのパスワードをチェックし、必要に応じてパスワードを変更する Java 関数を呼び出します。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、AuthUser という .NET メソッドを authenticate_user 接続イベント用のスクリプトとして登録します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
call ml_add_dnet_connection_script(
    'ver1', 'authenticate_user',
    'TestScripts.Test.AuthUser'
 )
```

次に示すのは、AuthUser メソッド用の C# シグニチャです。

```
public void AuthUser( ref int authStatus, string user,
string pwd, string newPwd )
```

.NET において C# で作成された authenticate_user スクリプトの詳しい例については、『Mobile Link 管理ガイド』> 「.NET 同期のサンプル」を参照してください。

authenticate_user_hashed 接続イベント

機能

カスタム・ユーザ認証メカニズムを実装します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	auth_status	INTEGER。これは INOUT パラメータです。
2	ml_username	VARCHAR(128)
3	hashed_user_password	BINARY(20)。ユーザがパスワードを入力しない場合は、NULL です。
4	hashed_new_password	BINARY(20)。ユーザがパスワードを変更しない場合は、NULL です。

デフォルトのアクション

Mobile Link 組み込みユーザ認証メカニズムを使用します。

説明

このイベントは authenticate_user と同じですが、パスワードの部分のみ異なります。パスワードは、ml_user.hashed_password カラムに格納されたものと同じように、ハッシュされた形式となります。パスワードをハッシュされた形式で渡すことにより、セキュリティが向上します。

一方向ハッシュが使用されます。一方向ハッシュはパスワードを使用し、それを各パスワードで (基本的に) ユニークなバイト・シーケンスに変換します。一方向ハッシュでは、統合データベースに実際のパスワードを保存せずにパスワードの認証を行えます。

authenticate_user と authenticate_user_hashed を両方とも定義し、両方のスクリプトが異なる auth_status コードを返す場合は、大きい方の値が使用されます。

参照

- 『Mobile Link クライアント』> 「Mobile Link ユーザの認証」
- 『Mobile Link クライアント』> 「カスタム・ユーザ認証」
- 「authenticate_user 接続イベント」420 ページ
- 「authenticate_parameters 接続イベント」417 ページ

SQL の例

一般的な authenticate_user_hashed スクリプトは、ストアド・プロシージャの呼び出しです。呼び出しの中のパラメータ順は、上記の順序と同じでなければなりません。次の例では、ml_add_connection_script を呼び出して、my_auth というストアド・プロシージャにイベントを割り当てます。

```
call ml_add_connection_script(
    'ver1', 'authenticate_user_hashed', 'call my_auth (
?, ?, ?)'
 )
```

次の Adaptive Server Anywhere のストアド・プロシージャは、認証にユーザ名とパスワードの両方を使用します。このプロシージャは、指定されたユーザ名が ULEmployee テーブルにある従業員 ID の 1 つであるかどうかだけを確認します。プロシージャは、Employee テーブルには hashed_pwd という名前の binary(20) のカラムがあることを前提としています。

```
CREATE PROCEDURE my_auth(
   inout @auth_status integer,
   in @user_name varchar(128),
   in @hpwd binary(20) )
 begin
   if exists
   ( select * from ulemployee
     where emp_id = @user_name
       and hashed_pwd = @hpwd )
   then
     message 'OK' type info to client;
     return 1000;
```

```
    else
      message 'Not OK' type info to client;
      return 4000;
    end if
  end
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、authUserHashed という Java メソッドを authenticate_user_hashed イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
    'ver1', 'authenticate_user_hashed',
    'ExamplePackage.ExampleClass.authUserHashed')
```

次に示すのは、サンプルの Java メソッド authUserHashed です。このメソッドは、ユーザのパスワードをチェックし、必要に応じてパスワードを変更する Java 関数を呼び出します。

```
public String authUserHashed(
    ianywhere.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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、AuthUserHashed という .NET メソッドを authenticate_user_hashed 接続イベント用のスクリプトとして登録します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
call ml_add_dnet_connection_script(
    'ver1',
    'authenticate_user_hashed',
    'TestScripts.Test.AuthUserHashed'
 )
```

次に示すのは、AuthUserHashed 呼び出し用の C# シグニチャです。

```
public void AuthUserHashed(
    ref int authStatus,
    string user,
    byte[] pwd,
    byte[] newPwd )
```

begin_connection 接続イベント

機能

Mobile Link 同期サーバが統合データベース・サーバに接続するときに呼び出されます。

パラメータ

なし。

デフォルトのアクション

なし。

説明

Mobile Link 同期サーバは、統合データベース・サーバへの各ワーカ・スレッド接続を開くときに、このイベントを実行します。Mobile Link 同期は、同期要求を受け取ると、要求に応じて接続を開きます。アプリケーションが Mobile Link 同期サーバへの接続を形成または再形成すると、Mobile Link 同期サーバはその同期の間、データベース・サーバへの接続を一時的に 1 つ割り付けます。

参照

- 「end_connection 接続イベント」480 ページ.

SQL の例

次の SQL スクリプトは、Adaptive Server Anywhere 統合データベースで動作します。これで 2 つの変数が作成されます。1 つは last_download タイムスタンプ、もう 1 つは従業員 ID の変数です。

```
call ml_add_connection_script(
     'custdb',
     'begin_connection',
     'create variable @LastDownload timestamp;
     create variable @EmployeeID integer;')
```

Java の例

*注意：*このスクリプトは、通常は Java では使用されません。これは、データベース変数の代わりにこのクラス・インスタンスのメンバ変数を使用し、メンバをコンストラクタで準備するためです。

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginConnection という Java メソッドを begin_connection イベント用のスクリプトとして登録します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
call ml_add_java_connection_script(
     'ver1',
     'begin_connection',
     'ExamplePackage.ExampleClass.beginConnection' )
```

次に示すのは、サンプルの Java メソッド beginConnection です。このメソッドは、接続レベルの変数を作成する SQL を返します。

```
public String beginConnection()
 { return("create variable @LastDownload timestamp;" );
}
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、BeginConnection という .NET メソッドを begin_connection 接続イベント用のスクリプトとして登録します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
call ml_add_dnet_connection_script(
    'ver1',
    'begin_connection',
    'TestScripts.Test.BeginConnection'
 )
```

次に示すのは、BeginConnection 呼び出し用のシグニチャです。

```
public string BeginConnection()
 { return("create variable @LastDownload timestamp;" );
}
```

begin_connection_autocommit 接続イベント

機能

オートコミットをオンにします。

パラメータ

なし。

デフォルトのアクション

オートコミットはオフです。

説明

Mobile Link 同期サーバが統合データベースに接続するときにエラーが発生した場合、アップロード・ストリームとダウンロード・ストリームをロールバックできるように、オートコミットをオフにします。

しかし、Adaptive Server Enterprise の統合データベースを使用している場合、オートコミットがオンでなければ、テンポラリ・テーブルを作成するなどの DDL 機能は実行できません。Adaptive Server Enterprise の統合データベースを使用している場合、begin_connection_autocommit イベントで DDL コマンドを実行します。イベントが終了すると、オートコミットがオフになります。

begin_connection_autocommit スクリプトは、繰り返し可能なように作成します。エラーまたはデッドロックが発生した場合、Mobile Link 同期サーバがスクリプトをリトライする必要があるからです (スクリプトのロールバックはできません)。

begin_download 接続イベント

機能

Mobile Link 同期サーバがダウンロード・データ・ストリームの準備を開始する直前に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download	TIMESTAMP
2	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

説明

Mobile Link 同期サーバは、ダウンロード情報を処理する最初のステップとしてこのイベントを実行します。ダウンロード情報は 1 つのトランザクションで処理されます。このイベントは、このトランザクションで最初に実行されます。

last_download タイムスタンプは、ダウンロード・フェーズの直前、最後に成功した同期の間に統合データベースから取得された値です。現在のユーザが同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

参照

- 「end_download 接続イベント」482 ページ

SQL の例

次の例では、ml_add_connection_script を呼び出して、SetDownloadParameters というストアド・プロシージャにイベントを割り当てます。

```
call ml_add_connection_script (
    'Lab',
    'begin_download',
    'CALL SetDownloadParameters( ?, ? )' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginDownloadConnection という Java メソッドを begin_download 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
    'example_ver',
    'begin_download',

'ExamplePackage.ExampleClass.beginDownloadConnection' )
```

次に示すのは、サンプルの Java メソッド beginDownloadConnection です。このメソッドは、以前に設定された JDBC 同期を使用して削除テーブルを準備する Java 関数 (prepDeleteTables) を呼び出します。

```
public String beginDownloadConnection(
     Timestamp ts, String user )
   throws java.sql.SQLException
 {  prepDeleteTables ( _syncConn, ts, user );
    return ( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、BeginDownload という .NET メソッドを begin_download 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script(
   'ver1',
   'begin_download',
   'TestScripts.Test.BeginDownload'
 )
```

次に示すのは、サンプルの .NET メソッド BeginDownload です。このメソッドは、以前に設定された JDBC 同期を使用して削除テーブルを準備する .NET 関数 (prepDeleteTables) を呼び出します。

```
public void BeginDownload(
   DateTime timestamp,
   string user )
 {  prepDeleteTables ( _syncConn, ts, user );
    return ( null ); }
```

begin_download テーブル・イベント

機能

挿入、更新、削除のダウンロード・ストリームを準備する直前に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download	TIMESTAMP
2	ml_username	VARCHAR (128)
3	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

Mobile Link 同期サーバは、特定のテーブルのダウンロード情報を準備する最初の手順として、このイベントを実行します。ダウンロード情報は、専用トランザクションで準備されます。このイベントの実行が、このトランザクションで最初のテーブル固有のアクションとなります。

リモート・データベースのテーブルごとに、begin_download スクリプトを 1 つ指定できます。このスクリプトは、テーブルが同期されている場合にのみ呼び出されます。

last_download タイムスタンプは、ダウンロード・フェーズの直前、最後に成功した同期の間に統合データベースから取得された値です。現在のユーザが同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

参照

◆ 「end_download テーブル・イベント」485 ページ

SQL の例

次の例は、Adaptive Server Anywhere 9 データベースで使用できます。最初の部分のコードは ml_add_table_script を呼び出し、2 番目のものは BeginTableDownload プロシージャを作成します。

```
call ml_add_table_script(
    'version1',
    'Leads',
    'begin_download',
    'call BeginTableDownLoad( ?, ?, ? ) );

 create procedure BeginTableDownload(
    LastDownload timestamp,
    MLUser varchar(128),
    TableName varchar(128) )
 begin
   execute immediate 'update ' || TableName ||
 ' set last_download_check = CURRENT TIMESTAMP
   where Owner = ' ||MLUser;
 end
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginDownloadTable という Java メソッドを begin_download テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
    'ver1',
    'table1',
    'begin_download',
    'ExamplePackage.ExampleClass.beginDownloadTable' )
```

次に示すのは、サンプルの Java メソッド beginDownloadTable です。このメソッドは、後のメンバ関数呼び出しで使用するために現在のテーブルの名前を保存します。

```
public String beginDownloadTable(
    Timestamp ts,
    String user,
    String table )
 {  _curTable = table;
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginTableDownload という .NET メソッドを begin_download テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1', 'table1', 'begin_download',
   'TestScripts.Test.BeginTableDownload'
 )
```

次に示すのは、サンプルの .NET メソッド BeginTableDownload です。このメソッドは、後のメンバ関数呼び出しで使用するために現在のテーブルの名前を保存します。

```
public void BeginTableDownload(
   DateTime timestamp,
   string user,
   string table )
 {  _curTable = table;
    return( null ); }
```

begin_download_deletes テーブル・イベント

機能

リモート・データベース内の指定したテーブルから削除するローのリストをフェッチする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download	TIMESTAMP
2	ml_username	VARCHAR (128)
3	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

このイベントは、リモート・データベースの指定したテーブルから削除するローのリストをフェッチする直前に実行されます。

リモート・データベースのテーブルごとに、begin_download_deletes スクリプトを 1 つ指定できます。

last_download タイムスタンプは、ダウンロード・フェーズの直前、最後に成功した同期の間に統合データベースから取得された値です。現在のユーザが同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

参照

- 「begin_download_rows テーブル・イベント」440 ページ
- 「end_download_rows テーブル・イベント」491 ページ

SQL の例

リモート・データベース上のデータ量を最小限に抑えるために、このイベントを使用して、download_delete_cursor の実行時に削除するデータにフラグを設定できます。次の例では、リモート・デバイスの 10 週以上経過した売上データに、削除対象を示すフラグを設定します。次の例は、Adaptive Server Anywhere 9 データベースで使用できます。このコードは ml_add_table_script を呼び出してから、beginDownloadDeletes プロシージャを作成します。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginDownloadDeletes という Java メソッドを begin_download_deletes テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
    'ver1',
    'table1',
    'begin_download_deletes',
    'ExamplePackage.ExampleClass.beginDownloadDeletes' )
```

次に示すのは、サンプルの Java メソッド beginDownloadDeletes です。このメソッドは、後のメンバ関数呼び出しで使用するために現在のテーブルの名前を保存します。

```
public String beginDownloadDeletes( Timestamp ts,
 String user, String table )
 {  _curTable = table;
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginDownloadDeletes という .NET メソッドを begin_download_deletes テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1', 'table1', 'begin_download_deletes',
   'TestScripts.Test.BeginDownloadDeletes'
 )
```

次に示すのは、サンプルの .NET メソッド BeginDownloadDeletes です。このメソッドは、後のメンバ関数呼び出しで使用するために現在のテーブルの名前を保存します。

```
public void BeginDownloadDeletes(
   DateTime timestamp,
   string user,
   string table )
 {  _curTable = table;
    return( null ); }
```

begin_download_rows テーブル・イベント

機能

リモート・データベース内の指定したテーブルで挿入または更新するローのリストをフェッチする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download	TIMESTAMP
2	ml_username	VARCHAR (128)
3	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

このイベントは、リモート・データベース内の指定したテーブルで挿入または更新されるローのストリームをフェッチする直前に、実行されます。

リモート・データベースのテーブルごとに、begin_download_rows スクリプトを 1 つ指定できます。

last_download タイムスタンプは、ダウンロード・フェーズの直前、最後に成功した同期の間に統合データベースから取得された値です。現在のユーザが同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

参照

- 「begin_download_deletes テーブル・イベント」437 ページ
- 「end_download_deletes テーブル・イベント」488 ページ

SQL の例

begin_download_rows テーブル・イベントを使用すると、このテーブルでダウンロードする必要がなくなったローにフラグを設定できます。次の例では、7 日以上経過した売上データをアーカイブします。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginDownloadRows という Java メソッドを begin_download_rows テーブル・イベント用のスクリプトとして登録します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
call ml_add_java_table_script(
    'ver1',
    'table1',
    'begin_download_rows',
    'ExamplePackage.ExampleClass.beginDownloadRows' )
```

次に示すのは、サンプルの Java メソッド beginDownloadRows です。このメソッドはテーブルとユーザを使用して UPDATE 文を生成します。Mobile Link はこの SQL 文を実行します。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginDownloadRows という .NET メソッドを begin_download_rows テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1', 'table1', 'begin_download_rows',
   'TestScripts.Test.BeginDownloadRows'
 )
```

次に示すのは、サンプルの .NET メソッド BeginDownloadRows です。このメソッドはテーブルとユーザを使用して UPDATE 文を生成します。Mobile Link はこの SQL 文を実行します。

```
public void BeginDownloadRows(
   DateTime timestamp,
   string user,
   string table )
 {  return( "update " + table + " set download_flag = 0
"
    + " where days(creation_time, CURRENT DATE) > 7 " +
    " and Owner = '" + user + "'" ); }
```

begin_publication 接続イベント

機能

同期しているパブリケーションに関する有用な情報を提供します。このスクリプトを使って、ファイル・ベースのダウンロードの世代番号を管理できます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

項目	パラメータ	説明
1	generation_number	INTEGER。これは INOUT パラメータです。使用している配備環境でファイル・ベースのダウンロードを使用しない場合、このパラメータは無視できます。デフォルトは 1 です。
2	ml_username	VARCHAR(128)。Ultra Light リモートが UL_SYNC_ALL を使用して同期されている場合、このイベントは 'unknown' という名前で 1 回呼び出されます。
3	publication_name	VARCHAR(128)
4	last_upload	TIMESTAMP。最後に成功したアップロード。
5	last_download	TIMESTAMP。最後に成功したダウンロード。

デフォルトのアクション

デフォルトの世代番号は 1 です。このイベントにスクリプトが定義されていない場合、リモートに送信される世代番号は必ず 1 になります。

説明

このイベントを使って、現在同期されているパブリケーションに基づいて、同期論理を設計できます。このイベントは、begin_synchronization イベントと同じトランザクションで呼び出され、begin_synchronization イベントの後で呼び出されます。このイベントは、パブリケーションが同期するたびに 1 回呼び出されます。

このイベントは、使用されるパブリケーションに基づいてダウンロードされるものに影響を与えることができます。たとえば、優先度パブリケーション (PriorityPub) と全テーブル用のパブリケーション (AllTablesPub) の双方の一部であるテーブルを考えます。begin_publication イベントのスクリプトは、Java クラスまたは SQL 変数やパッケージにパブリケーション名を保存できます。ダウンロード・スクリプトは、パブリケーションが PriorityPub と同期するか AllTablesPub と同期するかにより、異なった動作ができます。

世代番号

generation_number パラメータは、特にファイル・ベースのダウンロード用です。世代番号の出力値は、begin_synchronization スクリプトから end_synchronization スクリプトへ渡されます。generation_number の意味は、現在の同期がダウンロード・ファイルを作成するために使用されているか、現在の同期にアップロードが含まれているかによって異なります。

世代番号の出力値は、begin_publication スクリプトから end_publication スクリプトへ渡されます。generation_number の意味は、現在の同期がダウンロード・ファイルを作成するために使用されているか、現在の同期にアップロードが含まれているかによって異なります。

ファイルベースのダウンロードでは、世代番号を使って、ダウンロードの前にアップロードを強制的に行います。世代番号は、ダウンロード・ファイルに保存されます。アップロードを持つ同期中に、パブリケーションに対するサブスクリプションごとに 1 つの世代番号が出力されます。この番号はアップロード確認でリモート・データベースへ送信され、SYSSYNC.generation_number に保存されます。

参照

- 「end_publication 接続イベント」494 ページ
- 『Mobile Link 管理ガイド』> 「ファイルベースのダウンロード」
- 『Mobile Link 管理ガイド』> 「Mobile Link の世代番号」

SQL の例

同期されるパブリケーションごとに情報を記録する必要がある場合があります。次の例では、ml_add_connection_script を呼び出して、RecordPubSync というストアド・プロシージャにイベントを割り当てます。

```
call ml_add_connection_script(
 'version1',
 'begin_publication',
 '{call RecordPubSync( ?, ?, ?, ?, ? )}' );
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginPublication という Java メソッドを begin_publication 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script( 'ver1',
    'begin_publication',
    'ExamplePackage.ExampleClass.beginPublication' )
```

次に示すのは、サンプルの Java メソッド beginPublication です。このメソッドは後で使用する各パブリケーションの名前を保存します。

```
public String beginPublication(
     ianywhere.ml.script.InOutInteger
generation_number,
     String user,
     String pub_name,
     Timestamp last_upload,
     Timestamp last_download )
 {  _publicationNames[ _numPublications++ ] = pub_name;
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、BeginPub という .NET メソッドを begin_publication 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script( 'ver1',
   'begin_publication',
   'TestScripts.Test.BeginPub'
 )
```

次に示すのは、サンプルの .NET メソッド BeginPub です。このメソッドは後で使用する各パブリケーションの名前を保存します。

```
public void BeginPub(
     ref int generation_number,
     string user,
     string pub_name,
     DateTime last_upload,
     DateTime last_download )
 {
    _publicationNames[ _numPublications++ ] = pub_name;
 }
```

begin_synchronization 接続イベント

機能

同期処理の準備中にアプリケーションが Mobile Link 同期サーバに接続する時点で、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

説明

同期を準備するアプリケーションが Mobile Link 同期サーバとの接続を形成した直後に、Mobile Link 同期サーバがこのイベントを実行します。

このイベントは、アップロード・トランザクションの前に、別のトランザクションで実行されます。これは、統計値の管理に便利です。

参照

- 「end_synchronization 接続イベント」497 ページ
- 「begin_synchronization テーブル・イベント」449 ページ

SQL の例

ml_username の値を後続のスクリプトで何度も参照する場合は、その値をテンポラリ・テーブルまたは変数に格納できます。

```
Call ml_add_connection_script (
   'version1',
   'begin_synchronization',
   'set @EmployeeID = ?' );
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginSynchronizationConnection という Java メソッドを begin_synchronization 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script( 'ver1',
   'begin_synchronization',

'ExamplePackage.ExampleClass.beginSynchronizationConnec
tion'
  )
```

次に示すのは、サンプルの Java メソッド beginSynchronizationConnection です。このメソッドは後で使用する同期ユーザの名前を保存します。

```
public String beginSynchronizationConnection(
   String user )
 {  _curUser = user;
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、BeginSync という .NET メソッドを begin_synchronization 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script( 'ver1',
   'begin_synchronization',
   'TestScripts.Test.BeginSync'
 )
```

次に示すのは、サンプルの Java メソッド BeginSync です。このメソッドは後で使用する同期ユーザの名前を保存します。

```
public void BeginSync( string user )
 {  _curUser = user;
    return( null ); }
```

begin_synchronization テーブル・イベント

機能

同期処理の準備中にアプリケーションが Mobile Link 同期サーバに接続する時点で、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

同期を準備するアプリケーションが Mobile Link 同期サーバとの接続を形成すると、Mobile Link 同期サーバは、begin_synchronization 接続レベルのイベントを実行してから、このイベントを実行します。

リモート・データベースのテーブルごとに、begin_synchronization スクリプトを 1 つ指定できます。このイベントは、テーブルが同期されている場合にのみ呼び出されます。

参照

- 「end_synchronization テーブル・イベント」499 ページ
- 「begin_synchronization 接続イベント」447 ページ

SQL の例

begin_synchronization テーブル・イベントを使って、特定のテーブルの同期を設定します。次の Adaptive Server Anywhere の SQL プロシージャ・コールは、同期中にローを保存するためのテンポラリ・テーブルを作成するスクリプトを登録します。

```
call ml_add_table_script(
  'ver1',
  'sales_order',
  'begin_synchronization',
  'CREATE TABLE #sales_order
 (
     id            integer NOT NULL default
autoincrement,
     cust_id        integer NOT NULL,
     order_date        date NOT NULL,
     fin_code_id    char(2) NULL,
     region        char(7) NULL,
     sales_rep        integer NOT NULL,
     PRIMARY KEY (id),
 )' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginSynchronizationTable という Java メソッドを begin_synchronization テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_synchronization',

'ExamplePackage.ExampleClass.beginSynchronizationTable'
)
```

次に示すのは、サンプルの Java メソッド beginSynchronizationTable です。このメソッドは、このインスタンスに含まれるテーブル名のリストに現在のテーブル名を追加します。

```
public String beginSynchronizationTable(String user,
 String table )
 {  _tableList.add( table );
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginTableSync という .NET メソッドを begin_synchronization テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
  'ver1', 'table1',
   'begin_synchronization',
   'TestScripts.Test.BeginTableSync'
                  )
```

次に示すのは、サンプルの .NET メソッド BeginTableSync です。このメソッドは、このインスタンスに含まれるテーブル名のリストに現在のテーブル名を追加します。

```
public void BeginTableSync( string user, string table )
 {  _tableList.Add( table );
    return( null ); }
```

begin_upload 接続イベント

機能

Mobile Link 同期サーバがアップロードされた挿入、更新、削除のストリーム処理を開始する直前に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

説明

Mobile Link 同期サーバは、アップロードした情報を処理する最初の手順としてこのイベントを実行します。アップロード情報は 1 つのトランザクションで処理されます。このイベントは、このトランザクションで最初に実行されます。

参照

- 「end_upload 接続イベント」502 ページ
- 「begin_upload テーブル・イベント」455 ページ

SQL の例

begin_upload 接続イベントを使って、ローをアップロードする前に行う必要がある手順を実行できます。次の Adaptive Server Anywhere の SQL スクリプトは、sales_order テーブルの矛盾処理のために新旧のローの値を保存するテンポラリ・テーブルを作成します。

```
call ml_add_connection_script(
  'version1',
  'begin_upload',
  'CREATE TABLE #sales_order_conflicts
 (
     id            integer NOT NULL default
autoincrement,
     cust_id        integer NOT NULL,
     order_date        date NOT NULL,
     fin_code_id    char(2) NULL,
```

```
        region        char(7) NULL,
        sales_rep        integer NOT NULL,
        new_value        char(1) NOT NULL,
        PRIMARY KEY (id),
    )' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginUploadConnection という Java メソッドを begin_upload 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script( 'ver1',
'begin_upload',
 'ExamplePackage.ExampleClass.beginUploadConnection ' )
```

次に示すのは、サンプルの Java メソッド beginUploadConnection です。このメソッドは、Mobile Link 出力ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String beginUploadConnection( String user )
 {  java.lang.System.out.println(
      "Starting upload for user: " + user );
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、BeginUpload という .NET メソッドを begin_upload 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script(
   'ver1',
   'begin_upload',
   'TestScripts.Test.BeginUpload'
 )
```

次に示すのは、BeginUpload 呼び出し用の C# シグニチャです。

```
public void BeginUpload( string user )
```

次の C# サンプルは、後のイベントで使用するために現在のユーザ名を保存します。

```
public void BeginUpload( string curUser )
 {
     user = curUser;
 }
```

begin_upload テーブル・イベント

機能

Mobile Link 同期サーバがアップロードされた挿入、更新、削除のストリーム処理を開始する直前に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

Mobile Link 同期サーバは、アップロードした情報を処理する最初の手順としてこのイベントを実行します。アップロード情報は別のトランザクションで処理されます。このイベントの実行が、このトランザクションで最初のテーブル固有のアクションとなります。

リモート・データベースのテーブルごとに、begin_upload スクリプトを 1 つ指定できます。このスクリプトは、テーブルが実際に同期されている場合にのみ呼び出されます。

参照

- 「end_upload テーブル・イベント」504 ページ
- 「begin_upload 接続イベント」452 ページ

SQL の例

リモートからローをアップロードする場合は、変更内容を中間テーブルに入れて手動で処理できます。このイベントでは、グローバルなテンポラリ・テーブルを移植できます。

```
call ml_add_table_script(
   'version1',
   'Leads',
   'begin_upload',
   'insert into T_Leads SELECT *
 FROM Leads WHERE Owner = @EmployeeID')
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginUploadTable という Java メソッドを begin_upload テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'begin_upload',
   'ExamplePackage.ExampleClass.beginUploadTable'
  )
```

次に示すのは、サンプルの Java メソッド beginUploadTable です。このメソッドは、Mobile Link 出力ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String beginUploadTable( String user,
 String table )
 {  java.lang.System.out.println("Beginning to process
upload for: " + table);
 return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginTableUpload という .NET メソッドを begin_upload テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1', 'table1', 'begin_upload',
   'TestScripts.Test.BeginTableUpload'
 )
```

次に示すのは、サンプルの .NET メソッド BeginTableUpload です。このメソッドは、Mobile Link 出力ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void BeginTableUpload(
   string user,
   string table )
 {  System.Console.WriteLine("Beginning to process
upload for: " + table);
 return( null ); }
```

begin_upload_deletes テーブル・イベント

機能

リモート・データベース内の指定のテーブルから削除されたローをアップロードする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

このイベントは、2 番目のパラメータで指定したクライアント・テーブルでローを削除した結果生じる変更を適用する直前に、実行されます。

リモート・データベースのテーブルごとに、begin_upload_deletes スクリプトを 1 つ指定できます。このスクリプトは、テーブルが実際に同期されている場合にのみ呼び出されます。

参照

- 「end_upload_deletes テーブル・イベント」508 ページ

SQL の例

begin_upload_deletes 接続イベントは、特定のテーブルの挿入と更新をアップロードした後で、そのテーブルの削除をアップロードする前に行う必要のある手順を実行するために使用します。次の Adaptive Server Anywhere の SQL スクリプトは、アップロード中に一時的に削除を保存するためのテンポラリ・テーブルを作成します。

```
call ml_add_table_script(
  'ver1',
  'sales_order',
  'begin_upload_deletes',
  'CREATE TABLE #sales_order_deletes
 (
     id            integer NOT NULL default
autoincrement,
     cust_id        integer NOT NULL,
     order_date        date NOT NULL,
     fin_code_id    char(2) NULL,
     region        char(7) NULL,
     sales_rep        integer NOT NULL,
     PRIMARY KEY (id),
 )' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginUploadDeletes という Java メソッドを begin_upload_deletes テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script( 'ver1', 'table1',
 'begin_upload_deletes',
 'ExamplePackage.ExampleClass. beginUploadDeletes' )
```

次に示すのは、サンプルの Java メソッド beginUploadDeletes です。このメソッドは、Mobile Link 出力ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String beginUploadDeletes(
    String user,
    String table )
  throws java.sql.SQLException
 {  java.lang.System.out.println( "Starting upload
    deletes for table: " + table );
     return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginUploadDeletes という .NET メソッドを begin_upload_deletes テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script( 'ver1', 'table1',
   'begin_upload_deletes',
   'TestScripts.Test.BeginUploadDeletes'
 )
```

次に示すのは、サンプルの .NET メソッド BeginUploadDeletes です。このメソッドは、Mobile Link 出力ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void BeginUploadDeletes( string user,
   string table )
 {  System.Console.WriteLine( "Starting upload
    deletes for table: " + table );
     return( null ); }
```

begin_upload_rows テーブル・イベント

機能

リモート・データベース内の指定したテーブルから挿入と更新をアップロードする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

このイベントは、2 番目のパラメータで指定したクライアント・テーブルに対する挿入と削除から生じる変更を適用する直前に、実行されます。

リモート・データベースのテーブルごとに、begin_upload_rows スクリプトを 1 つ指定できます。このスクリプトは、テーブルが実際に同期されている場合にのみ呼び出されます。

参照

- 「end_upload_rows テーブル・イベント」511 ページ

SQL の例

begin_upload_rows 接続イベントを使って、特定のテーブルの挿入と更新をアップロードする前に行う必要がある手順を実行します。次のスクリプトは、統合データベースで Inventory テーブルへの挿入と更新を準備するストアド・プロシージャを呼び出します。

```
call ml_add_table_script(
  'MyCorp 1.0',
  'Inventory',
  'begin_upload_rows',
  'call PrepareForUpserts()' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginUploadRows という Java メソッドを begin_upload_rows テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'begin_upload_rows',
   'ExamplePackage.ExampleClass.beginUploadRows' )
```

次に示すのは、サンプルの Java メソッド beginUploadRows です。このメソッドは、Mobile Link 出力ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginUploadRows という .NET メソッドを begin_upload_rows テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1', 'table1', 'begin_upload_rows',
   'TestScripts.Test.BeginUploadRows'
 )
```

次に示すのは、サンプルの .NET メソッド BeginUploadRows です。このメソッドは、Mobile Link 出力ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void BeginUploadRows(
   string user,
   string table )
 {  System.Console.WriteLine( "Starting upload rows
    for table: " + table + " and user: " + user );
    return( null ); }
```

download_cursor テーブル・イベント

機能

ダウンロードされ、リモート・データベースで挿入または更新されるローを選択するためのカーソルを定義します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download	TIMESTAMP
2	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

デフォルトの download_cursor SQL スクリプトを生成するには、Mobile Link 同期サーバの -za オプションを使用します。また、Ultra Light アナライザがリファレンス・データベースに基づいた SELECT 文を生成します。この文を起動時に使用できます。

説明

Mobile Link 同期サーバは読み込み専用のカーソルを開き、それを使用してリモート・データベースにダウンロードするローのリストをフェッチします。このスクリプトには、適切な SELECT 文を含めてください。

パラメータは、last_download タイムスタンプとユーザ名です。SQL 文で疑問符を指定して選択すると、これらの値を使用できます。

リモート・データベースのテーブルごとに、download_cursor スクリプトを 1 つ指定できます。

last_download タイムスタンプは、ダウンロード・フェーズの直前、最後に成功した同期の間に統合データベースから取得された値です。現在のユーザが同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

Ultra Light クライアントに対する同期のダウンロード処理のパフォーマンスを最適化するには、プライマリ・キー値の範囲がデバイスで指定されている現在のローの外側にある場合に、ダウンロード・カーソル内のローをプライマリ・キー順に並べてください。たとえば、大きいリファレンス・テーブルをダウンロードする場合は、このような最適化による利点が得られます。

各 download_cursor スクリプトには、SELECT 文か、SELECT 文を含むプロシージャの呼び出しが必要です。Mobile Link 同期サーバは、SELECT 文を使用して統合データベース内でカーソルを定義します。

スクリプトでは、対応するリモート・データベース内のテーブルのカラムに対応するすべてのカラムを選択します。統合データベース内のカラムは、対応するリモート・データベースのカラムとは異なる名前にできますが、互換性のある型にしてください。

カラムは、対応するカラムがリモート・データベース内で定義されている順序に従って選択します。この順序は、リファレンス・データベースのカラム順と同じです。

download_cursor ではカスケード削除ができることに注意してください。そのため、データベースからレコードを削除できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 「ローをダウンロードするスクリプトの作成」308 ページ
- 「download_cursor スクリプトの作成」309 ページ
- 「リモート・データベース間でローを分割する」65 ページ
- 「download_delete_cursor テーブル・イベント」468 ページ

SQL の例

次の例は Oracle インストール環境の場合を示していますが、文はサポートされているすべてのデータベースに対して有効です。この例は、前回データをダウンロードした後に変更されたローのうち、emp_name カラム内のユーザ名と一致するローをすべてダウンロードします。

```
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 = ?' )
```

最初のパラメータ (last_download タイムスタンプ) を使用しないで、2 番目のパラメータ (Mobile Link ユーザ名) を使用する download_cursor SQL スクリプトを作成する場合は、ローに影響しないダミー句を追加します。次に例を示します。

```
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 =
?' )
```

引き続き両方のパラメータを使用しますが、最初の ? は何もしないプレースホルダです。

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、downloadCursor という Java メソッドを download_cursor テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
    'ver1',
    'ULCustomer',
    'download_cursor',
    'ExamplePackage.ExampleClass.downloadCursor ' )
```

次に示すのは、サンプルの Java メソッド downloadCursor です。このメソッドは、last_modified カラムが最終ダウンロード時間以上の場合に、SQL 文をダウンロード・ローに返します。

```
public String downloadCursor(java.sql.Timestamp
ts,String user )
  {
   return("SELECT cust_id, cust_name FROM ULCustomer
where last_modified >= ' " + ts + " ' ");
  }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、DownloadCursor という .NET メソッドを download_cursor テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1', 'table1', 'download_cursor',
   'TestScripts.Test.DownloadCursor'
 )
```

次に示すのは、サンプルの .NET メソッド DownloadCursor です。このメソッドは、テンポラリ・テーブルに *rows.txt* というファイルの内容を移植します。その結果、Mobile Link によって、テンポラリ・テーブルのローがリモート・データベースに送信されます。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
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 テーブル・イベント

機能

リモート・データベースで削除するローを選択するためのカーソルを定義します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download	TIMESTAMP
2	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

説明

Mobile Link 同期サーバは読み込み専用のカーソルを開き、リモート・データベースにダウンロードして挿入または削除するローのリストを、そのカーソルを使用してフェッチします。このスクリプトには、リモート・データベース内のテーブルから削除されるローのプライマリ・キー値を返す SELECT 文を含めてください。

パラメータは、last_download タイムスタンプとユーザ名です。SQL 文に疑問符を指定すると、これらの値を使用できます。

リモート・データベースのテーブルごとに、download_delete_cursor スクリプトを 1 つ指定できます。

テーブル内の 1 つ以上のローのプライマリ・キー・カラムで download_delete_cursor が NULL の場合、Mobile Link は、リモートにテーブル内のデータをすべて削除するように命令します。この動作の完全な説明については、『Mobile Link 管理ガイド』> 「テーブルから全ローを削除」を参照してください。

last_download タイムスタンプは、ダウンロード・フェーズの直前、最後に成功した同期の間に統合データベースから取得された値です。現在のユーザが同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

統合データベースから削除されたローは、download_delete_cursor イベントにより定義された結果セットには表示されないため、リモート・データベースから自動的に削除されないことに注意してください。リモート・データベースから削除されるローを識別するには、ローを非アクティブとして識別するカラムを統合データベース・テーブルに追加する方法があります。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 「download_cursor テーブル・イベント」464 ページ
- 「ローをダウンロードするスクリプトの作成」308 ページ
- 「リモート・データベース間でローを分割する」65 ページ
- 「download_delete_cursor スクリプトの作成」311 ページ

SQL の例

この例は Contact の例から抜粋したもので、*Samples¥MobiLink¥Contact¥build_consol.sql* にあります。この例は、このユーザが前回データをダウンロードした後に変更があった顧客 (`Customer.last_modified >= ?`) と、次のいずれかに該当する顧客をリモート・データベースから削除します。

- 同期中のユーザに属していない顧客 (`SalesRep.ml_username != ?`)
- 統合データベース内で非アクティブのマークが付いている顧客 (`Customer.active = 0`)

```
call ml_add_table_script(
    'ver1',
    'table1',
    'download_delete_cursor',
 'SELECT cust_id FROM Customer key join SalesRep
WHERE Customer.last_modified >= ? AND
( SalesRep.ml_username != ? OR Customer.active = 0
)')
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、downloadDeleteCursor という Java メソッドを download_delete_cursor イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
    'ver1',
    'table1',
    'download_delete_cursor',
    'ExamplePackage.ExampleClass.downloadDeleteCursor'
)
```

次に示すのは、サンプルの Java メソッド downloadDeleteCursor です。このメソッドは、ダウンロード削除カーソル用の SQL を生成する Java メソッドを呼び出します。

```
public String downloadDeleteCursor( Timestamp ts,
 String user )
 {  return( getDownloadCursor( _curUser, _curTable ) );
}
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、DownloadDeleteCursor という .NET メソッドを download_delete_cursor テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'download_delete_cursor',
   'TestScripts.Test.DownloadDeleteCursor'
 )
```

次に示すのは、サンプルの .NET メソッド DownloadDeleteCursor です。このメソッドは、ダウンロード削除カーソル用の SQL を生成する .NET メソッドを呼び出します。

```
public string DownloadDeleteCursor(
   DateTime timestamp,
   string user )
 {  return( getDownloadCursor( _curUser, _curTable ) );
}
```

download_statistics 接続イベント

機能

ダウンロード操作の同期統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR(128)。SYNCHRONIZATION USER 定義に指定した Mobile Link ユーザ名。
2	warnings	INTEGER。発行された警告の数。
3	errors	INTEGER。処理済みのエラーを含め、発生したエラーの数。
4	fetched_rows	INTEGER。download_cursor スクリプトによってフェッチされたローの数。
5	deleted_rows	INTEGER。download_deletes スクリプトによってフェッチされたローの数。
6	filtered_rows	INTEGER。(5) から実際にリモートに送信されたローの数。これには、アップロードされた値のダウンロード・フィルタが反映されます。
7	bytes	INTEGER。ダウンロードとしてリモートに送信されたバイト数。

デフォルトのアクション

なし。

説明

download_statistics イベントを使用すると、任意のユーザについてダウンロード統計を収集できます。ダウンロード・トランザクション終了時のコミット直前に、download_statistics 接続スクリプトが呼び出されます。

注意：
コマンド・ラインによっては、すべての警告やエラーのログが取られるとは限らないため、実際の警告数とエラー数が、ログを取られた警告数やエラー数より多くなる場合があります。

参照

- 「download_statistics テーブル・イベント」476 ページ
- 「upload_statistics 接続イベント」579 ページ
- 「upload_statistics テーブル・イベント」583 ページ
- 「synchronization_statistics 接続イベント」553 ページ
- 「synchronization_statistics テーブル・イベント」556 ページ
- 「time_statistics 接続イベント」559 ページ
- 「time_statistics テーブル・イベント」562 ページ
- 『Mobile Link 管理ガイド』> 「Mobile Link モニタ」

SQL の例

次の例は、同期の統計を download_audit というテーブルに挿入します。

```
call ml_add_connection_script(
   'ver1', 'download_statistics',
 'INSERT INTO download_audit(
   user_name, warnings, errors, deleted_rows,
   fetched_rows, download_rows, bytes )
 VALUES (?,?,?,?,?,?,?)')
```

監査テーブルに重要な統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、downloadStatisticsConnection という Java メソッドを download_statistics イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
   'ver1',
   'download_statistics',

'ExamplePackage.ExampleClass.downloadStatisticsConnecti
on' )
```

次に示すのは、サンプルの Java メソッド downloadStatisticsConnection です。このメソッドは、フェッチしたローの数を Mobile Link 出力ログに出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、DownloadStats という .NET メソッドを download_statistics 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script(
   'ver1',
   'download_statistics',
   'TestScripts.Test.DownloadStats'
 )
```

次に示すのは、サンプルの .NET メソッド DownloadStats です。このメソッドは、フェッチしたローの数を Mobile Link 出力ログに出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void DownloadStats(
   string user,
   int warnings,
   int errors,
   int deletedRows,
   int fetchedRows,
```

```
  int downloadRows,
  int bytes )
{  System.Console.WriteLine( "download connection
   stats fetchedRows: " + fetchedRows );
   return( null ); }
```

download_statistics テーブル・イベント

機能

ダウンロード操作の同期統計をテーブル別に追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR(128)。これは SYNCHRONIZATION USER 定義に指定した Mobile Link ユーザ名です。
2	table	VARCHAR(128)。テーブル名。
3	warnings	INTEGER。発行された警告の数。
4	errors	INTEGER。処理済みのエラーを含め、発生したエラーの数。
5	fetched_rows	INTEGER。download_cursor スクリプトによってフェッチされたローの数。
6	deleted_rows	INTEGER。download_deletes スクリプトによってフェッチされたローの数。
7	filtered_rows	INTEGER。(6) から実際にリモートに送信されたローの数。これには、アップロードされた値のダウンロード・フィルタが反映されます。
8	bytes	INTEGER。ダウンロードとしてリモートに送信されたバイト数。

デフォルトのアクション

なし。

説明

download_statistics イベントを使用すると、任意のユーザとテーブルについて、そのテーブルに適用されるダウンロードの統計を収集できます。ダウンロード・トランザクション終了時のコミット直前に、download_statistics テーブル・スクリプトが呼び出されます。

参照

- 「download_statistics 接続イベント」472 ページ
- 「upload_statistics 接続イベント」579 ページ
- 「upload_statistics テーブル・イベント」583 ページ
- 「synchronization_statistics 接続イベント」553 ページ
- 「synchronization_statistics テーブル・イベント」556 ページ
- 「time_statistics 接続イベント」559 ページ
- 「time_statistics テーブル・イベント」562 ページ
- 『Mobile Link 管理ガイド』> 「Mobile Link モニタ」

SQL の例

次の例は、同期の統計を download_audit というテーブルに挿入します。監査テーブルに重要な統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'download_statistics',
 'INSERT INTO download_audit (
   user_name, table, warnings, errors,
   deleted_rows, fetched_rows, download_rows, bytes)
 VALUES (?,?,?,?,?,?,?,?)')
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、downloadStatisticsTable という Java メソッドを download_statistics テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'download_statistics',

'ExamplePackage.ExampleClass.downloadStatisticsTable' )
```

次に示すのは、サンプルの Java メソッド downloadStatisticsTable です。このメソッドは、このテーブルの統計を Mobile Link 出力ログに出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、DownloadTableStats という .NET メソッドを download_statistics テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'download_statistics',
   'TestScripts.Test.DownloadTableStats'
 )
```

次に示すのは、サンプルの .NET メソッド DownloadTableStats です。このメソッドは、このテーブルの統計を Mobile Link 出力ログに出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void DownloadTableStats(
   string user,
   string table,
   int warnings,
   int errors,
   int deletedRows,
   int fetchedRows,
```

```
  int downloadRows,
  int bytes )
{  System.Console.WriteLine( "download table stats "
   + "table: " + table + "bytes: " + bytes );
   return( null ); }
```

end_connection 接続イベント

機能

停止準備中、または接続プールから接続が削除されるとき、Mobile Link 同期サーバが統合データベース・サーバとの接続を閉じる直前に、任意の文を処理します。

このスクリプトは通常、begin_connection スクリプトによって起動されたすべてのアクションを完了し、取得されていたリソースをすべて解放するために使用されます。

パラメータ

なし。

デフォルトのアクション

なし。

説明

Mobile Link 同期サーバと統合データベース・サーバ間の接続を閉じる直前に、end_connection スクリプトを使用して、選択したアクションを実行できます。

参照

- 「begin_connection 接続イベント」429 ページ

SQL の例

次の Adaptive Server Anywhere の SQL スクリプトは、begin_connection script が作成したテンポラリ・テーブルを削除します。厳密に言うと、このテーブルは明示的に削除する必要はありません。接続が切断されるときに ASA が自動的に削除します。テンポラリ・テーブルを明示的に削除する必要があるかどうかは、統合データベースのタイプによります。

```
call ml_add_connection_script(
  'version 1.0',
  'end_connection',
  'drop table #sync_info' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endConnection という Java メソッドを end_connection イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
    'ver1',
    'end_connection',
    'ExamplePackage.ExampleClass.endConnection' )
```

次に示すのは、サンプルの Java メソッド endConnection です。このメソッドは、Mobile Link 出力ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String endConnection()
 {  java.lang.System.out.println( "ending connection"
);
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、EndConnection という .NET メソッドを end_connection 接続イベント用のスクリプトとして登録します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
call ml_add_dnet_connection_script(
   'ver1',
   'end_connection',
   'TestScripts.Test.EndConnection'
 )
```

次に示すのは、サンプルの .NET メソッド EndConnection です。このメソッドは、Mobile Link 出力ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void EndConnection()
 {  System.Console.WriteLine( "ending connection" );
    return( null ); }
```

end_download 接続イベント

機能

Mobile Link 同期サーバがダウンロード・データの準備を完了した直後に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download	TIMESTAMP
2	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

説明

すべてのローのダウンロードと、ダウンロード通知を要求している場合は受信確認の受信後に、Mobile Link 同期サーバはこのスクリプトを実行します。ダウンロード情報は 1 つのトランザクションで処理されます。このスクリプトの実行は、このトランザクションの最後の非統計アクションです。

last_download タイムスタンプは、ダウンロード・フェーズの直前、最後に成功した同期の間に統合データベースから取得された値です。現在のユーザが同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

参照

- 「begin_download 接続イベント」432 ページ

SQL の例

次の例は、end_download 接続スクリプトの、考えられる用途の 1 つを示します。ULEmpCust テーブルの各ローには、action カラムが含まれています。次のスクリプトは、このカラムの値を使用して、レコードをリモート・データベースから削除します。

```
all ml_add_java_connection_script(
   'ver1',
   'end_download',
 'DELETE FROM ULEmpCust ec
   WHERE ? IS NOT NULL
     AND ec.emp_id = ? AND action = ''D''')
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endDownloadConnection という Java メソッドを end_download 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
   'ver1',
   'end_download',
   'ExamplePackage.ExampleClass.endDownloadConnection'
)
```

次に示すのは、サンプルの Java メソッド endDownloadConnection です。ULEmpCust テーブルの各ローには、action カラムが含まれています。次のスクリプトは、このカラムの値を使用して、レコードをリモート・データベースから削除します。また、ダウンロードが終了する前に、現在の Mobile Link 接続 (以前に保存した Mobile Link 接続) を使用して更新を実行します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、EndDownload という .NET メソッドを end_download 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script(
   'ver1',
   'end_download',
   'TestScripts.Test.EndDownload' )
```

次に示すのは、サンプルの .NET メソッド EndDownload です。ULEmpCust テーブルの各ローには、action カラムが含まれています。次のスクリプトは、このカラムの値を使用して、レコードをリモート・データベースから削除します。また、ダウンロードが終了する前に、現在の Mobile Link 接続 (以前に保存した Mobile Link 接続) を使用して更新を実行します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
public void EndDownload(
   DateTime timestamp,
   string user )
 {  string del_sql =    "DELETE FROM ULEmpCust ec " +
    "WHERE ec.emp_id = '" + user + "' " +
    "AND action = 'D' ";
    execUpdate( _syncConn, del_sql );
    return( null );
 }
```

end_download テーブル・イベント

機能

Mobile Link 同期サーバがダウンロードされた挿入、更新、削除のストリームの準備を終了した直後に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download	TIMESTAMP
2	ml_username	VARCHAR (128)
3	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

すべてのローのダウンロードと受信確認の受信後に、Mobile Link 同期サーバはこのスクリプトを実行します。ダウンロード情報は、別のトランザクションで準備されます。このスクリプトの実行が、このトランザクションで最後のテーブル固有の非統計アクションとなります。

リモート・データベースのテーブルごとに、end_download スクリプトを 1 つ指定できます。

last_download タイムスタンプは、ダウンロード・フェーズの直前、最後に成功した同期の間に統合データベースから取得された値です。現在のユーザが同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

参照

- 「begin_download テーブル・イベント」434 ページ
- 「end_download 接続イベント」482 ページ

SQL の例

end_download テーブル・イベントを使って、特定のテーブルをダウンロードした後で行う必要がある手順を実行します。次の Adaptive Server Anywhere の SQL スクリプトは、sales_summary テーブルのダウンロード・ローを保持するために prepare_for_download スクリプトが作成したテンポラリ・テーブルを削除します。

```
call ml_add_table_script(
  'MyCorp 1.0',
  'sales_summary',
  'end_download',
  'drop table #sales_summary_download' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endDownloadTable という Java メソッドを end_download テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script (
   'ver1',
   'table1',
   'end_download',
   'ExamplePackage.ExampleClass.endDownloadTable' )
```

次に示すのは、サンプルの Java メソッド endDownloadTable です。このメソッドは現在のテーブル・メンバ変数をリセットします。

```
public String endDownloadTable( Timestamp ts,
 String user, String table )
 {  _curTable = null;
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndTableDownload という .NET メソッドを end_download テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'end_download',
   'TestScripts.Test.EndTableDownload'
 )
```

次に示すのは、サンプルの .NET メソッド EndTableDownload です。このメソッドは現在のテーブル・メンバ変数をリセットします。

```
public void EndTableDownload
   DateTime timestamp,
   string user,
   string table )
 {  _curTable = null;
    return( null ); }
```

end_download_deletes テーブル・イベント

機能

リモート・データベース内の指定されたテーブルから削除するローのリストを準備した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download	TIMESTAMP
2	ml_username	VARCHAR (128)
3	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

このスクリプトは、リモート・データベース内の指定されたテーブルから削除されるローのリストを準備した直後に実行されます。

リモート・データベースのテーブルごとに、end_download_deletes スクリプトを 1 つ指定できます。

last_download タイムスタンプは、ダウンロード・フェーズの直前、最後に成功した同期の間に統合データベースから取得された値です。現在のユーザが同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

参照

- 「begin_download_deletes テーブル・イベント」437 ページ
- 「end_download 接続イベント」482 ページ
- 「begin_download_rows テーブル・イベント」440 ページ

- 「end_download_rows テーブル・イベント」491 ページ
- 「download_delete_cursor テーブル・イベント」468 ページ

SQL の例

リモート・データベース上のローに削除マークを付ける必要がある場合があります。次のスクリプトは、OnRemote という統合データベースのカラムを更新します。*注意：*UPDATE の WHERE 句は、download_delete_cursor イベントのスクリプトに使用される WHERE 句に一致します。

```
Call ml_add_table_script(
   'version1',
   'Leads',
   'end_download_deletes',
   'UPDATE Leads SET OnRemote = 0
       WHERE LastModified >= ?
         AND Owner = ? AND DeleteFlag=1');
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endDownloadDeletes という Java メソッドを end_download_deletes テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'end_download_deletes',
   'ExamplePackage.ExampleClass.endDownloadDeletes' )
```

リモート・データベース上のローに削除マークを付ける必要がある場合があります。次に示すのは、サンプルの Java メソッド endDownloadDeletes です。このメソッドは、レコードがリモート・データベース上に存在しなくなったことを示すため、OnRemote という統合データベースのカラムを更新します。*注意：*UPDATE の WHERE 句は、download_delete_cursor イベントのスクリプトに使用される WHERE 句に一致します。

```
public String endDownloadDeletes(
   Timestamp ts,
   String user,
   String table )
 { return( "UPDATE Leads SET OnRemote = 0
      WHERE LastModified >= ?
      AND Owner = ? AND DeleteFlag=1" ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndDownloadDeletes という .NET メソッドを end_download_deletes テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'end_download_deletes',
   'TestScripts.Test.EndDownloadDeletes'
 )
```

リモート・データベース上のローに削除マークを付ける必要がある場合があります。次に示すのは、サンプルの .NET メソッド EndDownloadDeletes です。このメソッドは、レコードがリモート・データベース上に存在しなくなったことを示すため、OnRemote という統合データベースのカラムを更新します。UPDATE の WHERE 句は、download_delete_cursor イベントのスクリプトに使用される WHERE 句に一致します。

```
public void EndDownloadDeletes(
   DateTime timestamp, string user, string table)
 { return( "UPDATE Leads SET OnRemote = 0
      WHERE LastModified >= ?
      AND Owner = ? AND DeleteFlag=1" ); }
```

end_download_rows テーブル・イベント

機能

リモート・データベース内の指定されたテーブルで挿入または更新するローのリストを準備した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download	TIMESTAMP
2	ml_username	VARCHAR (128)
3	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

このスクリプトは、リモート・データベース内の指定されたテーブルで挿入または更新されるローのストリームを準備した直後に実行されます。

リモート・データベースのテーブルごとに、end_download_rows スクリプトを 1 つ指定できます。

last_download タイムスタンプは、ダウンロード・フェーズの直前、最後に成功した同期の間に統合データベースから取得された値です。現在のユーザが同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

参照

- 「begin_download_rows テーブル・イベント」440 ページ
- 「end_download 接続イベント」482 ページ

- 「end_download_deletes テーブル・イベント」488 ページ
- 「begin_download_deletes テーブル・イベント」437 ページ

SQL の例

リモート・データベースへのダウンロードが成功したことを示すマークを、ローに付ける必要がある場合があります。次のスクリプトは、OnRemote という統合データベースのカラムを更新します。*注意：* UPDATE の WHERE 句は、download_delete_cursor イベントのスクリプトに使用される WHERE 句に一致します。

```
call ml_add_table_script(
    'version1',
    'Leads',
    'end_download_rows',
   'UPDATE Leads SET OnRemote = 1 WHERE LastModified >=
?
       AND Owner = ? AND DownloadFlag=1');
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endDownloadRows という Java メソッドを end_download_rows テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
    'ver1',
    'table1',
    'end_download_rows',
    'ExamplePackage.ExampleClass.endDownloadRows' )
```

次に示すのは、サンプルの Java メソッド endDownloadRows です。このメソッドは、Mobile Link 出力ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndDownloadRows という .NET メソッドを end_download_rows テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
  'ver1', 'table1', 'end_download_rows',
   'TestScripts.Test.EndDownloadRows'
 )
```

次に示すのは、サンプルの Java メソッド endDownloadRows です。このメソッドは、Mobile Link 出力ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void EndDownloadRows(
   DateTime timestamp,
   string user,
   string table )
 {  System.Console.WriteLine( "Done downloading
    inserts and updates for table " + table );
    return( null ); }
```

end_publication 接続イベント

機能

同期しているパブリケーションに関する有用な情報を提供します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

項目	パラメータ	説明
1	generation_number	INTEGER。使用している配備環境でファイル・ベースのダウンロードを使用しない場合、このパラメータは無視できます。デフォルトは 1 です。
2	ml_username	VARCHAR(128)。Ultra Light リモートが UL_SYNC_ALL を使用して同期されている場合、このイベントは 'unknown' という名前で 1 回呼び出されます。
3	publication_name	VARCHAR(128)
4	last_upload	TIMESTAMP。最後に成功したアップロード。
5	last_download	TIMESTAMP。最後に成功したダウンロード。

デフォルトのアクション

なし。

説明

このイベントを使って、現在同期されているパブリケーションに基づいて、同期論理を設計できます。このイベントは、end_synchronization イベントと同じトランザクションで呼び出され、end_synchronization イベントの前に呼び出されます。このイベントは、パブリケーションが同期するたびに 1 回呼び出されます。

現在の同期がアップロードを正常に適用すると、last_upload パラメータにはこの最終アップロードが適用された時刻が含まれます。現在の同期がダウンロードを正常に確認すると、last_download 時刻にはこの最終ダウンロードが生成された時刻が含まれます。これは、最終ダウンロード・タイムスタンプとしてダウンロード・スクリプトに渡された値と同じです。

世代番号

generation_number パラメータは、特にファイル・ベースのダウンロード用です。

世代番号の出力値は、begin_publication スクリプトから end_publication スクリプトへ渡されます。generation_number の意味は、現在の同期がダウンロード・ファイルを作成するために使用されているか、現在の同期にアップロードが含まれているかによって異なります。

ファイルベースのダウンロードでは、世代番号を使って、ダウンロードの前にアップロードを強制的に行います。世代番号は、ダウンロード・ファイルに保存されます。

参照

- 「begin_publication 接続イベント」443 ページ
- 『Mobile Link 管理ガイド』> 「ファイルベースのダウンロード」

SQL の例

同期されるパブリケーションごとに情報を記録する必要がある場合があります。次の例では、ml_add_connection_script を呼び出して、RecordPubEndSync というストアド・プロシージャにイベントを割り当てます。

```
call ml_add_connection_script(
    'version1',
    'end_publication',
    'call RecordPubEndSync( ?, ?, ?, ?, ? )' );
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endPublication という Java メソッドを end_publication 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
    'ver1',
    'end_publication',
    'ExamplePackage.ExampleClass.endPublication' )
```

次に示すのは、サンプルの Java メソッド endPublication です。このメソッドは、Mobile Link ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String endPublication(
     ianywhere.ml.script.InOutInteger
generation_number,
     String user,
     String pub_name,
     Timestamp last_upload,
     Timestamp last_download )
 {  System.out.println(
      "Finished synchronizing publication " + pub_name
);
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、EndPub という .NET メソッドを end_publication 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script( 'ver1',
   'end_publication',
   'TestScripts.Test.EndPub'
 )
```

次に示すのは、サンプルの Java メソッド endPublication です。このメソッドは、Mobile Link ログにメッセージを出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void EndPub(
     ref int generation_number,
     string user,
     string pub_name,
     DateTime last_upload,
     DateTime last_download )
 {
     Console.Write(
      "Finished synchronizing publication " + pub_name
);
 }
```

end_synchronization 接続イベント

機能

同期処理の完了時にアプリケーションを Mobile Link 同期サーバから切断する時点で、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	sync_ok	INTEGER。この値は、同期が成功すると 1 に、失敗すると 0 になります。

デフォルトのアクション

なし。

説明

同期が完了し、ダウンロード通知を要求している場合は Mobile Link クライアントからダウンロード・ストリームの受信確認が返された後に、Mobile Link 同期サーバはこのスクリプトを実行します。

このスクリプトは、ダウンロード・トランザクションの後に、別のトランザクションで実行されます。これは、統計値の管理に便利です。

参照

- 「begin_synchronization 接続イベント」447 ページ
- 「begin_synchronization テーブル・イベント」449 ページ
- 「end_synchronization テーブル・イベント」499 ページ

SQL の例

次の Adaptive Server Anywhere の SQL スクリプトは、同期試行の終了時刻と同期の成功または失敗を記録するストアド・プロシージャを呼び出します。

```
call ml_add_connection_script(
  'ver1',
  'end_synchronization',
  'call RecordEndOfSyncAttempt(?,?)' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endSynchronizationConnection という Java メソッドを end_synchronization イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
   'ver1',
   'end_synchronization',

'ExamplePackage.ExampleClass.endSynchronizationConnecti
on'
 )
```

次に示すのは、サンプルの Java メソッド endSynchronizationConnection です。このメソッドは JDBC 接続を使用して更新を実行します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、EndSync という .NET メソッドを end_synchronization 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script(
   'ver1',
   'end_synchronization',
   'TestScripts.Test.EndSync'
 )
```

次に示すのは、サンプルの .NET メソッド EndSync です。このメソッドは、テーブル sync_count を更新します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
public void EndSync( string user )
 {
   return("UPDATE sync_count set cnt =
    count + 1 where user_id = '" + user + "' ");
 }
```

end_synchronization テーブル・イベント

機能

同期処理の完了時にアプリケーションを Mobile Link 同期サーバから切断する時点で、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	table	VARCHAR (128)
3	sync_ok	INTEGER。この値は、同期が成功すると 1 に、失敗すると 0 になります。

デフォルトのアクション

なし。

説明

アプリケーションが同期を終了し、Mobile Link 同期サーバから切断しようとしているとき、Mobile Link 同期サーバは、同じ名前の接続レベルのスクリプトの前に、このスクリプトを実行します。

リモート・データベースのテーブルごとに、end_synchronization スクリプトを 1 つ指定できます。

参照

- 「begin_synchronization テーブル・イベント」449 ページ
- 「end_synchronization 接続イベント」497 ページ
- 「end_synchronization テーブル・イベント」499 ページ

SQL の例

次の Adaptive Server Anywhere の SQL スクリプトは、begin_synchronization スクリプトが作成したテンポラリ・テーブルを削除します。

```
call ml_add_table_script(
  'ver1',
  'sales_order',
  'end_synchronization',
  'drop table #sales_order' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endSynchronizationTable という Java メソッドを end_synchronization テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
    'ver1',
    'table1',
    'end_synchronization',

'ExamplePackage.ExampleClass.endSynchronizationTable' )
```

次に示すのは、サンプルの Java メソッド endSynchronizationTable です。このメソッドは、begin_synchronization スクリプトが作成したテンポラリ・テーブルを削除する SQL を返します。

```
public String endSynchronizationTable(  String user,
 String table )
 {  return( "drop table #sales_order" ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndTableSync という .NET メソッドを end_synchronization テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1', 'table1', 'end_synchronization',
   'TestScripts.Test.EndTableSync'
 )
```

次に示すのは、サンプルの Java メソッド EndTableSync です。このメソッドは、begin_synchronization スクリプトが作成したテンポラリ・テーブルを削除する SQL を返します。

```
public void EndTableSync( string user, string table )
 {  return( "drop table #sales_order" ); }
```

end_upload 接続イベント

機能

Mobile Link 同期サーバがアップロードされた挿入、更新、削除の処理を完了した直後に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

説明

Mobile Link 同期サーバは、アップロードした情報を処理する最後のステップとしてこのスクリプトを実行します。アップロード情報は 1 つのトランザクションで処理されます。このスクリプトは、このトランザクションで統計スクリプトの前に実行される最後のアクションです。

参照

- 「begin_upload 接続イベント」452 ページ
- 「end_upload テーブル・イベント」504 ページ

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endUploadConnection という Java メソッドを end_upload 接続イベント用のスクリプトとして登録します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
call ml_add_java_connection_script(
    'ver1',
    'end_upload',
    'ExamplePackage.ExampleClass.endUploadConnection' )
```

次に示すのは、サンプルの Java メソッド endUploadConnection です。このメソッドは、データベースの操作を実行するメソッドを呼び出します。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、EndUpload という .NET メソッドを end_upload 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script(
   'ver1',
   'end_upload',
   'TestScripts.Test.EndUpload'
 )
```

次に示すのは、EndUpload 呼び出し用の C# シグニチャです。

```
public void EndUpload( string user )
```

end_upload テーブル・イベント

機能

Mobile Link 同期サーバがアップロードされた挿入、更新、削除のストリーム処理を終了した直後に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

Mobile Link 同期サーバは、アップロードした情報を処理する最後のステップとしてこのスクリプトを実行します。アップロード情報は別のトランザクションで処理されます。このスクリプトの実行が、このトランザクションで最後のテーブル固有のアクションになります。

リモート・データベースのテーブルごとに、end_upload スクリプトを 1 つ指定できます。

参照

- 「begin_upload テーブル・イベント」455 ページ
- 「end_upload 接続イベント」502 ページ

SQL の例

次の文は、ストアド・プロシージャと end_upload スクリプトを定義します。最初の部分のコードは ml_add_table_script を呼び出し、2 番目のものは ULCustomerIDPool_maintain プロシージャを作成します。

```
ml_add_table_script(
   'custdb',
   'ULCustomerIDPool',
   'end_upload',
ULCustomerIDPool_maintain( ? );)

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;
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endUploadTable という Java メソッドを end_upload テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
    'ver1',
    'table1'
    'end_upload',
    'ExamplePackage.ExampleClass.endUploadTable' )
```

次に示すのは、サンプルの Java メソッド endUploadTable です。このメソッドは、渡されたテーブル名と関連する名前のテーブルに対する削除を生成します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
public String endUploadTable(   String user,
 String table)
 {  return( "DELETE from '" + table + "_temp'" ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndTableUpload という .NET メソッドを end_upload テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
    'ver1',
    'table1',
    'end_upload',
    'TestScripts.Test.EndTableUpload'
 )
```

次に示すのは、EndTableUpload 呼び出し用の C# シグニチャです。

```
public void EndTableUpload(
    string user, string table )
```

次の C# サンプルは、テンポラリ・テーブルに挿入されたローを、スクリプトに渡されたテーブルに移動します。

```
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 テーブル・イベント

機能

リモート・データベース内の指定されたテーブルからアップロードされた削除を適用した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

このスクリプトは、2 番目のパラメータで指定したリモート・テーブルでローを削除した結果生じる変更を適用した直後に実行されます。

リモート・データベースのテーブルごとに、end_upload_deletes スクリプトを 1 つ指定できます。

参照

- 「begin_upload_deletes テーブル・イベント」458 ページ

SQL の例

このイベントを使用すると、中間テーブル上でアップロード・ストリーム中に削除されたローを処理できます。ベース・テーブルのローを中間テーブルのローと比較して、削除されたローの処理を決定できます。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endUploadDeletes という Java メソッドを end_upload_deletes テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
    'ver1',
    'table1',
    'end_upload_deletes',
    'ExamplePackage.ExampleClass.endUploadDeletes' )
```

次に示すのは、サンプルの Java メソッド endUploadDeletes です。このメソッドは、データベースを操作する Java メソッドを呼び出します。

```
public String endUploadDeletes( String user,
 String table )
   throws java.sql.SQLException
 {  processUploadedDeletes( _syncConn, table );
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndUploadDeletes という .NET メソッドを end_upload_deletes テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
    'ver1',
    'table1',
    'end_upload_deletes',
    'TestScripts.Test.EndUploadDeletes'
 )
```

次に示すのは、サンプルの .NET メソッド EndUploadDeletes です。このメソッドは、データベースを操作する .NET メソッドを呼び出します。

```
public void EndUploadDeletes( string user, string table
)
 {  processUploadedDeletes( _syncConn, table );
    return( null ); }
```

end_upload_rows テーブル・イベント

機能

リモート・データベース内の指定されたテーブルからアップロードされた挿入と更新を適用した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

アップロードされた情報によっては、統合データベースでローを挿入したり更新したりする必要があります。このスクリプトは、2 番目のパラメータで指定したリモート・テーブルに対する修正の結果生じる変更を適用した直後に実行されます。

リモート・データベースのテーブルごとに、end_upload_rows スクリプトを 1 つ指定できます。

参照

- 「begin_upload_rows テーブル・イベント」461 ページ

SQL の例

このイベントを使用して、中間テーブル上でアップロード・ストリーム中に削除されたローを処理します。次の例で示すように、ベース・テーブルのローを中間テーブルのローと比較して、削除されたローの処理を決定できます。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endUploadRows という Java メソッドを end_upload_rows テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
    'ver1',
    'table1',
    'end_upload_rows',
    'ExamplePackage.ExampleClass.endUploadRows' )
```

次に示すのは、サンプルの Java メソッド endUploadRows です。このメソッドは、データベースを操作する Java メソッドを呼び出します。

```
public String endUploadRows(String user,
 String table )
   throws java.sql.SQLException
 {  processUploadedRows( _syncConn, table );
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndUploadRows という .NET メソッドを end_upload_rows テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'end_upload_rows',
   'TestScripts.Test.EndUploadRows'
 )
```

次に示すのは、サンプルの .NET メソッド endUploadRows です。このメソッドは、データベースを操作する .NET メソッドを呼び出します。

```
public void EndUploadRows(
   string user,
   string table )
 {  processUploadedRows( _syncConn, table );
    return( null ); }
```

example_upload_cursor テーブル・イベント

機能

Mobile Link 同期サーバが、リモート・データベースに挿入されたローを処理するためにアップロード・ストリームを処理している間は使用しないイベントを提供します。このイベントは呼び出されません。

パラメータ

項目	パラメータ
1	column 1
2	column 2
…	…

説明

文に基づく example_upload_cursor スクリプトは、example_upload_cursor 文で指定されたのと同じカラム値を直接挿入します。example_upload_cursor イベントは、Mobile Link では呼び出されません。

SQL の例

このスクリプトは呼び出されません。呼び出された場合は、値を統合データベース内のテーブル Customer に挿入します。このテーブルの最後のカラムでは、Customer がアクティブであると識別されます。最後のカラムは、リモート・データベースには含まれません。

```
SELECT cust_id, name, rep_id
 FROM customer
 WHERE cust_id=?
```

example_upload_delete テーブル・イベント

機能

アップロード・ストリームを処理し、リモート・データベースから削除されたローを処理します。このスクリプトは、Mobile Link では呼び出されません。

パラメータ

項目	パラメータ
1	column 1
2	column 2
…	…

説明

文に基づく example_upload_delete スクリプトは、リモート・データベース内で削除されるローを処理します。統合データベース側で実行される動作として DELETE 文を指定できますが、他の動作も指定できます。

参照

- 「upload_delete テーブル・イベント」568 ページ

SQL の例

この例は、リモート・データベースから削除される顧客に非アクティブのマークを付けます。

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

example_upload_insert テーブル・イベント

機能

リモート・データベースに挿入されたローを処理するために、Mobile Link 同期サーバがアップロード・ストリーム処理中に使用するイベントを提供します。

パラメータ

項目	パラメータ
1	column 1
2	column 2
…	…

説明

文に基づく example_upload_insert スクリプトは、upload_insert 文で指定されたのと同じカラム値を直接挿入します。

example_upload_insert イベントは呼び出されません。

参照

- 「upload_insert テーブル・イベント」572 ページ

SQL の例

このスクリプトは呼び出されません。呼び出された場合は、値を統合データベース内のテーブル Customer に挿入します。このテーブルの最後のカラムでは、Customer がアクティブであると識別されます。最後のカラムは、リモート・データベースには含まれません。

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

example_upload_update テーブル・イベント

機能

リモート・データベース側で更新されたローを処理する、アップロード・ストリーム用のサンプル・イベントです。サンプル・スクリプトは Mobile Link では呼び出されませんが、フォームは upload_update イベントと同じです。

パラメータ

句	パラメータ
SET	column 1
	column 2
	…
WHERE	primary key 1
	primary key 2
	…

説明

example_upload_update イベント・スクリプトを作成するには、dbmlsync コマンド・ラインでオプション -za を指定します。

参照

- 「upload_update テーブル・イベント」587 ページ

SQL の例

この例は、リモート・データベース内の Customer テーブルに対する更新を処理します。このスクリプトは、統合データベース内のテーブル Customer 内の値を更新します。注意 : このスクリプトが呼び出されることはなく、あくまでもサンプル・スクリプトとして扱われます。

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

handle_error 接続イベント

機能

Mobile Link 同期サーバで SQL エラーが発生したときに実行されます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	action_code	INTEGER。これは INOUT パラメータです。
2	error_code	INTEGER
3	error_message	TEXT
4	ml_username	VARCHAR (128)
5	table	VARCHAR(128)。スクリプトがテーブル・スクリプトでない場合は、テーブル名は NULL です。

デフォルトのアクション

handle_error スクリプトが定義されない場合、またはこのスクリプトが原因でエラーが発生した場合、デフォルト・アクション・コードは 3000 です。現在のトランザクションをロールバックし、現在の同期をキャンセルします。

説明

Mobile Link 同期サーバは、現在の action_code で送信します。最初は 1 回の SQL 操作によって発生したエラーのセットごとに 3000 が設定されます。通常、エラー数は SQL 操作ごとに 1 つのみですが、複数の場合もあります。この handle_error スクリプトは、セットに含まれるエラーごとに 1 回呼び出されます。最初のエラーに渡されるアクション・コードは 3000 です。以降の呼び出しには、直前の呼び出し

から返されたアクション・コードが渡されます。Mobile Link は、複数の呼び出しから返される値のうち最も大きい番号が付いた値を使用します。

スクリプト内でアクション・コードを修正し、Mobile Link に次の処理を指示する値を返すことができます。アクション・コード・パラメータには、次のいずれかの値を指定します。

- **1000**　現在のローをスキップして、処理を続行します。
- **3000**　現在のトランザクションをロールバックし、現在の同期をキャンセルします。これはデフォルト・アクション・コードで、handle_error スクリプトが定義されない場合、またはこのスクリプトが原因でエラーが発生した場合に使用されます。
- **4000**　現在のトランザクションのロールバック、同期のキャンセル、Mobile Link 同期サーバの停止を行います。

handle_error イベント用の SQL スクリプトは、ストアド・プロシージャとして実装してください。

Mobile Link 同期サーバは、同期処理中にエラーが発生したときにこのスクリプトを実行します。エラーの内容は、エラー・コードとメッセージで識別できます。同期の一部としてエラーが発生した場合は、ユーザ名が指定されます。それ以外の場合、この値は NULL です。

特定のテーブルの操作中にエラーが発生した場合は、テーブル名が指定されます。それ以外の場合、この値は NULL です。テーブル名は、クライアント・アプリケーションでのテーブル名です。この名前に直接対応するものが統合データベース内にあるかどうかは、同期システムの設計によって異なります。

Mobile Link 同期サーバが次に何を行うかは、アクション・コードに示されます。Mobile Link 同期サーバは、エラーの重大度に応じてアクション・コードにデフォルト値を設定してから、このスクリプトを呼び出します。この値は、スクリプトで変更することができます。スクリプトは、必ずアクション・コードを返すか設定するようにします。

handle_error スクリプトから値を返すには、次の 2 種類の方法があります。

- 次のように、プロシージャの OUTPUT パラメータにアクション・パラメータを渡す。

```
CALL my_handle_error( ?, ?, ?, ?, ? )
```

- 次のように、プロシージャまたは関数の戻り値を介してアクション・コードを設定する。

```
? = CALL my_handle_error( ?, ?, ?, ? )
```

  ほとんどの DBMS では、RETURN 文を使用してプロシージャまたは関数からの戻り値を設定します。

CustDB サンプル・アプリケーションには、さまざまなデータベース管理環境に対するエラー・ハンドラが含まれています。

参照

- 「report_error 接続イベント」544 ページ
- 「report_odbc_error 接続イベント」547 ページ
- 「handle_odbc_error 接続イベント」522 ページ

SQL の例

次の例は、Adaptive Server Anywhere 統合データベースで動作します。これによって、アプリケーションは冗長挿入を無視できます。

```
call ml_add_connection_script(
     'ver1',
     'handle_error',
     'call ULHandleError(?,?,?,?,?)' )


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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、handleError という Java メソッドを handle_error 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
     'ver1',
     'handle_error',
     'ExamplePackage.ExampleClass.handleError' )
```

次に示すのは、サンプルの Java メソッド handleError です。このメソッドは、渡されたデータに基づいてエラーを処理します。また、エラーの結果生じるエラー・コードも判別します。

```
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 接続イベント

機能

ODBC ドライバ・マネージャによってトリガされたエラーを Mobile Link 同期サーバが検出すると、実行されます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	action_code	INTEGER。これは INOUT パラメータです。
2	ODBC_state	VARCHAR(5)
3	error_message	TEXT
4	ml_username	VARCHAR (128)
5	table	VARCHAR (128)

デフォルトのアクション

Mobile Link 同期サーバは、デフォルトのアクション・コードを選択します。スクリプト内でアクション・コードを修正し、Mobile Link に次の処理を指示する値を返すことができます。アクション・コード・パラメータには、次のいずれかの値を指定します。

- **1000** 現在のローをスキップして、処理を続行します。
- **3000** 現在のトランザクションをロールバックし、現在の同期をキャンセルします。これはデフォルト・アクション・コードで、handle_error スクリプトが定義されない場合、またはこのスクリプトが原因でエラーが発生した場合に使用されます。

- **4000**　現在のトランザクションのロールバック、同期のキャンセル、Mobile Link 同期サーバの停止を行います。

説明

Mobile Link 同期サーバは、同期処理中に ODBC ドライバ・マネージャによってフラグが設定されたエラーを検出したときに、このスクリプトを実行します。エラーの内容は、エラー・コードで識別できます。

Mobile Link 同期サーバが次に何を行うかは、アクション・コードに示されます。Mobile Link 同期サーバは、エラーの重大度に応じてアクション・コードにデフォルト値を設定してから、このスクリプトを呼び出します。この値は、スクリプトで変更することができます。スクリプトは、必ずアクション・コードを返すか設定するようにします。

handle_odbc_error スクリプトは、handle_error スクリプトと report_error スクリプトの後、report_odbc_error スクリプトの前に呼び出されます。

エラー処理スクリプトがどちらか 1 つだけ定義されている場合、そのスクリプトからの戻り値によってエラー処理が決定されます。エラー処理スクリプトが両方とも定義されている場合、Mobile Link 同期サーバは数値の一番大きいアクション・コードを使用します。handle_error と handle_ODBC_error の両方が定義されると、Mobile Link はすべての呼び出しから返されるアクション・コードのうち、最も大きい番号を持つものを使用します。

参照

- 「handle_error 接続イベント」518 ページ
- 「report_error 接続イベント」544 ページ
- 「report_odbc_error 接続イベント」547 ページ

SQL の例

次の例は、Adaptive Server Anywhere 統合データベースで動作します。これによって、アプリケーションは ODBC 整合性制約違反を無視できます。

```
call ml_add_connection_script(
  'ver1',
  'handle_odbc_error',
  'call HandleODBCError( ?, ?, ?, ?, ? )'  )
 CREATE PROCEDURE HandleODBCError( INOUT action
integer,
 IN odbc_state varchar(5), IN error_message
varchar(1000),
```

```
 IN user_name varchar(128), IN table_name varchar(128)
)
 BEGIN
   if odbc_state = '23000' then
       -- ignore the error and keep going
       SET action = 1000;
   else
       -- abort the synchronization
       SET action = 3000;
   end if;
 END
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、handleODBCError という Java メソッドを handle_odbc_error イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
 'ver1', 'handle_odbc_error',
'ExamplePackage.ExampleClass.handleODBCError'
 )
```

次に示すのは、サンプルの Java メソッド handleODBCError です。このメソッドは、渡されたデータに基づいてエラーを処理します。また、エラーの結果生じるエラー・コードも判別します。

```
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_error_message 接続イベント

機能

このスクリプトを使用すると、リモート・データベースに送信される (エラー、警告、情報) メッセージのテキストをカスタマイズできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	error_message	VARBINARY(1024)。これは INOUT パラメータです。
2	ml_username	VARCHAR (128)
3	error_code	INT

デフォルトのアクション

なし。

説明

modify_error_message イベントの SQL スクリプトは、ストアド・プロシージャとして実装してください。

SQL の例

次の例は、データベースが 1 日前から現在までの間に同期されたかどうかに関わらず、1 日前からのデータをすべてダウンロードします。

最初に、Adaptive Server Anywhere 統合データベースのプロシージャを作成します。

```
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
```

次に、Adaptive Server Anywhere 統合データベースに、スクリプトをインストールします。

```
call ml_add_connection_script(
   'modify_ts_test',
   'modify_last_download_timestamp',
   'call  ModifyLastDownloadTimestamp ( ?, ? )' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、modifyLastDownloadTimestamp という Java メソッドを modify_last_download_timestamp 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
   'ver1',
   'modify_last_download_timestamp',

'ExamplePackage.ExampleClass.modifyLastDownloadTimestam
p' )
```

次に示すのは、サンプルの Java メソッド modifyLastDownloadTimestamp です。このメソッドは、現在の新しいタイムスタンプを出力し、渡されたタイムスタンプを修正します。

```
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_last_download_timestamp 接続イベント

機能

このスクリプトを使用して、現在の同期で last_download タイムスタンプを修正できます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download_timestamp	TIMESTAMP。これは INOUT パラメータです。
2	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

説明

このスクリプトは、現在の同期に対する最終ダウンロード・タイムスタンプを変更するために使用します。このスクリプトが定義されている場合、Mobile Link 同期サーバがこのスクリプトを呼び出し、ダウンロード・スクリプトに渡す last_download タイムスタンプとして、修正した last_download タイムスタンプを使用します。

modify_last_download_timestamp イベントの SQL スクリプトは、ストアド・プロシージャとして実装してください。Mobile Link 同期サーバは、ストアド・プロシージャへの最初のパラメータとして last_download_timestamp を渡し、タイムスタンプをストアド・プロシージャが渡した最初の値で置き換えます。

たとえば、次の download_cursor スクリプトを定義する場合、Mobile Link 同期サーバは ? を変更された最終ダウンロード・タイムスタンプの値に置き換えてから、SELECT 文を実行します。

```
SELECT pk, c2, c3 FROM test WHERE last_modified >= ?
```

このスクリプトは、同じトランザクション内にある prepare_for_download スクリプトの直前に実行されます。

SQL の例

最初に、Adaptive Server Anywhere 統合データベースのプロシージャを作成します。次に、Oracle の場合の例を示します。

```
CREATE PROCEDURE modify_nldts(
         nldts        OUT DATE,
         ldts        IN DATE,
         user_name     IN VARCHAR )
         AS
         BEGIN
          -- N is max replication latency in
consolidated cluster
         nldts := nldts - N;
         END;
```

次に、Adaptive Server Anywhere、Adaptive Server Enterprise、Microsoft SQL Server の場合の例を示します。

```
CREATE PROCEDURE modify_nldts
         @nldts     DATETIME OUTPUT,
         @ldts      DATETIME,
         @t_name    VARCHAR( 128 )
         AS
         BEGIN
          -- N is max replication latency in
consolidated cluster
         SELECT @nldts = @nldts - N
         END
```

次に、スクリプトを統合データベースにインストールします。以下は、Adaptive Server Anywhere 統合データベースの場合の例です。

```
ml_add_connection_script(
   'my_version',
   'modify_last_download_timestamp',
   '{call modify_nldts(?,?,?)}' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、modifyLastDownloadTimestamp という Java メソッドを modify_last_download_timestamp 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
   'ver1',
   'modify_last_download_timestamp',

'ExamplePackage.ExampleClass.modifyLastDownloadTimestam
p' )
```

次に示すのは、サンプルの Java メソッド modifyLastDownloadTimestamp です。このメソッドは、現在の新しいタイムスタンプを出力し、渡されたタイムスタンプを修正します。

```
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 接続イベント

機能

このスクリプトを使用して、次の同期の last_download タイムスタンプを修正できます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	download_timestamp	TIMESTAMP。これは INOUT パラメータです。
2	last_download_timestamp	TIMESTAMP
3	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

説明

このスクリプトは、次の同期に対する最終ダウンロード・タイムスタンプを変更するために使用します。このスクリプトが定義されている場合、Mobile Link 同期サーバがこのスクリプトを呼び出し、次の最終ダウンロード・タイムスタンプをリモートに送信します。このタイムスタンプは、次の同期の一部としてリモートから送信されます。

modify_next_last_download_timestamp イベントの SQL スクリプトは、ストアド・プロシージャとして実装してください。Mobile Link 同期サーバは、ストアド・プロシージャの最初のパラメータとして download_timestamp を渡し、タイムスタンプをストアド・プロシージャが渡した最初の値で置き換えます。

このスクリプトを使用して、Mobile Link 同期サーバが Mobile Link クライアントに送信したダウンロード・タイムスタンプを変更できます。クライアントが dbmlsync である場合、タイムスタンプは SYSSYNC システム・テーブルに保存されます。

このスクリプトは、ユーザ・テーブルがダウンロードされた後にダウンロード・トランザクションで実行されます。

SQL の例

次の例は、このスクリプトの適用例の 1 つを示します。最初に、Adaptive Server Anywhere 統合データベースのプロシージャを作成します。

```
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
```

次に、Adaptive Server Anywhere 統合データベースに、スクリプトをインストールします。

```
call ml_add_connection_script(
   'modify_ts_test',
   'modify_next_last_download_timestamp',
   'call  ModifyNextDownloadTimestamp ( ?, ?, ? )' )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、modifyNextDownloadTimestamp という Java メソッドを modify_next_last_download_timestamp 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
    'ver1',
    'modify_next_last_download_timestamp',

'ExamplePackage.ExampleClass.modifyNextDownloadTimestam
p' )
```

次に示すのは、サンプルの Java メソッド modifyNextDownloadTimestamp です。このメソッドは、ダウンロード・タイムスタンプを 1 時間前に設定します。

```
public String modifyNextDownloadTimestamp(
   Timestamp download_timestamp,
   Timestamp last_download,
   String user_name )
 {  download_timestamp.setHours(
    download_timestamp.getHours() -1 );
    return( null ); }
```

modify_user 接続イベント

機能

Mobile Link ユーザ名を指定します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

項目	パラメータ	説明
1	ml_username	VARCHAR(128)。これは INOUT パラメータです。

デフォルトのアクション

なし。

説明

Mobile Link サーバは、ユーザ名をパラメータとして指定します。このパラメータによってスクリプトが呼び出され、Mobile Link クライアントからユーザ名が送信されます。必要に応じて、代替ユーザ名を作成することもできます。このスクリプトを使用すると、Mobile Link スクリプトの呼び出しで使用するユーザ名を変更できます。

ml_username パラメータは、ユーザ名を格納するのに十分な長さでなければなりません。

modify_user イベント用の SQL スクリプトは、ストアド・プロシージャとして実装してください。

参照

- 「authenticate_user 接続イベント」420 ページ
- 「authenticate_user_hashed 接続イベント」425 ページ

SQL の例

次の例は、Adaptive Server Anywhere 統合データベースで動作します。マッピング・テーブル user_device を使用して、リモート・データベースのユーザ名をデバイスを使用しているユーザの ID にマッピングします。この方法は、同じ人物が複数のリモート (PDA やラップトップなど) を所有し、(ユーザの名前または ID に基づいて) 同じ同期論理が必要なときに使用できます。

```
call ml_add_connection_script(
  'ver1',
  'modify_user',
  'call ModifyUser( ? )'  )
 CREATE PROCEDURE ModifyUser( INOUT u_name varchar(128)
)
 BEGIN
  select user_name
     into u_name
     from user_device
     where device_name = u_name
 END
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、modifyUser という Java メソッドを modify_user 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
    'ver1',
    'modify_user',
    'ExamplePackage.ExampleClass.modifyUser' )
```

次に示すのは、サンプルの Java メソッド modifyUser です。このメソッドは、データベースからユーザ ID を取得し、それを使用してユーザ名を設定します。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、ModUser という .NET メソッドを modify_user 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script(
   'ver1',
   'modify_user',
   'TestScripts.Test.ModUser'
 )
```

次に示すのは、ModUser 呼び出し用の C# シグニチャです。

```
public void ModUser( string user )
```

new_row_cursor テーブル・イベント (旧式)

機能

以前にリモート・データベース内で更新されたが、現在、統合データベースにある値と競合しているローの新しい値を挿入するために、Mobile Link 同期サーバが使用する挿入カーソルを定義します。

アップロードを実行するには文ベースのイベントを使用してください
このスクリプトは使用されなくなりました。代わりに、文ベースのイベント upload_new_row_insert を使用してください。new_row_cursor イベントのサポートは、今後のリリースでは削除される可能性があります。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

説明

リモート・データベースでローが更新されると、Mobile Link クライアントは元の値のコピーを保存します。クライアントは、Mobile Link 同期サーバに古い値と新しい値の両方を送信します。また、強制的な競合モードで INSERT 操作を入力するために使用されます。

Mobile Link 同期サーバは更新されたローを受信すると、upload_cursor を使用して、元の値と統合データベース内の現在の値を比較します。アップロードされた古い値が統合データベース内の現在の値と一致しない場合、そのローに競合が発生します。ローを更新する代わりに、Mobile Link 同期サーバは古い値と新しい値の両方を、それぞれ old_row_cursor と new_row_cursor を使用して統合データベースに挿入します。

Mobile Link 同期サーバは、カーソルを使用してアップロードされた新しい値を競合するローから統合データベースに挿入します。このスクリプトには、このカーソルの定義に使用される SELECT 文が含まれます。

一般的には、テンポラリ・テーブルを使用して競合するローの古い値と新しい値を保存します。以前のスクリプトにこれらのテンポラリ・テーブルを作成できます。

リモート・データベースのテーブルごとに、new_row_cursor スクリプトを 1 つ指定できます。

通常、select リストのカラムは、順序と型の両方をクライアント・テーブルのカラムに一致させます。ただし、Mobile Link 同期サーバでは、カラムを 1 つ追加できます。その場合、Mobile Link 同期サーバは、ユーザ名を最初のカラムに自動的に挿入してから、通常どおり残りのカラムを使用して新しいローの値の挿入を続行します。

注意
次のスクリプトが同じテーブルに対して定義されている場合、このスクリプトは無視されます。upload_insert、upload_update、upload_delete、upload_fetch、upload_new_row_insert、upload_old_row_insert

---

参照

- 「upload_new_row_insert テーブル・イベント」575 ページ
- 『Mobile Link 管理ガイド』> 「競合の解決」
- 「resolve_conflict テーブル・イベント」550 ページ

SQL の例

次の SELECT 文は、CustDB サンプル・アプリケーションに適した *new_row_cursor* スクリプトを定義します。

```
call ml_add_table_script(
   'ver1',
   'table1',
   'new_row_cursor',
 'SELECT order_id, cust_id, prod_id, emp_id,
     disc, quant, notes, status
   FROM ULNewOrder FOR update')
```

ULOrder テーブルのプライマリ・キーは order_id です。

次の SELECT 文も、同じクライアント・テーブルで使用できます。異なる点は、ローが 1 つ追加されていることです。Mobile Link 同期サーバは、最初のカラムに自動的にユーザ名を保存します。

```
SELECT user_name, order_id, cust_id, prod_id,
     emp_id, disc, quant, notes, status
   FROM ULNewOrder FOR update
```

Java の例

このスクリプトは有効な SQL を返します。

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、newRowCursor という Java メソッドを new_row_cursor イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'new_row_cursor',
   'ExamplePackage.ExampleClass.newRowCursor' )
```

次に示すのは、サンプルの Java メソッド newRowCursor です。このメソッドは、Java メソッドを呼び出して新しいロー・カーソル文を動的に生成します。

```
public String newRowCursor()
 {  return( getRowCursor ( _curTable ) ); }
```

old_row_cursor テーブル・イベント (旧式)

機能

以前にリモート・データベースで更新されたが、現在、統合データベースにある値と競合するローの古い値を挿入するために、Mobile Link 同期サーバが使用するカーソルを定義します。このイベントは、強制的な競合モードで、削除されたローの値を挿入するためにも使用されます。

アップロードを実行するには文ベースのイベントを使用してください
このスクリプトは使用されなくなりました。代わりに、文ベースのイベント upload_old_row_insert を使用してください。old_row_cursor イベントのサポートは、今後のリリースでは削除される可能性があります。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

説明

リモート・データベースでローが更新されると、Mobile Link クライアントは元の値のコピーを保存します。クライアントは、Mobile Link 同期サーバに古い値と新しい値の両方を送信します。

Mobile Link 同期サーバは更新されたローを受信すると、upload_cursor を使用して、元の値と統合データベース内の現在の値を比較します。アップロードされた古い値が統合データベース内の現在の値と一致しない場合は、そのローに競合が発生します。ローを更新する代わりに、Mobile Link 同期サーバは古い値と新しい値の両方を、old_row_cursor イベントと new_row_cursor イベントを使用して統合データベースに挿入します。

一般的には、テンポラリ・テーブルを使用して競合するローの古い値と新しい値を保存します。Adaptive Server Anywhere では、以前のスクリプトにこれらのテーブルを作成できます。ASA 以外の統合データベースがテンポラリ・テーブルをサポートしている場合がありますが、通常は、ASA が提供するテンポラリ・テーブルとはかなり異なります。詳細については、使用している DBMS のマニュアルを参照してください。テンポラリ・テーブルの代わりに、Mobile Link ユーザ名用の追加カラムがあるベース・テーブルを使用できます。このカラムは、ベース・テーブルのローを同時同期間で効果的に分割します。

Mobile Link 同期サーバは、カーソルを使用してアップロードされた古い値を競合するローから統合データベースに挿入します。このスクリプトには、このカーソルの定義に使用される SELECT 文が含まれます。

リモート・データベースのテーブルごとに、old_row_cursor スクリプトを 1 つ指定できます。

通常、select リストのカラムは、順序と型の両方をクライアント・テーブルのカラムに一致させます。ただし、Mobile Link 同期サーバでは、カラムを 1 つ追加できます。その場合、Mobile Link 同期サーバは、ユーザ名を最初のカラムに自動的に挿入し、その後、通常どおり残りのカラムを使用して古いローの値の挿入を続行します。

参照

- 「upload_old_row_insert テーブル・イベント」577 ページ
- 『Mobile Link 管理ガイド』> 「競合の解決」
- 「resolve_conflict テーブル・イベント」550 ページ

SQL の例

次の SELECT 文は、Oracle インストール環境用の CustDB サンプル・アプリケーションに適した *old_row_cursor* スクリプトを定義します。ULOrder テーブルのプライマリ・キーは order_id です。

```
call ml_add_table_script(
   'ver1',
   'table1',
   'old_row_cursor',
 'SELECT order_id, cust_id, prod_id, emp_id,
     disc, quant, notes, status
  FROM ULOldOrder')
```

次の SELECT 文も、同じクライアント・テーブルで使用できます。異なる点は、ローが 1 つ追加されていることです。Mobile Link 同期サーバは、最初のカラムに自動的にユーザ名を保存します。

```
SELECT user_name, order_id, cust_id, prod_id,
     emp_id, disc, quant, notes, status
  FROM ULOldOrder
```

Java の例

このスクリプトは有効な SQL を返します。

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、oldRowCursor という Java メソッドを old_row_cursor イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'old_row_cursor',
   'ExamplePackage.ExampleClass.oldRowCursor' )
```

次に示すのは、サンプルの Java メソッド oldRowCursor です。このメソッドは、Java メソッドを呼び出して古いロー・カーソル文を動的に生成します。

```
public String oldRowCursor()
 {  return( getRowCursor( _curTable ) ); }
```

prepare_for_download 接続イベント

機能

アップロード・トランザクションとダウンロード・トランザクション間で必要な操作を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	last_download	TIMESTAMP
2	ml_username	VARCHAR (128)

デフォルトのアクション

なし。

説明

Mobile Link 同期サーバは、アップロード・トランザクションからダウンロード・トランザクションの開始までの間に、このスクリプトを別個のトランザクションとして実行します。

last_download タイムスタンプは、ダウンロード・フェーズの直前、最後に成功した同期の間に統合データベースから取得された値です。現在のユーザが同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

参照

- 「end_upload 接続イベント」502 ページ
- 「begin_download 接続イベント」432 ページ

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、prepareForDownload という Java メソッドを prepare_for_download イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
    'ver1',
    'prepare_for_download',
    'ExamplePackage.ExampleClass.prepareForDownload' )
```

次に示すのは、サンプルの Java メソッド prepareForDownload です。このメソッドはデータベース内のローを修正する Java メソッドを呼び出します。

```
public String prepareForDownload( Timestamp ts,
 String user )
 {  adjustUploadedRows( _syncConn, user );
    return( null ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、PrepareForDownload という .NET メソッドを prepare_for_download 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script( 'ver1',
   'prepare_for_download',
   'TestScripts.Test.PrepareForDownload'
 )
```

次に示すのは、PrepareForDownload 呼び出し用の C# シグニチャです。

```
public void PrepareForDownload(
   DateTime timestamp,
   string user )
```

report_error 接続イベント

機能

エラーのログを取ったり、handle_error スクリプトによって選択されたアクションを記録したりできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	action_code	INTEGER。このパラメータは必須です。
2	error_code	INTEGER。以下のパラメータを 1 つも指定しない場合、このパラメータはオプションです。
3	error_message	TEXT。以下のパラメータを 1 つも指定しない場合、このパラメータはオプションです。
4	ml_username	VARCHAR(128)。以下のパラメータを 1 つも指定しない場合、このパラメータはオプションです。
5	table	VARCHAR(128)。このパラメータはオプションです。

デフォルトのアクション

なし。

説明

このスクリプトを使用すると、エラーのログを取ったり、handle_error スクリプトによって選択されたアクションを記録したりできます。このスクリプトは、handle_error スクリプトが定義されているかどうか

に関わらず handle_error イベントの後に実行されます。また、同期接続とは異なるデータベース接続 (管理／情報接続) の専用トランザクションで常に実行されます。

エラーの内容は、エラー・コードとエラー・メッセージで識別できます。現在のエラーの原因となった SQL 操作について、エラー処理スクリプトの最後の呼び出しによってアクション・コード値が返されます。

同期の一部としてエラーが発生した場合は、ユーザ名が指定されます。それ以外の場合、この値は NULL です。

特定のテーブルの操作中にエラーが発生した場合は、テーブル名が指定されます。それ以外の場合、この値は NULL です。テーブル名は、リモート・データベースでのテーブル名です。この名前に直接対応するものが統合データベース内にあるかどうかは、同期システムの設計によって異なります。

参照

- 「handle_error 接続イベント」518 ページ
- 「handle_odbc_error 接続イベント」522 ページ
- 「report_odbc_error 接続イベント」547 ページ

SQL の例

次の例は、Adaptive Server Anywhere 統合データベースで動作します。これは、同期エラーを記録するために使用されるテーブルにローを挿入します。

```
call ml_add_connection_script(
  'ver1',
  'report_error',
  'insert into sync_error(
    action_code,
    error_code,
    error_message,
    user_name,
    table_name )
     values( ?, ?, ?, ?, ? )'  )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、reportError という Java メソッドを report_error 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
    'ver1',
    'report_error,
    'ExamplePackage.ExampleClass.reportError' )
```

次に示すのは、サンプルの Java メソッド reportError です。このメソッドは、Mobile Link が提供する JDBC 接続を使用してテーブルにエラーのログを取ります。また、アクション・コードも設定します。

```
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 接続イベント

機能

エラーのログを取ったり、handle_odbc_error スクリプトによって選択されたアクションを記録したりできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	action_code	INTEGER。このパラメータは必須です。
2	ODBC_state	VARCHAR(5)。以下のパラメータを 1 つも指定しない場合、このパラメータはオプションです。
3	error_message	TEXT。以下のパラメータを 1 つも指定しない場合、このパラメータはオプションです。
4	ml_username	VARCHAR(128)。以下のパラメータを 1 つも指定しない場合、このパラメータはオプションです。
5	table	VARCHAR(128)。このパラメータはオプションです。

デフォルトのアクション

なし。

説明

このスクリプトを使用すると、エラーのログを取ったり、handle_odbc_error スクリプトによって選択されたアクションを記録したりできます。このスクリプトは、handle_odbc_error スクリプトが定

義されているかどうかに関わらず handle_odbc_error イベントの後に実行されます。また、同期接続とは異なるデータベース接続 (管理／情報接続) の専用トランザクションで常に実行されます。

エラーの内容は、エラー・コードとエラー・メッセージで識別できます。現在のエラーの原因となった SQL 操作について、エラー処理スクリプトの最後の呼び出しによってアクション・コード値が返されます。

同期の一部としてエラーが発生した場合は、ユーザ名が指定されます。それ以外の場合、この値は NULL です。

特定のテーブルの操作中にエラーが発生した場合は、テーブル名が指定されます。それ以外の場合、この値は NULL です。テーブル名は、リモート・データベースでのテーブル名です。この名前に直接対応するものが統合データベース内にあるかどうかは、同期システムの設計によって異なります。

参照

- 「handle_error 接続イベント」518 ページ
- 「handle_odbc_error 接続イベント」522 ページ
- 「report_error 接続イベント」544 ページ

SQL の例

次の例は、Adaptive Server Anywhere 統合データベースで動作します。これは、同期エラーを記録するために使用されるテーブルにローを挿入します。

```
call ml_add_connection_script(
  'ver1',
  'report_odbc_error',
  'insert into sync_error(
     action_code,
     odbc_state,
     error_message,
     user_name,
     table_name )
   values( ?, ?, ?, ?, ? )'  )
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、reportODBCError という Java メソッドを report_odbc_error イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
    'ver1',
    'report_odbc_error',
    'ExamplePackage.ExampleClass.reportODBCError' )
```

次に示すのは、サンプルの Java メソッド reportODBCError です。このメソッドは、Mobile Link が提供する JDBC 接続を使用してテーブルにエラーのログを取ります。また、アクション・コードも設定します。

```
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 テーブル・イベント

機能

特定のテーブルの競合を解決する処理を定義します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	table	VARCHAR (128)

デフォルトのアクション

なし。

説明

リモート・データベースでローが更新されると、Mobile Link クライアントは元の値のコピーを保存します。クライアントは、Mobile Link 同期サーバに古い値と新しい値の両方を送信します。

Mobile Link 同期サーバは更新されたローを受信すると、元の値と統合データベース内の現在の値を比較します。比較は、upload_fetch スクリプトを使用して行われます。

アップロードされた古い値が統合データベース内の現在の値と一致しない場合は、そのローに競合が発生します。ローを更新する代わりに、Mobile Link 同期サーバは古い値と新しい値の両方を統合データベースに挿入します。古いローと新しいローは、それぞれスクリプト upload_old_row_insert と upload_new_row_insert を使用して処理されます。カーソルベースのアップロードを使用している場合、ローの処理にはそれぞれ old_row_cursor と new_row_cursor が使用されます。

値が挿入されると、Mobile Link 同期サーバは resolve_conflict スクリプトを実行します。ここで、競合を解決できます。どのスキームでも選択して実装できます。

このスクリプトは競合ごとに 1 回実行されます。

別の方法として、resolve_conflict スクリプトを定義する代わりに、end_upload_rows スクリプトまたは end_upload テーブル・スクリプトで競合解決論理を使用して、集合指向型の方法で競合を解決することもできます。

リモート・データベースのテーブルごとに、resolve_conflict スクリプトを 1 つ指定できます。

参照

- 「upload_old_row_insert テーブル・イベント」577 ページ
- 「upload_new_row_insert テーブル・イベント」575 ページ
- 「upload_update テーブル・イベント」587 ページ
- 「old_row_cursor テーブル・イベント (旧式)」539 ページ
- 「new_row_cursor テーブル・イベント (旧式)」536 ページ
- 「end_upload_rows テーブル・イベント」511 ページ

SQL の例

次の文は、Oracle インストール環境用の CustDB サンプル・アプリケーションに適した *resolve_conflict* スクリプトを定義します。このスクリプトは、ストアド・プロシージャ *ULResolveOrderConflict* を呼び出します。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、resolveConflict という Java メソッドを resolve_conflict テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
    'ver1',
    'table1',
    'resolve_conflict',
    'ExamplePackage.ExampleClass.resolveConflict' )
```

次に示すのは、サンプルの Java メソッド resolveConflict です。このメソッドは、Mobile Link が提供する JDBC 接続を使用する Java メソッドを呼び出します。また、アクション・コードも設定します。

```
public String resolveConflict( String user,
  String table)
 {  resolveRows(_syncConn, user ); }
```

synchronization_statistics 接続イベント

機能

同期統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	warnings	INTEGER
3	errors	INTEGER
4	deadlocks	INTEGER
5	synchronized_tables	INTEGER
6	connection_retries	INTEGER

デフォルトのアクション

なし。

説明

synchronization_statistics イベントを使用すると、任意のユーザと接続について、現在の同期に関する各種の統計を収集できます。最後の同期トランザクション終了時のコミット直前に、synchronization_statistics 接続スクリプトが呼び出されます。

参照

- 「download_statistics 接続イベント」472 ページ
- 「download_statistics テーブル・イベント」476 ページ
- 「upload_statistics 接続イベント」579 ページ
- 「upload_statistics テーブル・イベント」583 ページ
- 「synchronization_statistics テーブル・イベント」556 ページ

- 「time_statistics 接続イベント」559 ページ
- 「time_statistics テーブル・イベント」562 ページ
- 『Mobile Link 管理ガイド』> 「Mobile Link モニタ」

SQL の例

次の例は、同期の統計を sync_con_audit テーブルに挿入します。

```
call ml_add_connection_script(
   'ver1',
   'synchronization_statistics',
 'INSERT INTO sync_con_audit(
   ml_user, warnings, errors,
   deadlocks, synchronized_tables,
   connection_retries)
 VALUES (?,?,?,?,?,?)')
```

監査テーブルに統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、synchronizationStatisticsConnection という Java メソッドを synchronization_statistics 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
   'ver1',
   'synchronization_statistics',

'ExamplePackage.ExampleClass.synchronizationStatisticsC
onnection'
 )
```

次に示すのは、サンプルの Java メソッド synchronizationStatisticsConnection です。このメソッドは、統計の一部を Mobile Link 出力ログへ出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、SyncStats という .NET メソッドを synchronization_statistics 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script(
   'ver1',
   'synchronization_statistics',
   'TestScripts.Test.SyncStats'
 )
```

次に示すのは、サンプルの .NET メソッド SyncStats です。このメソッドは、統計の一部を Mobile Link 出力ログへ出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void SyncStats(
   string user,
   int warnings,
   int errors,
   int deadLocks,
   int syncedTables,
   int connRetries )
 {  System.Console.WriteLine( "synch statistics
    number of deadlocks: " + deadlocks ;
    return( null ); }
```

synchronization_statistics テーブル・イベント

機能

同期統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	table	VARCHAR (128)
3	warnings	INTEGER
4	errors	INTEGER

デフォルトのアクション

なし。

説明

synchronization_statistics イベントを使用すると、任意のユーザとテーブルについて、同期中に発生した警告とエラーの数を収集できます。最後の同期トランザクション終了時のコミット直前に、synchronization_statistics テーブル・スクリプトが呼び出されます。

参照

- 「download_statistics 接続イベント」472 ページ
- 「download_statistics テーブル・イベント」476 ページ
- 「upload_statistics 接続イベント」579 ページ
- 「upload_statistics テーブル・イベント」583 ページ
- 「synchronization_statistics 接続イベント」553 ページ
- 「time_statistics 接続イベント」559 ページ
- 「time_statistics テーブル・イベント」562 ページ
- 『Mobile Link 管理ガイド』> 「Mobile Link モニタ」

SQL の例

次の例は、同期の統計を sync_tab_audit テーブルに挿入します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
 'INSERT INTO sync_tab_audit ( ml_user, table,
 warnings, errors) VALUES (?,?,?,?)')
```

監査テーブルに同期統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、synchronizationStatisticsTable という Java メソッドを synchronization_statistics テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'synchronization_statistics',

'ExamplePackage.ExampleClass.synchronizationStatisticsT
able'
 )
```

次に示すのは、サンプルの Java メソッド synchronizationStatisticsTable です。このメソッドは、統計の一部を Mobile Link 出力ログへ出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、SyncTableStats という .NET メソッドを synchronization_statistics テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'synchronization_statistics',
   'TestScripts.Test.SyncTableStats'
 )
```

次に示すのは、サンプルの .NET メソッド SyncTableStats です。このメソッドは、統計の一部を Mobile Link 出力ログへ出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void SyncTableStats(
   string user,
   string table,
   int warnings,
   int errors )
 {  System.Console.WriteLine( "synch statistics for
    table: " + table + " errors: " + errors );
    return( null ); }
```

time_statistics 接続イベント

機能

ユーザ別、イベント別の時間統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	event_name	VARCHAR (128)
3	num_calls	INTEGER
4	min_time	INTEGER。ミリ秒。
5	max_time	INTEGER。ミリ秒。
6	total_time	INTEGER。ミリ秒。

デフォルトのアクション

なし。

説明

time_statistics イベントを使用すると、任意のユーザについて同期中の時間統計を収集できます。対応するスクリプトがあるイベントについてのみ、統計が収集されます。単一のイベントが数回発生する場合、スクリプトは集合データを収集します。このスクリプトは、ユーザ、イベント、テーブル間で時間を比較する場合に特に便利です。

参照

- 「time_statistics テーブル・イベント」562 ページ
- 「download_statistics 接続イベント」472 ページ
- 「download_statistics テーブル・イベント」476 ページ
- 「upload_statistics 接続イベント」579 ページ

- 「upload_statistics テーブル・イベント」583 ページ
- 「synchronization_statistics 接続イベント」553 ページ
- 「synchronization_statistics テーブル・イベント」556 ページ
- 『Mobile Link 管理ガイド』> 「Mobile Link モニタ」

SQL の例

次の例は、統計情報を time_statistics テーブルに挿入します。

```
call ml_add_connection_script(
   'ver1',
   'time_statistics',
 'INSERT INTO time_statistics (id, ml_user, table,
      event_name, num_calls, min_time, max_time,
total_time)
    VALUES (ts_id.nextval,?,?,?,?,?,?)')
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、timeStatisticsConnection という Java メソッドを time_statistics 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
   'ver1',
   'time_statistics',

'ExamplePackage.ExampleClass.timeStatisticsConnection'
)
```

次に示すのは、サンプルの Java メソッド timeStatisticsConnection です。このメソッドは、prepare_for_download イベントの統計を出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
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(
       "prepare_for_download num_calls: " + num_calls +
       "total_time: " + total_time ); } }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、TimeStats という .NET メソッドを time_statistics 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script(
   'ver1',
   'time_statistics',
   'TestScripts.Test.TimeStats'
 )
```

次に示すのは、サンプルの Java メソッド TimeStats です。このメソッドは、prepare_for_download イベントの統計を出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void TimeStats(
   string user,
   string eventName,
   int numCalls,
   int minTime,
   int maxTime,
   int totTime )
 {  if( event_name=="prepare_for_download")
    {  System.Console.WriteLine(
       "prepare_for_download num_calls: " + num_calls +
       "total_time: " + total_time ); } }
```

time_statistics テーブル・イベント

機能

時間統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
1	ml_username	VARCHAR (128)
2	table	VARCHAR (128)
3	event_name	VARCHAR (128)
4	num_calls	INTEGER
5	min_time	INTEGER。ミリ秒。
6	max_time	INTEGER。ミリ秒。
7	total_time	INTEGER。ミリ秒。

デフォルトのアクション

なし。

説明

time_statistics テーブル・イベントを使用すると、任意のユーザとテーブルについて同期中の時間統計を収集できます。対応するスクリプトがあるイベントについてのみ、統計が収集されます。単一のイベントが数回発生する場合、スクリプトは集合データを収集します。このスクリプトは、ユーザ、イベント、テーブル間で時間を比較する場合に特に便利です。

参照

- 「time_statistics 接続イベント」559 ページ
- 「download_statistics 接続イベント」472 ページ

- 「download_statistics テーブル・イベント」476 ページ
- 「upload_statistics 接続イベント」579 ページ
- 「upload_statistics テーブル・イベント」583 ページ
- 「synchronization_statistics 接続イベント」553 ページ
- 「synchronization_statistics テーブル・イベント」556 ページ
- 『Mobile Link 管理ガイド』> 「Mobile Link モニタ」

SQL の例

次の例は、統計情報を time_statistics テーブルに挿入します。

```
INSERT INTO time_statistics(
  ml_user, table, event_name, num_calls,
  min_time, max_time, total_time)
 VALUES (?,?,?,?,?,?,?)
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、timeStatisticsTable という Java メソッドを time_statistics テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'time_statistics',
   'ExamplePackage.ExampleClass.timeStatisticsTable' )
```

次に示すのは、サンプルの Java メソッド timeStatisticsTable です。このメソッドは、upload_old_row_insert イベントの統計を出力します。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、TimeTableStats という .NET メソッドを time_statistics テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'time_statistics',
   'TestScripts.Test.TimeTableStats'
 )
```

次に示すのは、サンプルの .NET メソッド TimeTableStats です。このメソッドは、upload_old_row_insert イベントの統計を出力します。

```
public void TimeTableStats(
   string user,
   string table,
   string eventName,
   int numCalls,
   int minTime,
   int maxTime,
   int totTime )
 {  if( event_name == "upload_old_row_insert")
    {  System.Console.WriteLine(
       "upload_old_row_insert num_calls: " + num_calls +
       "total_time: " + total_time ); } }
```

upload_cursor テーブル・イベント (旧式)

機能

Mobile Link 同期サーバがアップロード・ストリームの処理中に、ローの挿入、更新、削除に使用するカーソルを定義します。

アップロードを実行するには文ベースのイベントを使用してください
このスクリプトは使用されなくなりました。アップロード・ストリームを処理するには、upload_cursor イベントの代わりに、文ベースのイベント upload_delete、upload_insert、upload_update を使用してください。upload_cursor イベントのサポートは、今後のリリースでは削除される可能性があります。

パラメータ

項目	パラメータ
1	primary key 1
2	primary key 2
…	…

デフォルトのアクション

なし。

説明

Mobile Link 同期サーバは、クライアント・アプリケーションからアップロードしたローに基づいて、統合データベース内でローの挿入、更新、削除に使用するカーソルを開きます。このスクリプトには適切な SELECT 文を含めるか、またはこのスクリプトで適切な SELECT 文を含むストアド・プロシージャを呼び出してください。

パラメータは、対応するクライアント・テーブルのプライマリ・キーに含まれる各カラムの値です。これらを WHERE 句で使用して、同期処理で値に基づいてユニークなローを識別できるようにしてください。パラメータの型と順序は、example_upload_cursor スクリプトで定義されます。この順序は、リモート・データベースの対応するテーブル定義の順序と同じです。これはリファレンス・データベースから順番にコピーされたものです。

リモート・データベースのテーブルごとに、upload_cursor スクリプトを 1 つ指定できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 『Mobile Link 管理ガイド』> 「ローをアップロードするスクリプトの作成」
- 「upload_delete テーブル・イベント」568 ページ
- 「upload_insert テーブル・イベント」572 ページ
- 「upload_update テーブル・イベント」587 ページ

SQL の例

次の SELECT 文は、CustDB サンプル・アプリケーションのアップロード・カーソルを定義します。

```
call ml_add_table_script(
   'ver1',
   'table1',
   'upload_cursor',
 'SELECT cust_id, cust_name
 FROM ULCustomer
 WHERE cust_id = ?')
```

CustDB サンプル・アプリケーションの ULCustomer テーブルのプライマリ・キーは、カラム cust_id です。統合データベース内の対応するテーブルが Customer の場合は、上の文を次のように変更します。

```
SELECT cust_id, cust_name
 FROM Customer
 WHERE cust_id = ?
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadCursor という Java メソッドを upload_cursor イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'upload_cursor',
   'ExamplePackage.ExampleClass.uploadCursor' )
```

次に示すのは、サンプルの Java メソッド uploadCursor です。このメソッドは、getUploadCursor を呼び出して、アップロード・カーソルを動的に生成します。

```
public String uploadCursor()
 { return( getUploadCursor( _curTable ) ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、UploadCursor という .NET メソッドを upload_cursor イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'upload_cursor',
   'TestScripts.Test.UploadCursor' )
```

次のサンプル C# は、テンポラリ・テーブルの内容を削除します。その後、ローをテンポラリ・テーブルにアップロードする SQL を返します。

```
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 テーブル・イベント

機能

リモート・データベースから削除されたローを処理するために、Mobile Link 同期サーバがアップロード・ストリーム処理中に使用するイベントを提供します。

パラメータ

項目	パラメータ
1	primary key 1
2	primary key 2
…	…

デフォルトのアクション

なし。

説明

文ベースの upload_delete スクリプトは、リモート・データベースで削除されるローを処理します。統合データベース側で実行される動作として DELETE 文を指定できますが、他の動作も指定できます。

リモート・データベースのテーブルごとに、upload_delete スクリプトを 1 つ指定できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 「upload_insert テーブル・イベント」572 ページ
- 「upload_update テーブル・イベント」587 ページ

SQL の例

この例は Contact の例から抜粋したもので、*Samples¥MobiLink¥Contact¥build_consol.sql* にあります。リモート・データベースから削除される顧客に、非アクティブのマークを付けます。

```
call ml_add_table_script(
   'ver1',
   'table1',
   'upload_delete',
 'UPDATE Customer SET active = 0 WHERE cust_id=?')
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadDeleteTable という Java メソッドを upload_delete テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'upload_delete',
   'ExamplePackage.ExampleClass.uploadDeleteTable' )
```

次に示すのは、サンプルの Java メソッド uploadDeleteTable です。このメソッドは、UPLOAD 文を動的に生成する genUD を呼び出します。

```
public string uploadDeleteTable()
 {  return( genUD(_curTable) ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、UploadDelete という .NET メソッドを upload_delete テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'upload_delete',
   'TestScripts.Test.UploadDelete'
 )
```

次に示すのは、サンプルの .NET メソッド UploadDelete です。このメソッドは、UPLOAD 文を動的に生成する genUD を呼び出します。

```
public string UploadDelete( object pk1 )
 {  return( genUD(_curTable) ); }
```

upload_fetch テーブル・イベント

機能

アップロード・ストリームの文ベースの処理中に、Mobile Link 同期サーバが更新の競合を識別するために使用するイベントを提供します。

パラメータ

項目	パラメータ
1	primary key 1
2	primary key 2
…	…

デフォルトのアクション

なし。

説明

文ベースの upload_fetch スクリプトは、競合検出の目的で、同期テーブルからローをフェッチします。このスクリプトは、upload_update イベントに対応します。

結果セットのカラム数は、このテーブルについてリモート・データベースからアップロードされるカラムの数と一致します。返される値がアップロードされるローの更新前のイメージと一致しないと、競合が識別されます。

リモート・データベースのテーブルごとに、upload_fetch スクリプトを 1 つ指定できます。

以下のスクリプトが 1 つも定義されていない場合、このスクリプトは無視されます。upload_new_row_insert、upload_old_row_insert、resolve_conflict

参照

- 「resolve_conflict テーブル・イベント」550 ページ
- 「upload_delete テーブル・イベント」568 ページ
- 「upload_insert テーブル・イベント」572 ページ
- 「upload_update テーブル・イベント」587 ページ

SQL の例

次の SQL スクリプトは Contact の例から抜粋したもので、SQL Anywhere のインストール環境の *Samples¥MobiLink¥Contact¥build_consol.sql* にあります。リモート・データベースの Product テーブル内で更新されるローのアップロード時に発生する競合を識別するために使用されます。このスクリプトは、テーブル Product からローを選択しますが、統合データベースとリモート・データベースのスキーマによっては、2 つのテーブルの名前が一致しない場合があります。

```
call ml_add_table_script(
   'ver1',
   'table1',
   'upload_fetch',
 'SELECT id, name, size, quantity, unit_price
 FROM Product WHERE id=?')
```

Java の例

このスクリプトは有効な SQL を返します。

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadFetchTable という Java メソッドを upload_fetch テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'upload_fetch',
   'ExamplePackage.ExampleClass.uploadFetchTable' )
```

次に示すのは、サンプルの Java メソッド uploadFetchTable です。このメソッドは、UPLOAD 文を動的に生成する genUF を呼び出します。

```
public string uploadFetchTable()
 {  return( genUF(_curTable) ); }
```

upload_insert テーブル・イベント

機能

リモート・データベースに挿入されたローを処理するために、Mobile Link 同期サーバがアップロード・ストリーム処理中に使用するイベントを提供します。

パラメータ

項目	パラメータ
1	column 1
2	column 2
…	…

デフォルトのアクション

なし。

説明

文ベースの upload_insert スクリプトは、カラム値の直接挿入を実行します。

リモート・データベースのテーブルごとに、upload_insert スクリプトを 1 つ指定できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 「upload_delete テーブル・イベント」568 ページ
- 「upload_update テーブル・イベント」587 ページ
- 「upload_fetch テーブル・イベント」570 ページ

SQL の例

この例では、リモート・データベース内の Customer テーブルに対して行われた挿入を処理します。このスクリプトは、統合データベース内のテーブル Customer に値を挿入します。このテーブルの最後のカラムでは、Customer がアクティブであると識別されます。最後のカラムは、リモート・データベースには含まれません。

```
call ml_add_table_script(
   'ver1',
   'Customer',
   'upload_insert',
 'INSERT INTO Customer( cust_id, name, rep_id, active )
 VALUES ( ?, ?, ?, 1 )')
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadInsertTable という Java メソッドを upload_insert テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'upload_insert',
   'ExamplePackage.ExampleClass.uploadInsertTable' )
```

次に示すのは、サンプルの Java メソッド uploadInsertTable です。このメソッドは、UPLOAD 文を動的に生成します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
public string uploadInsertTable()
 {  return("insert into" + _curTable +
getCols(_curTable)
    + "values" + getQM(_curTable)); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、UploadInsert という .NET メソッドを upload_insert テーブル・イベント用のスクリプトとして登録します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'upload_insert',
   'TestScripts.Test.UploadInsert'
 )
```

次に示すのは、サンプルの .NET メソッド UploadInsert です。このメソッドは、ULCustomer テーブル用の UPLOAD 文を返します。

```
public static string UploadInsert()
  {
    return("INSERT INTO ULCustomer(cust_id,cust_name)
values (?,?)");
  }
```

upload_new_row_insert テーブル・イベント

機能

通常、文ベースのアップロード用の競合解決スクリプトは、リモート・データベースからアップロードされるローの古い値と新しい値にアクセスする必要があります。このイベントを使用すると、リモート・データベースからアップロードされるローの更新済みの新しい値を処理できます。

パラメータ

項目	パラメータ
ml_username	VARCHAR(128)。このパラメータはオプションです。
1	column 1
2	column 2
…	…

デフォルトのアクション

なし。

説明

Mobile Link クライアントが更新済みのローを Mobile Link 同期サーバに送信するときは、新しい値 (更新後イメージ) だけでなく、古いローの値 (更新前イメージ) のコピーも含まれています。更新前イメージが統合データベースの現在の値と一致しないと、競合が検出されます。

このイベントを使用すると、更新後イメージの値をテーブルに保存できます。このイベントは、文ベースの更新用の競合解決プロシージャ開発を支援するために使用できます。このイベントのパラメータは、対応する統合データベース・テーブルで更新が実行される前のリモート・データベースからの新しい値を保持しています。また、文ベースの強制的な競合モードで、ローを挿入するために使用されます。

このイベントのスクリプトは、resolve_conflict スクリプトが使用するテンポラリ・テーブルに新しいローを挿入する insert 文である場合がほとんどです。

リモート・データベースのテーブルごとに、upload_new_row_insert スクリプトを 1 つ指定できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 『Mobile Link 管理ガイド』> 「競合の解決」
- 「resolve_conflict テーブル・イベント」550 ページ
- 「upload_old_row_insert テーブル・イベント」577 ページ
- 「upload_update テーブル・イベント」587 ページ
- 「ユーザ名の格納」89 ページ

SQL の例

この例は、リモート・データベース内の product テーブルに対する更新を処理します。このスクリプトは、ローの新しい値をグローバルなテンポラリ・テーブル product_conflict に挿入します。このテーブルの最後のカラムでは、ローが新しいローとして識別されます。

```
call ml_add_table_script(
   'ver1',
   'table1',
   'upload_new_row_insert',
 'INSERT INTO DBA.product_conflict(
   id, name, size, quantity, unit_price, row_type )
 VALUES( ?, ?, ?, ?, ?, 'N' )')
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadNewRowInsertTable という Java メソッドを upload_new_row_insert テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'upload_new_row_insert',

'ExamplePackage.ExampleClass.uploadNewRowInsertTable'
 )
```

次に示すのは、サンプルの Java メソッド uploadNewRowInsertTable です。このメソッドは、UPLOAD 文を動的に生成します。これは Adaptive Server Anywhere 統合データベース用の構文です。

```
public string uploadNewRowInsertTable()
 {  return("insert into" + _curTable + "_new" +
    getCols(_curTable) + "values" + getQM(_curTable)); }
```

upload_old_row_insert テーブル・イベント

機能

通常、文ベースのアップロード用の競合解決スクリプトは、リモート・データベースからアップロードされるローの古い値と新しい値にアクセスする必要があります。このイベントを使用すると、リモート・データベースからアップロードされるローの古い値を処理できます。

パラメータ

項目	パラメータ
ml_username	VARCHAR(128)。このパラメータはオプションです。
1	column 1
2	column 2
…	…

デフォルトのアクション

なし。

説明

Mobile Link クライアントが更新済みのローを Mobile Link 同期サーバに送信するときは、新しい値 (更新後イメージ) だけでなく、古いローの値 (更新前イメージ) のコピーも含まれています。更新前イメージが統合データベースの現在の値と一致しないと、競合が検出されます。

このイベントを使用すると、更新前イメージの値をテーブルに保存できます。このイベントは、文ベースの更新用の競合解決プロシージャ開発を支援するために使用できます。このイベントのパラメータは、対応する統合データベース・テーブルで更新が実行される前のリモート・データベースからの古い値を保持しています。また、文ベースの強制的な競合モードで、ローを挿入するために使用されます。

このイベントのスクリプトは、resolve_conflict スクリプトが使用するテンポラリ・テーブルに古いローを挿入する insert 文である場合がほとんどです。

リモート・データベースのテーブルごとに、upload_old_row_insert スクリプトを 1 つ指定できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 『Mobile Link 管理ガイド』> 「競合の解決」
- 「resolve_conflict テーブル・イベント」550 ページ
- 「upload_new_row_insert テーブル・イベント」575 ページ
- 「upload_update テーブル・イベント」587 ページ
- 「ユーザ名の格納」89 ページ

SQL の例

この例は、リモート・データベース内の product テーブルに対する更新を処理します。このスクリプトは、ローの古い値をグローバルなテンポラリ・テーブル product_conflict に挿入します。このテーブルの最後のカラムでは、ローが古いローとして識別されます。

```
call ml_add_table_script(
   'ver1',
   'table1',
   'upload_old_row_insert',
 'insert into DBA.product_conflict(
 id, name, size, quantity, unit_price, row_type )
 values( ?, ?, ?, ?, ?, 'O' )')
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadOldRowInsertTable という Java メソッドを upload_old_row_insert テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'upload_old_row_insert',

'ExamplePackage.ExampleClass.uploadNewRowInsertTable'
 )
```

次に示すのは、サンプルの Java メソッド uploadOldRowInsertTable です。このメソッドは、UPLOAD 文を動的に生成します。

```
public string uploadOldRowInsertTable()
 {  old" + getCols(_curTable) +
    "values" + getQM(_curTable)); }
```

upload_statistics 接続イベント

機能

アップロード操作の同期統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
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

デフォルトのアクション

なし。

説明

upload_statistics イベントを使用すると、任意のユーザについてアップロードに関する統計を収集できます。アップロード・トランザクション終了時のコミット直前に、upload_statistics 接続スクリプトが呼び出されます。

参照

- 「download_statistics 接続イベント」472 ページ
- 「download_statistics テーブル・イベント」476 ページ
- 「upload_statistics テーブル・イベント」583 ページ
- 「synchronization_statistics 接続イベント」553 ページ
- 「synchronization_statistics テーブル・イベント」556 ページ
- 「time_statistics 接続イベント」559 ページ
- 「time_statistics テーブル・イベント」562 ページ
- 『Mobile Link 管理ガイド』> 「Mobile Link モニタ」

SQL の例

次の例は、アップロード操作での同期の統計を upload_summary_audit テーブルに挿入します。

```
call ml_add_connection_script(
    'ver1',
    'upload_statistics',
 'INSERT INTO upload_summary_audit (
  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 (?,?,?,?,?,?,?,?,?,?,?,?,?,?)'
```

監査テーブルに統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadStatisticsConnection という Java メソッドを upload_statistics 接続イベント用のスクリプトとして登録します。

```
call ml_add_java_connection_script(
    'ver1',
    'upload_statistics',

'ExamplePackage.ExampleClass.uploadStatisticsConnection
' )
```

次に示すのは、サンプルの Java メソッド uploadStatisticsConnection です。このメソッドは、統計の一部を Mobile Link 出力ログへ出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、UploadStats という .NET メソッドを upload_statistics 接続イベント用のスクリプトとして登録します。

```
call ml_add_dnet_connection_script(
   'ver1',
   'upload_statistics',
   'TestScripts.Test.UploadStats'
 )
```

次に示すのは、サンプルの .NET メソッド UploadStats です。このメソッドは、統計の一部を Mobile Link 出力ログへ出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
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)
 {  System.Console.WriteLine( "updated rows: " +
    updatedRows ); }
```

upload_statistics テーブル・イベント

機能

特定のテーブルに対するアップロード操作について、同期統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」を参照してください。

イベント・パラメータは、後続のパラメータを指定しない場合にのみオプションになります。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

項目	パラメータ	説明
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

項目	パラメータ	説明
15	deadlocks	INTEGER

デフォルトのアクション

なし。

説明

upload_statistics イベントを使用すると、任意のユーザについて、任意のテーブルに適用される同期発生の重要な統計を収集できます。アップロード・トランザクション終了時のコミット直前に、upload_statistics テーブル・スクリプトが呼び出されます。

参照

- 「download_statistics 接続イベント」472 ページ
- 「upload_statistics 接続イベント」579 ページ
- 「upload_statistics テーブル・イベント」583 ページ
- 「synchronization_statistics 接続イベント」553 ページ
- 「synchronization_statistics テーブル・イベント」556 ページ
- 「time_statistics 接続イベント」559 ページ
- 「time_statistics テーブル・イベント」562 ページ
- 『Mobile Link 管理ガイド』> 「Mobile Link モニタ」

SQL の例

次の例は、アップロードの統計を追跡するために使用されるテーブルにローを挿入します。

```
call ml_add_connection_script(
  'ver1',
  'upload_statistics',
  'INSERT INTO my_upload_statistics
     ( user_name, table_name, num_warnings, num_errors,
       inserted_rows, deleted_rows, updated_rows,
       conflicted_inserts, conflicted_deletes,
       conflicted_updates, ignored_inserts,
       ignored_deletes, ignored_updates, bytes,
       deadlocks )
    VALUES( ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?
)'  )
```

次の例は、Oracle 統合データベースで動作します。

```
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,
 ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
```

監査テーブルに統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadStatisticsTable という Java メソッドを upload_statistics テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'upload_statistics',
   'ExamplePackage.ExampleClass.uploadStatisticsTable'
)
```

次に示すのは、サンプルの Java メソッド uploadStatisticsTable です。このメソッドは、統計の一部を Mobile Link 出力ログへ出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
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 の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、UploadTableStats という .NET メソッドを upload_statistics テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'upload_statistics',
   'TestScripts.Test.UploadTableStats'
 )
```

次に示すのは、サンプルの .NET メソッド uploadStatisticsTable です。このメソッドは、統計の一部を Mobile Link 出力ログへ出力します (このメソッドは開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
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 )
 {  System.Console.WriteLine( "updated rows: " +
    updatedRows ); }
```

upload_update テーブル・イベント

機能

リモート・データベースで更新されるローを処理するために、Mobile Link 同期サーバがアップロード・ストリーム処理中に使用するイベントを提供します。

パラメータ

句	パラメータ
SET	non-primary key column 1
	non-primary key column 2
	…
WHERE	primary key 1
	primary key 2
	…

デフォルトのアクション

なし。

説明

文ベースの upload_update スクリプトは、UPLOAD 文で指定されたカラム値の直接更新を実行できます。

WHERE 句には、同期するプライマリ・キー・カラムをすべて含めます。SET 句には、同期する非プライマリ・キー・カラムをすべて含めます。

SET 句にはテーブルにある非プライマリ・キー・カラムをすべて指定します。Mobile Link は、適切な数のカラム値を送信します。また、WHERE 句には必要な数だけプライマリ・キーを指定できますが、ここですべてを指定してください。Mobile Link は適切な値を送信し、これらのカラム値とプライマリ・キー値を、スキーマの Mobile Link レポートに表示される順に送信します。-vh オプションを使用すると、このテーブル・スキーマのカラムの順序を指定できます。

リモート・データベースのテーブルごとに、upload_update スクリプトを 1 つ指定できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 「upload_delete テーブル・イベント」568 ページ
- 「upload_fetch テーブル・イベント」570 ページ
- 「upload_insert テーブル・イベント」572 ページ

SQL の例

この例は、リモート・データベース内の Customer テーブルに対する更新を処理します。このスクリプトは、統合データベース内のテーブル Customer 内の値を更新します。

```
call ml_add_table_script(
   'ver1',
   'table1',
   'upload_update',
 'UPDATE Customer SET name=?, rep_id=? WHERE
cust_id=?')
```

Java の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadUpdateTable という Java メソッドを upload_update テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_java_table_script(
   'ver1',
   'table1',
   'upload_update',
   'ExamplePackage.ExampleClass.uploadUpdateTable' )
```

次に示すのは、サンプルの Java メソッド uploadUpdateTable です。このメソッドは、UPLOAD 文を動的に生成する genUU というメソッドを呼び出します。

```
public string uploadUpdateTable()
 {  return( genUU(_curTable) ); }
```

.NET の例

次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、UploadUpdate という .NET メソッドを upload_update テーブル・イベント用のスクリプトとして登録します。

```
call ml_add_dnet_table_script(
   'ver1',
   'table1',
   'upload_update',
   'TestScripts.Test.UploadUpdate'
 )
```

次に示すのは、UploadUpdate 呼び出し用の C# シグニチャです。

```
public string UploadUpdate()
```

第 3 部　Mobile Link リファレンス

第 3 部では、Mobile Link リファレンス情報を示します。

第 16 章

ストアド・プロシージャ

この章の内容

この章では、Mobile Link の事前に定義されたストアド・プロシージャについて説明します。

スクリプトを追加または削除するためのストアド・プロシージャ

同期スクリプトは、統合データベース内のシステム・テーブルに追加しないと使用できません。次のストアド・プロシージャを使用して、同期スクリプトを統合データベースに追加します。これらのストアド・プロシージャを使用してスクリプトを削除することもできます。

注意

- ストアド・プロシージャを使用してスクリプトを追加する場合、スクリプトは 1 つの文字列になります。スクリプト内の文字列はすべて、エスケープする必要があります。Adaptive Server Anywhere の場合、文字列を終了しないように各引用符 (') を 2 つ重ねる必要があります。
- Adaptive Server Enterprise 11.5 以前のバージョンに 255 バイトを超えるスクリプトを追加する場合は、ストアド・プロシージャを使用できません。長いスクリプトを定義する場合、Sybase Central を使用するか、直接挿入します。
- バージョン 6 より前の IBM DB2 では、カラム名とその他の識別子は 18 文字までしかサポートされないため、名前がトランケートされます。たとえば、ml_add_connection_script は ml_add_connection_ に短縮されます。

ml_add_connection_script

機能

このストアド・プロシージャを使用して、SQL 接続スクリプトを統合データベースに追加するか、または統合データベースから削除します。

パラメータ

項目	説明	備考
1	version name	VARCHAR (128)
2	event name	VARCHAR (128)

項目	説明	備考
3	script contents	Adaptive Server Anywhere と MS SQL Server の場合は TEXT。ASE の場合は VARCHAR(16384)。バージョン 12.5 より前の ASE の場合は VARCHAR(255)。DB2 の場合は VARCHAR(4000)。Oracle の場合は VARCHAR。

説明

接続スクリプトを削除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトを追加すると、スクリプトが ml_script テーブルに挿入され、このスクリプトを指定のイベントとスクリプト・バージョンに関連付ける適切な参照が定義されます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

参照

- 『Mobile Link 管理ガイド』> 「統合データベースでのスクリプトの追加と削除」
- 「ml_add_table_script」595 ページ
- 「ml_add_dnet_connection_script」597 ページ
- 「ml_add_dnet_table_script」598 ページ
- 「ml_add_java_connection_script」599 ページ
- 「ml_add_java_table_script」600 ページ

例

次の文は、begin_synchronization イベントと関連付けられた接続スクリプトを Adaptive Server Anywhere 統合データベースのスクリプト・バージョン custdb に追加します。追加されるスクリプト自体は、@EmployeeID 変数を設定する単一の文です。

```
call ml_add_connection_script( 'custdb',
     'begin_synchronization',
     'set @EmployeeID = ?' )
```

ml_add_table_script

機能

このストアド・プロシージャを使用して、SQL テーブル・スクリプトを統合データベースに追加するか、または統合データベースから削除します。

パラメータ

項目	説明	備考
1	version name	VARCHAR (128)
2	table name	VARCHAR (128)
3	event name	VARCHAR (128)
4	script contents	Adaptive Server Anywhere と MS SQL Server の場合は TEXT。ASE の場合は VARCHAR(16384)。バージョン 12.5 より前の ASE の場合は VARCHAR(255)。DB2 の場合は VARCHAR(4000)。Oracle の場合は VARCHAR。

説明

テーブル・スクリプトを削除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトを追加すると、スクリプトが ml_script テーブルに挿入され、このスクリプトを指定のテーブル、イベント、スクリプト・バージョンに関連付ける適切な参照が定義されます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

参照

- 『Mobile Link 管理ガイド』> 「統合データベースでのスクリプトの追加と削除」
- 「ml_add_connection_script」594 ページ
- 「ml_add_dnet_connection_script」597 ページ
- 「ml_add_dnet_table_script」598 ページ
- 「ml_add_java_connection_script」599 ページ
- 「ml_add_java_table_script」600 ページ

例

次のコマンドは、Customer テーブルの upload_insert イベントに対応するテーブル・スクリプトを追加します。

```
call ml_add_table_script( 'default', 'Customer',
'upload_insert',
  'INSERT INTO Customer( cust_id, name, rep_id, active )
       VALUES ( ?, ?, ?, 1 )' )
```

ml_add_dnet_connection_script

機能

このストアド・プロシージャを使用して、.NET 接続スクリプトを統合データベースに追加するか、または統合データベースから削除します。

パラメータ

項目	説明	備考
1	version name	VARCHAR (128)
2	event name	VARCHAR (128)
3	script contents	Adaptive Server Anywhere と MS SQL Server の場合は TEXT。ASE の場合は VARCHAR(16384)。バージョン 12.5 より前の ASE の場合は VARCHAR(255)。DB2 の場合は VARCHAR(4000)。Oracle の場合は VARCHAR。

説明

接続スクリプトを削除するには、スクリプトの内容パラメータを NULL に設定します。

script 値は、たとえば MyClass.MyMethod などの Mobile Link 同期サーバのクラスパスに含まれるクラス内のパブリック・メソッドです。

スクリプトを追加すると、メソッドが指定のイベントとスクリプト・バージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

参照

- 『Mobile Link 管理ガイド』> 「統合データベースでのスクリプトの追加と削除」
- 「ml_add_dnet_table_script」598 ページ
- 「ml_add_connection_script」594 ページ
- 「ml_add_table_script」595 ページ
- 「ml_add_java_table_script」600 ページ
- 『Mobile Link 管理ガイド』> 「メソッド」

例

次の例は、ExampleClass クラスの beginDownloadConnection メソッドを begin_download イベントに割り当てます。

```
call ml_add_dnet_connection_script( 'ver1',
 'begin_download',
 'ExamplePackage.ExampleClass.beginDownloadConnection')
```

ml_add_dnet_table_script

機能

このストアド・プロシージャを使用して、.NET テーブル・スクリプトを統合データベースに追加するか、または統合データベースから削除します。

パラメータ

項目	説明	備考
1	version name	VARCHAR (128)
2	table name	VARCHAR (128)
3	event name	VARCHAR (128)
4	script contents	Adaptive Server Anywhere と MS SQL Server の場合は TEXT。ASE の場合は VARCHAR(16384)。バージョン 12.5 より前の ASE の場合は VARCHAR(255)。DB2 の場合は VARCHAR(4000)。Oracle の場合は VARCHAR。

説明

接続スクリプトを削除するには、スクリプトの内容パラメータを NULL に設定します。

script 値は、たとえば MyClass.MyMethod などの Mobile Link 同期サーバのクラスパスに含まれるクラス内のパブリック・メソッドです。

スクリプトを追加すると、メソッドが指定のテーブル、イベント、スクリプト・バージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

参照

- 『Mobile Link 管理ガイド』> 「統合データベースでのスクリプトの追加と削除」
- 「ml_add_dnet_connection_script」597 ページ
- 「ml_add_connection_script」594 ページ
- 「ml_add_table_script」595 ページ

- 「ml_add_java_connection_script」599 ページ
- 『Mobile Link 管理ガイド』> 「メソッド」

例

次の例では、EgClass クラスの empDownloadCursor メソッドを emp テーブルの download_cursor イベントに割り当てます。

```
call ml_add_dnet_table_script( 'ver1', 'emp',

'download_cursor',EgPackage.EgClass.empDownloadCursor')
```

ml_add_java_connection_script

機能

このストアド・プロシージャを使用して、Java 接続スクリプトを統合データベースに追加するか、または統合データベースから削除します。

パラメータ

項目	説明	備考
1	version name	VARCHAR (128)
2	event name	VARCHAR (128)
3	script contents	Adaptive Server Anywhere と MS SQL Server の場合は TEXT。ASE の場合は VARCHAR(16384)。バージョン 12.5 より前の ASE の場合は VARCHAR(255)。DB2 の場合は VARCHAR(4000)。Oracle の場合は VARCHAR。

説明

接続スクリプトを削除するには、スクリプトの内容パラメータを NULL に設定します。

script 値は、たとえば MyClass.MyMethod などの Mobile Link 同期サーバのクラスパスに含まれるクラス内のパブリック・メソッドです。

スクリプトを追加すると、メソッドが指定のイベントとスクリプト・バージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

参照

- 『Mobile Link 管理ガイド』> 「統合データベースでのスクリプトの追加と削除」
- 「ml_add_connection_script」594 ページ
- 「ml_add_table_script」595 ページ
- 「ml_add_dnet_connection_script」597 ページ
- 「ml_add_dnet_table_script」598 ページ
- 「ml_add_java_table_script」600 ページ
- 『Mobile Link 管理ガイド』> 「メソッド」

例

次の例は、*Samples¥MobiLink¥JavaAuthentication* のサンプルからの抜粋です。この例では、CustEmpScripts クラスの endConnection メソッドを end_connection イベントに割り当てます。

```
call ml_add_java_connection_script( 'ver1',
 'end_connection',
 'CustEmpScripts.endConnection')
```

ml_add_java_table_script

機能

このストアド・プロシージャを使用して、Java テーブル・スクリプトを統合データベースに追加するか、または統合データベースから削除します。

パラメータ

項目	説明	備考
1	version name	VARCHAR (128)
2	table name	VARCHAR (128)
3	event name	VARCHAR (128)
4	script	Adaptive Server Anywhere と MS SQL Server の場合は TEXT。ASE の場合は VARCHAR(16384)。バージョン 12.5 より前の ASE の場合は VARCHAR(255)。DB2 の場合は VARCHAR(4000)。Oracle の場合は VARCHAR。

説明

接続スクリプトを削除するには、スクリプト・パラメータを NULL に設定します。

script 値は、たとえば MyClass.MyMethod などの Mobile Link 同期サーバのクラスパスに含まれるクラス内のパブリック・メソッドです。

スクリプトを追加すると、メソッドが指定のテーブル、イベント、スクリプト・バージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

参照

- 『Mobile Link 管理ガイド』> 「統合データベースでのスクリプトの追加と削除」
- 「ml_add_connection_script」594 ページ
- 「ml_add_table_script」595 ページ
- 「ml_add_dnet_connection_script」597 ページ
- 「ml_add_dnet_table_script」598 ページ
- 「ml_add_java_connection_script」599 ページ
- 『Mobile Link 管理ガイド』> 「メソッド」

例

次の例は、*Samples¥MobiLink¥JavaAuthentication* のサンプルからの抜粋です。この例では、CustEmpScripts クラスの empDownloadCursor メソッドを emp テーブルの download_cursor イベントに割り当てます。

```
call ml_add_java_table_script( 'ver1', 'emp',
 'download_cursor','CustEmpScripts.empDownloadCursor')
```

プロパティを追加または削除するためのストアド・プロシージャ

次のストアド・プロシージャを使用して、プロパティを統合データベースに追加します。これらのストアド・プロシージャを使用してプロパティを削除することもできます。

ml_add_property

機能

このストアド・プロシージャを使用して、Mobile Link のプロパティを追加または削除します。このストアド・プロシージャは、Mobile Link システム・テーブル ml_property のローを変更します。

パラメータ

項目	説明	備考
1	component_name	VARCHAR (128)
2	prop_set_name	VARCHAR (128)
3	prop_name	VARCHAR (128)
4	prop_value	LONG VARCHAR

説明

- **component_name** このパラメータが、**ScriptVersion** または **SIS** であるレコードを作成できます。

  スクリプト・バージョンごとにプロパティを保存するには、このパラメータを **ScriptVersion** に設定します。

  サーバによって開始される同期のプロパティの場合は、このパラメータを **SIS** に設定します。詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「プロパティの設定」を参照してください。

- **prop_set_name** component_name が **ScriptVersion** である場合、このパラメータはスクリプト・バージョンの名前です。

component_name が **SIS** である場合、このパラメータはプロパティを設定している Notifier、ゲートウェイ、または Carrier の名前です。

- **prop_name** このパラメータは、プロパティの名前です。

  component_name が **ScriptVersion** である場合、このパラメータは定義するプロパティです。これらのプロパティは、次のメソッドを使用して参照します。

  - DBConnectionContext からの場合：getVersion と getProperties
  - ServerContext からの場合：getPropertiesByVersion、getProperties、getPropertySetNames

  詳細については、『Mobile Link 管理ガイド』> 「Mobile Link Java API リファレンス」および『Mobile Link 管理ガイド』> 「Mobile Link .NET API リファレンス」を参照してください。

  component_name が **SIS** である場合、このパラメータは Notifier、ゲートウェイ、または Carrier のプロパティです。プロパティのリストについては、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「Mobile Link 通知プロパティ」を参照してください。

- **prop_value** このパラメータは、プロパティの値です。

  プロパティを削除するには、prop_value パラメータを NULL に設定します。

サーバ起動同期

サーバ起動同期では、ml_add_property ストアド・プロシージャを使用すると、Notifier、ゲートウェイ、Carrier のプロパティを設定できます。

たとえば、x という SMTP ゲートウェイのプロパティ `server=mailserver1` を追加するには、次のように入力します。

```
ml_add_property(
'SIS','SMTP(x)','server','mailserver1');
```

冗長なプロパティがすべての Notifier とゲートウェイに適用されるため、特定の prop_set_name を指定することはできません。冗長性の設定を変更するには、次のように prop_set_name を空のままにします。

```
ml_add_property( 'SIS','','verbosity',2);
```

プロパティの設定の詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「プロパティの設定」を参照してください。サーバ起動同期のプロパティの完全なリストについては、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「Mobile Link 通知プロパティ」を参照してください。

スクリプト・バージョン

通常の Mobile Link 同期では、このストアド・プロシージャを使用して、プロパティをスクリプト・バージョンに関連付けることができます。この場合、component_name を **ScriptVersion** に設定します。任意のプロパティを指定し、Java クラスまたは .NET クラスを使用してプロパティにアクセスできます。

たとえば、LDAP サーバを MyVersion というスクリプト・バージョンに関連付けるには、次のように入力します。

```
ml_add_property( 'ScriptVersion','MyVersion','ldap-
server','MyServer' )
```

詳細については、『Mobile Link 管理ガイド』> 「Mobile Link Java API リファレンス」および『Mobile Link 管理ガイド』> 「Mobile Link .NET API リファレンス」で、次のメソッドを参照してください。

- DBConnectionContext からの場合：getVersion と getProperties
- ServerContext からの場合：getPropertiesByVersion、getProperties、getPropertySetNames

参照

「ml_property」628 ページ

第 17 章

ユーティリティ

この章の内容

この章では、Mobile Link 同期サーバのユーティリティについて説明します。

Mobile Link クライアントのユーティリティについては、『Mobile Link クライアント』> 「ユーティリティ」を参照してください。

他の Adaptive Server Anywhere ユーティリティについては、『ASA データベース管理ガイド』> 「データベース管理ユーティリティ」を参照してください。

Mobile Link 停止ユーティリティ

ローカル・マシン上の Mobile Link 同期サーバを停止します。

構文

dbmlstop [*options*] [*server-name*]

オプション	説明
@data	指定された環境変数または設定ファイルからオプションを読み込む。
-f	強制シャットダウン。ハード・シャットダウンが機能しない場合に使用します。
-h	ハード・シャットダウン。Mobile Link がすべての同期を停止して終了します。リモートによっては、エラーをレポートする場合があります。
-q	クワイエット・モード。バナーを表示しません。
-t *time*	ソフト・シャットダウン。ただし、指定の時間が経過したらハード・シャットダウンを実行します。*time* は数字で、後ろに D、H、M、または S (日、時間、分、秒) が続きます。たとえば、`-t 10m` と指定すると、10 分後または現在の同期が完了した時点 (早い方) で、サーバがシャットダウンされます。D、H、M、S の大文字と小文字は区別されません。
-w	コマンドから戻る前に、Mobile Link 同期サーバがシャットダウンされるのを待機する。

パラメータ

@data　このオプションを使用すると、指定された環境変数または設定ファイルからオプションを読み込むことができます。同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。

設定ファイルの詳細については、『ASA データベース管理ガイド』> 「設定ファイルの使用」を参照してください。

設定ファイル内のパスワードやその他の情報を保護する場合は、ファイル非表示ユーティリティを使用すると、設定ファイルの内容を難読化できます。

『ASA データベース管理ガイド』> 「dbfhide コマンド・ライン・ユーティリティを使用してファイル内容を隠す」を参照してください。

Server-name　-zs オプションで Mobile Link 同期サーバを起動した場合は、同じサーバ名を指定してシャットダウンしてください。

詳細については、「-zs オプション」278 ページを参照してください。

説明

デフォルト (-f、-h、-t のいずれも指定されていない場合) では、dbmlstop によってソフト・シャットダウンが行われます。

- **ソフト・シャットダウン** これは、現在の同期が完了すると、Mobile Link 同期サーバが新しい接続を受け入れずに終了することを意味します。
- **ハード・シャットダウン** これは、Mobile Link 同期サーバがすべての同期を停止して終了することを意味します。リモートによっては、エラーをレポートする場合があります。

Mobile Link ユーザ認証ユーティリティ

統合データベース側で Mobile Link ユーザを登録します。Adaptive Server Anywhere リモートの場合は、CREATE SYNCHRONIZATION USER 文を使用して、リモート・データベース側でユーザを事前に作成しておきます。

構文

dbmluser [*options*] **-c** "*connection-string*"
{ **-f** *file* | **-u** *user* [**-p** *password*] }

オプション	説明
@data	指定された環境変数または設定ファイルからオプションを読み込む。
-c "*keyword*=*value*;…"	データベース接続パラメータを指定する。ODBC データ・ソースを使用して統合データベースに接続するには、接続文字列にユーティリティのパーミッションを指定します。このパラメータは必須です。
-d	-f または -u で指定したユーザ名を削除する。
-dl	このスイッチを指定すると、ウィンドウかコマンド・ラインに、またログ・ファイル内にも、メッセージが表示される。
-f *filename*	指定したファイルから、ユーザ名とパスワードを読み込む。ファイルは、1 行ごとにユーザ名とパスワードが 1 つずつ、空白スペースで区切って指定されているものを使用します。-f または -u を指定してください。
-o *filename*	指定したファイルに出力メッセージのログを取る。

オプション	説明
-os *size*	出力ファイルのサイズを制限する。*size* には、出力メッセージのログを取るファイルの最大サイズを、バイト単位で指定します。単位をキロバイトまたはメガバイトで指定するには、それぞれサフィックス k または m を使用します。デフォルトでは、サイズは無制限となります。最小のサイズ制限は 10 KB です。
-ot *filename*	ログ・ファイルをトランケートし、このファイルに出力メッセージを追加する。デフォルトでは、画面に出力を送信します。
-p *password*	ユーザと関連付けるパスワード。このオプションは、-u を指定した場合のみ使用できます。
-pc *collation-id*	ユーザ名とパスワードの文字セット変換用のデータベース照合 ID を指定する。ここには、『ASA データベース管理ガイド』>「初期化ユーティリティのオプション」にリストされている Adaptive Server Anywhere 照合ラベルの 1 つを指定します。このオプションは、ファイルから読み込まれるユーザ名とパスワードが、ロケールで決定されるデフォルトの文字セットとは異なる文字セットでエンコードされている場合に必要です。
-u *ml_username*	追加するユーザ名を指定する (削除する場合は、-d を指定する)。1 行のコマンド・ラインで指定できるユーザは 1 人だけです。パスワードを使用している場合は、このオプションを -p とともに使用します。-f または -u を指定してください。

オプション

@data　このオプションを使用すると、指定された環境変数または設定ファイルからオプションを読み込むことができます。指定された名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。

設定ファイルの詳細については、『ASA データベース管理ガイド』> 「設定ファイルの使用」を参照してください。

設定ファイル内のパスワードやその他の情報を保護する場合は、ファイル非表示ユーティリティを使用すると、設定ファイルの内容を難読化できます。

『ASA データベース管理ガイド』> 「dbfhide コマンド・ライン・ユーティリティを使用してファイル内容を隠す」を参照してください。

説明

ユーザとパスワードのペアを指定すると、dbmluser ユーティリティはまずそのユーザを追加しようとします。ユーザをすでに統合データベースに追加してある場合は、そのユーザのパスワードを更新します。

ユーザ名を統合データベースに登録するには、別の方法も使用できます。

- Sybase Central を使用する。
- dbmlsrv9 で -zu+ コマンド・ライン・オプションを指定する。この場合、最初に同期するときに、統合データベースに追加されていない既存の Mobile Link ユーザが追加されます。

追加されるのは、リモート・データベースにすでに存在する Mobile Link ユーザです。リモート・データベース側でユーザを追加する場合、次のオプションがあります。

- Adaptive Server Anywhere リモートの場合、CREATE SYNCHRONIZATION USER を使用して名前を設定し、そのユーザ名で同期する。
- Ultra Light リモートの場合は、ul_synch_info 構造体の user_name フィールドを使用するか、Java で、ULSynchInfo クラスの SetUserName() メソッドを使用してから同期する。

参照

- 『Mobile Link クライアント』> 「Mobile Link ユーザの認証」

証明書読み込みユーティリティ

readcert ユーティリティを使用して、証明書内の値を表示し、証明書のチェーンを検証します。

構文

readcert *certificate-name*

説明

指定できる証明書は楕円曲線または RSA です。

ECC_TLS または RSA_TLS 同期ストリームを介して同期が発生すると、Mobile Link 同期サーバは、証明書をクライアントに送信します。送信する証明書はエンティティの署名付き証明書だけでなく、自己署名ルート証明書にまで至ります。クライアントは、チェーンが有効でチェーン内のルート証明書が信用できることを確認します。

このユーティリティは、X509 認証証明書をスキャンし、フィールド値を表示します。その後、証明書のチェーンが有効であることを確認します。チェーン内のいずれかの証明書の期限が切れていたり、順序が正しくなかったり、なかったりした場合、検証エラーが表示されます。

参照

- 「証明書生成ユーティリティ」612 ページ

証明書生成ユーティリティ

gencert ユーティリティを使用して、新しい楕円曲線証明書または RSA 証明書を作成するか、事前に生成した証明書要求に署名します。

構文

gencert [**-c** | **-s**] [**-r** | **-q**]

オプション	説明
-c	他の証明書の署名に使用できる証明書を指定する。**-r** と共に使用すると、エンタープライズ・ルート証明書が生成されます。
-s	サーバ識別証明書を指定する。サーバ識別情報は、サーバのプライベート・キーとパブリック証明書を組み合わせたものです。Mobile Link 同期サーバを起動するとき (Mobile Link トランスポート・レイヤ・セキュリティの場合)、またはデータベース・サーバを起動するとき (Adaptive Server Anywhere クライアント・サーバのトランスポート・レイヤ・セキュリティの場合)、サーバ識別証明書が参照されます。**-r** と共に使用すると、自己署名証明書が生成されます。
-r	自己署名ルート証明書を指定する。**-s** と共に使用すると、自己署名証明書が作成されます。**-c** と共に使用すると、他の証明書の署名に使用できるエンタープライズ・ルート証明書が作成されます。他のオプションは使用せずに **gencert -r** を指定すると、サーバ証明書またはエンタープライズ・ルートとして使用できる証明書が作成されます。このオプションは、**-q** との互換性はありません。

オプション	説明
-q *request-file*	事前に生成した証明書要求に署名する。**-s** と共に使用すると、サーバ証明書が作成されます。**-c** と共に使用すると、他の証明書の署名に使用できるエンタープライズ・ルート証明書が作成されます。他のオプションは使用せずに **gencert -q** を指定すると、サーバ証明書またはエンタープライズ・ルートとして使用できる証明書が作成されます。**-q** オプションは、**-r** オプションとの互換性はありません。

-s または -c を指定しない場合、証明書には両方のオプションで提供される機能が含まれます。つまり、この証明書は、他の証明書の署名で使用することも、サーバ証明書として直接使用することもできます。

説明

gencert ユーティリティを使用すると、Mobile Link 同期または Adaptive Server Anywhere クライアント・サーバ通信を保護するために使用される、信頼できるパブリック証明書、プライベート・キー、サーバ証明書を生成できます。このユーティリティは、各種のセキュリティ設定に対して、X509 証明書 (標準の証明書形式) を作成します。

gencert ユーティリティでは、次の情報の入力が要求されます。

- **暗号化** 楕円曲線暗号化または RSA 暗号化の選択が要求されます。楕円曲線証明書を生成する場合は、gencert は楕円曲線キー・ペアを生成します。RSA 証明書を生成する場合、gencert は 512 ～ 2048 の範囲でキー・サイズを指定するように要求し、RSA を使用して証明書を作成します (一般に、キーが長くなるほど暗号化は強力になりますが、処理時間が長くなります)。

  どちらの暗号化を選択する場合も、サーバとクライアントを起動するときに、その暗号化を指定してください。ECC 証明書の場合は ecc_tls を指定し、RSA 証明書の場合は rsa_tls または rsa_tls_fips を指定します。

- **国、都道府県、地方** これらの値は、一般的な証明書識別情報を提供します。グローバル署名証明書を使用する予定の場合は、サード・パーティの認証局によって地方フィールドも要求されます。

  認証局の使用の詳細については、「グローバル署名証明書」214 ページを参照してください。

- **組織、組織単位、通称** これらのフィールドによって、クライアントが正しい証明書を認証していることがさらに保証されます。クライアント側では、これらは certificate_company、certificate_unit、certificate_name というプロトコル・オプションにそれぞれ対応します。

  「証明書フィールドの確認」223 ページを参照してください。

- **シリアル番号** 証明書のシリアル番号の選択が要求されます。シリアル番号は英数字です。

- **証明書の有効期間 (年数)** エンタープライズ・ルート証明書の有効期間 (年数) が要求されます。エンタープライズ・ルート証明書が失効すると、署名されているすべてのサーバ証明書も無効になります。指定した期間が過ぎると、エンタープライズ・ルート、各サーバ証明書、クライアントに配布されるパブリック証明書を再生成する必要があります。

- **プライベート・キーを保護するためのパスワードを入力してください。** certificate_password プロトコル・オプションで指定するパスワードです。

- **証明書を保存するファイル名を入力してください。** 証明書のファイル名とロケーションを選択します。

- **プライベート・キーを保存するファイル名を入力してください。** プライベート・キーのファイル名とロケーションを選択します。

- **サーバ ID を保存するファイル名を入力してください。** 自己署名ルート証明書を作成している場合のみ、この情報の入力が要求されます。サーバ証明書のファイル名とロケーションを選択します。

参照

Mobile Link 同期を保護するための証明書の使用については、『Mobile Link 管理ガイド』> 「Mobile Link トランスポート・レイヤ・セキュリティ」を参照してください。

Adaptive Server Anywhere RDBMS を保護するための証明書の使用については、『SQL Anywhere Studio セキュリティ・ガイド』> 「Adaptive Server Anywhere トランスポート・レイヤ・セキュリティ」を参照してください。

- 「証明書読み込みユーティリティ」611 ページ
- 『ASA データベース管理ガイド』> 「-ec サーバ・オプション」
- 『ASA データベース管理ガイド』> 「Encryption 接続パラメータ [ENC]」
- 「FIPS 140-2 承認」207 ページ

例

- 次の例では、gencert -r オプションを使用して、RSA 自己署名サーバ証明書を生成します。

```
> gencert -r
 Certificate Generation Tool
 Choose certificate type ((R)SA or (E)CC): R
 Enter key length (512-2048): 1024
 Generating key pair...
 Country: CA
 State/Province: ON
 Locality: Waterloo
 Organization: Sybase, Inc.
 Organizational Unit: IAS
 Common Name: MobiLink
 Serial Number: 123
 Certificate valid for how many years: 12
 Enter password to protect private key: mypwd123
 Enter file path to save certificate:
public_cert.crt
 Enter file path to save private key:
server_key.pri
 Enter file path to save server identity:
server_cert.crt
```

自己署名パブリック証明書 *public_cert.crt* をクライアントに配布します。サーバ証明書 *server_cert.crt* にはサーバ証明書とプライベート・キーが含まれ、Mobile Link 同期サーバ (Mobile Link トランスポート・レイヤ・セキュリティの場合) またはデータ

ベース・サーバ (Adaptive Server Anywhere クライアント・サーバのトランスポート・レイヤ・セキュリティの場合) で保存されます。

- 次の例では、gencert -r および -c オプションを使用して、エンタープライズ・ルート証明書を生成します。

```
> gencert -r -c
 Certificate Generation Tool
 Choose certificate type ((R)SA or (E)CC): E
 Generating key pair...
 Country: CA
 State/Province: ON
 Locality: Waterloo
 Organization: Sybase, Inc.
 Organizational Unit: IAS
 Common Name: MobiLink
 Serial Number: 123
 Certificate valid for how many years: 15
 Enter password to protect private key: mypwd123
 Enter file path to save certificate:
public_root_cert.crt
 Enter file path to save private key: root_key.pri
```

パブリック・エンタープライズ・ルート証明書 *public_root_cert.crt* が、クライアントに配布されます。エンタープライズ・プライベート・キー *root_key.pri* を、安全なロケーションに保存します。生成されたファイルを使用して、サーバ証明書に署名できます。

- 次の例では、gencert -s オプションを使用して、署名されたサーバ証明書を作成します。

```
> gencert -s
 Certificate Generation Tool
 Choose certificate type ((R)SA or (E)CC): E
 Generating key pair...
 Country: CA
 State/Province: ON
 Locality: Waterloo
 Organization: Sybase, Inc.
 Organizational Unit: IAS
 Common Name: MobiLink
 Serial Number: 123
 Certificate valid for how many years: 7
```

```
 Enter file path of signer's certificate:
public_root_cert.crt
 Enter file path of signer's private key:
root_key.pri
 Enter password for signer's private key: mypwd123
 Enter password to protect private key:
SERV1privatekeypwd
 Enter file path to save server identity:
serv1_cert.crt
```

証明書の署名として、前の例で作成されたエンタープライズ・ルート証明書を使用します。Mobile Link 同期サーバまたはデータベース・サーバを起動するときに、サーバ識別情報 *serv1_cert.crt* およびパスワード SERV1privatekeypwd を参照します。

- 次の例では、gencert -q オプションを使用して、*certreq.txt* という証明書要求に署名します。

```
>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
```

証明書要求は、reqtool を使用して作成されます。詳細については、次の項目を参照してください。

- 「グローバル署名証明書」214 ページ (Mobile Link トランスポート・レイヤ・セキュリティの場合)
- 『SQL Anywhere Studio セキュリティ・ガイド』> 「グローバル署名証明書」(Adaptive Server Anywhere トランスポート・レイヤ・セキュリティの場合)

第 18 章

Mobile Link システム・テーブル

この章の内容

この章では、Mobile Link の管理テーブルについて説明します。

概要

Mobile Link システム・テーブルには、Mobile Link ユーザ、サブスクリプション、テーブル、スクリプト、スクリプト・バージョンの情報が保管されています。このシステム・テーブルは統合データベース用の Mobile Link 設定スクリプトを実行したときに作成され、Mobile Link を使用するたびに更新されます。Mobile Link システム・テーブルは、直接変更できる点が RDBMS システム・テーブルと異なります。

このテーブルを作成する方法については、『Mobile Link 管理ガイド』> 「統合データベースの設定」を参照してください。

Mobile Link システム・テーブルは統合データベースに格納されます。ほとんどのシステム・テーブルとは異なり、Mobile Link システム・テーブルは修正できます。しかし、まれな状況を除き、直接修正しないでください。また、ml_qa で始まる Mobile Link システム・テーブルは変更しないでください。

Adaptive Server Anywhere のリモート・データベースにもシステム・テーブルがあります。詳細については、『ASA SQL リファレンス・マニュアル』> 「システム・テーブル」を参照してください。Ultra Light データベースにはシステム・テーブルがありません。

説明

- この章では、Adaptive Server Anywhere 統合データベースの Mobile Link システム・テーブルで使用されるデータ型について説明します。一部の RDBMS では、データ型が多少異なります。
- IBM DB2 UDB バージョン 5.2 では、カラム名と他の識別子に 18 文字までしか使用できません。DB2 5.2 統合データベースでは、Mobile Link システム・テーブルが必要に応じてトランケートされます。

ml_connection_script

指定されたスクリプト・バージョンにおいて、スクリプトを指定されたイベントに関連付けます。

カラム	説明
version_id	INTEGER。プライマリ・キー。接続スクリプトのバージョンです。version_id カラムは、ml_script_version テーブルの version_id カラムを参照します。
event	VARCHAR(128)。プライマリ・キー。event カラムは、接続スクリプトをトリガするイベントの名前を格納します。
script_id	INTEGER。外部キー。script_id カラムは、ml_script システム・テーブルの script_id カラムを参照します。接続スクリプトのテキストは、ml_script システム・テーブルに格納されます。

備考

ml_connection_script Mobile Link システム・テーブルの内容を簡単に表示できるビュー (ml_connection_scripts) があります。

ml_device

このテーブルは、サーバ起動同期にのみ使用されます。また、このテーブルにはデバイスの追跡に必要なデバイス名が格納されます。

カラム	説明
device_name	VARCHAR(255)。プライマリ・キー。
listener_version	VARCHAR(128)。非 null。
listener_protocol	INTEGER。非 null。
info	VARCHAR(255)。非 null。
ignore_tracking	CHAR(1)。非 null。
source	VARCHAR(255)。非 null。

説明

- **device_name** デバイスに指定されている名前。dblsn -e オプションを使用して名前を指定した場合を除き、この名前はオペレーティング・システムから取り出されます。
- **listener_version** このフィールドには、デバイスにインストールされているソフトウェア用の SQL Anywhere Studio のバージョン番号が含まれています。この値を変更してもソフトウェアの動作には影響しませんが、この値は診断時に役立つことがあります。
- **listener_protocol** バージョン 9.0.1 よりも前の SQL Anywhere Studio の Listener と、-s オプションを使用して起動した Listener の場合、この値は **0** です。9.0.0 よりも後の Palm Listener の場合は **1** です。-s オプションを使用せずに起動した、9.0.0 よりも後の Windows Listeners の場合は **2** です。
- **info** デバイスの情報。このフィールドに情報が格納されるのは、dblsn -f オプションを使用して情報を入力した場合です。
- **ignore_tracking** このカラムが **y** である場合は、追跡情報はローに書き込まれません。このカラムが **n** である場合は、追跡情報がローに書き込まれます。

- **source** デバイスの自動追跡機能によってローが作成された場合、このカラムは **tracking** です。そうでない場合は、このカラムは空です。ただし、ストアド・プロシージャを使用してこのカラムを変更し、このローのデータの取得先に関する情報を追加した場合を除きます。このカラムが **tracking** に設定されていない場合は、カラムの値がソフトウェアの動作に影響することはありません。

備考

Mobile Link システム・テーブル ml_device、ml_device_address、ml_listening には、サーバ起動同期に使用されるデバイスに関する情報が格納されています。DeviceTracker ゲートウェイは、この情報を使用して、Mobile Link ユーザ名ごとにターゲット・デバイスを処理します。

ほとんどの場合、これらのテーブルを変更する必要はありません。ただし、デバイスがデバイス追跡機能をサポートしていない場合、またはトラブルシューティングのためにデバイス追跡機能を無効にする場合は、事前に定義されたストアド・プロシージャを使用して、このシステム・テーブルのローを追加または削除できます。詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「デバイス・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」を参照してください。

自動追跡機能を停止する場合は、override_tracking を **y** に設定します。また、この場合は **tracking** 以外のソース名を使用してください。詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「ml_set_device」を参照してください。

参照

- 『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「ml_set_device」
- 『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「ml_delete_device」

ml_device_address

このテーブルは、サーバ起動同期にのみ使用されます。また、このテーブルにはデバイスの追跡に必要なアドレス情報が格納されます。

カラム	説明
device_name	VARCHAR(255)。プライマリ・キー。外部キー参照 dbo.ml_device。非 null。
medium	VARCHAR(255)。プライマリ・キー。非 null。
address	VARCHAR(255)。非 null。
active	CHAR(1)。非 null。
last_modified	タイムスタンプ。非 null。デフォルトのタイムスタンプ。
ignore_tracking	CHAR(1)。非 null。
source	VARCHAR(255)。非 null。

説明

- **device_name** デバイスに指定されている名前。dblsn -e オプションを使用して名前を指定した場合を除き、この名前はオペレーティング・システムから取り出されます。
- **medium** UDP では、これは **_UDP_** です。それ以外の場合、これはネットワーク・プロバイダ ID です。
- **address** UDP では、これは *ip*:*port-number* です。この場合、*ip* は IP アドレスまたはホスト名です。SMTP では、これは電話番号です。
- **active** アクティブな場合、これは **y** で、それ以外の場合は **n** です。DeviceTracker ゲートウェイは、UDP チャネルの応答がなく、予備の SMTP 配信パスがある場合に UDP チャネルを非アクティブにすることがあります。
- **last_modified** このローが最後に更新されたのがいつであるかを示します。

- **ignore_tracking** このカラムが **y** である場合は、追跡情報はローに書き込まれません。このカラムが **n** である場合は、追跡情報がローに書き込まれます。

- **source** デバイスの自動追跡機能によってローが作成された場合、このカラムは **tracking** です。そうでない場合は、このカラムは空です。ただし、ストアド・プロシージャを使用してこのカラムを変更し、このローのデータの取得先に関する情報を追加した場合を除きます。このカラムが **tracking** に設定されていない場合は、カラムの値がソフトウェアの動作に影響することはありません。

備考

Mobile Link システム・テーブル ml_device、ml_device_address、ml_listening には、サーバ起動同期に使用されるデバイスに関する情報が格納されています。DeviceTracker ゲートウェイは、この情報を使用して、Mobile Link ユーザ名ごとにターゲット・デバイスを処理します。

ほとんどの場合、これらのテーブルを変更する必要はありません。ただし、デバイスがデバイス追跡機能をサポートしていない場合、またはトラブルシューティングのためにデバイス追跡機能を無効にする場合は、事前に定義されたストアド・プロシージャを使用して、このシステム・テーブルのローを追加または削除できます。詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「デバイス・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」を参照してください。

自動追跡機能を停止する場合は、override_tracking を **y** に設定します。また、この場合は **tracking** 以外のソース名を使用してください。詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「ml_set_device_address」を参照してください。

参照

- 『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「ml_set_device_address」
- 『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「ml_delete_device_address」

ml_listening

このテーブルは、サーバ起動同期にのみ使用されます。また、Mobile Link ユーザ名を、デバイスの追跡に使用するデバイス名にマッピングします。

カラム	説明
ml_user	VARCHAR(128)。プライマリ・キー。非 null。
device_name	VARCHAR(255)。外部キー参照 dbo.ml_device。非 null。
listening	CHAR(1)。非 null。
ignore_tracking	CHAR(1)。非 null。
source	VARCHAR(255)。非 null。

説明

- **ml_user** Mobile Link ユーザ名。これによって、リモート・データベースがユニークに識別されます。

  dblsn -t+ オプションを使用する場合は、ml_user に対して定義されているエイリアスです。詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「デバイス・トラッキング用の Listener オプション」を参照してください。

- **device_name** デバイスに指定されている名前。dblsn -e オプションを使用して名前を指定した場合を除き、この名前はオペレーティング・システムから取り出されます。

- **listening** アクティブな Listener の場合、これは **y** で、それ以外の場合は **n** です。このフィールドは、dblsn オプション -t を使用したときに設定されます。また、ストアド・プロシージャを使用して手動で設定することもできます。

- **ignore_tracking** このカラムが **y** である場合は、追跡情報はローに書き込まれません。このカラムが **n** である場合は、追跡情報がローに書き込まれます。

- **source** デバイスの自動追跡機能によってローが作成された場合、このカラムは **tracking** です。そうでない場合は、このカラムは空です。ただし、ストアド・プロシージャを使用してこのカラムを変更し、このローのデータの取得先に関する情報を追加した場合を除きます。このカラムの値は、ソフトウェアの動作に影響しません。

備考

Mobile Link システム・テーブル ml_device、ml_device_address、ml_listening には、サーバ起動同期に使用されるデバイスに関する追跡情報が格納されています。DeviceTracker ゲートウェイは、この情報を使用して、Mobile Link ユーザ名ごとにターゲット・デバイスを処理します。

ほとんどの場合、これらのテーブルを変更する必要はありません。ただし、デバイスがデバイス追跡機能をサポートしていない場合、またはトラブルシューティングのためにデバイス追跡機能を無効にする場合は、事前に定義されたストアド・プロシージャを使用して、このテーブルのローを追加または削除できます。詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「デバイス・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」を参照してください。

自動追跡機能を停止する場合は、override_tracking を Yes に設定します。また、この場合は **tracking** 以外のソース名を使用してください。

参照

- 『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「ml_set_listening」
- 『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「ml_delete_listening」

ml_property

いくつかの Mobile Link のプロパティを格納します。

カラム	説明
component_name	VARCHAR(128)。複合プライマリ・キーの最初の部分。
property_set_name	VARCHAR(128)。複合プライマリ・キーの 2 番目の部分。
property_name	VARCHAR(128)。複合プライマリ・キーの 3 番目の部分。
property_value	LONG VARCHAR

説明

- **component_name** ユーザ定義のプロパティの場合、これは **ScriptVersion** または **SIS** です。
- **prop_set_name** component_name が **ScriptVersion** である場合、これはスクリプト・バージョンの名前です。

  component_name が **SIS** である場合、これはプロパティを設定している Notifier、ゲートウェイ、または Carrier の名前です。
- **prop_name** これは、プロパティの名前です。

  component_name が **ScriptVersion** である場合、これはユーザ定義のプロパティです。

  component_name が **SIS** である場合、これは Notifier、ゲートウェイ、または Carrier のプロパティです。プロパティのリストについては、『Mobile Link サーバ起動同期ユーザーズ・ガイド』> 「Mobile Link 通知プロパティ」を参照してください。
- **prop_value** これは、プロパティの値です。

備考

このテーブルには、名前と値の組み合わせが格納されます。このテーブルの一部のプロパティは、Mobile Link によって内部的に使用されます。さらに、ストアド・プロシージャ ml_add_property を使用して、このテーブルにローを追加または削除できます。

component_name **ScriptVersion** を使用すると、スクリプト・バージョンごとに情報を格納し、その情報に Java や .NET のスクリプト論理でアクセスできます。

このテーブルには、component_name、property_set_name、property_name で構成される複合プライマリ・キーがあります。

参照

- 「ml_add_property」602 ページ

ml_qa_delivery

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。

カラム	説明
msgid	VARCHAR(255)。グローバルにユニークなメッセージ識別子。
address	VARCHAR(255)。ターゲット受信者のアドレス。
client	VARCHAR(255)。ターゲット受信者のクライアント。
status	INTEGER。メッセージのステータス。**1** (保留)、**10** (受信中)、**30** (有効期限切れ)、**50** (受信可能)、**60** (受信済み) のいずれかです。
statustime	TIMESTAMP。このステータスになった日時。
verbiage	VARCHAR(32767)。ステータスのローカライズされた説明 (存在する場合)。
syncstatus	INTEGER。このメッセージに関する、クライアントとサーバ間の同期の状態。**0** (非同期)、**1** (同期)、**2** (メッセージが同期することはない) のいずれかです。
receiverid	VARCHAR(128)。メッセージの受信者を識別する、受信者によって設定される識別子 (存在する場合)。
last_modified	TIMESTAMP。ステータスが最後に変更された日時。

備考

このテーブルの所有者は ml_qa_user_group です。

ml_qa_delivery_client

カラム	説明
msgid	VARCHAR(255)。グローバルにユニークなメッセージ識別子。
address	VARCHAR(255)。ターゲット受信者のアドレス。
client	VARCHAR(255)。ターゲット受信者のクライアント。
status	INTEGER。メッセージのステータス。**1** (保留)、**10** (受信中)、**30** (有効期限切れ)、**50** (受信可能)、**60** (受信済み) のいずれかです。
statustime	TIMESTAMP。このステータスになった日時。
verbiage	VARCHAR(32767)。ステータスのローカライズされた説明 (存在する場合)。
syncstatus	INTEGER。このメッセージに関する、クライアントとサーバ間の同期の状態。**0** (非同期)、**1** (同期)、**2** (メッセージが同期することはない) のいずれかです。
receiverid	VARCHAR(128)。メッセージの受信者を識別する、受信者によって設定される識別子 (存在する場合)。

備考　このテーブルの所有者は ml_qa_user_group です。

ml_qa_global_props

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。このテーブルには、グローバルな *name-value* の組み合わせが含まれています。この組み合わせは、転送ルールで使用されます。

カラム	説明
client	VARCHAR(255)。プライマリ・キー。プロパティに関連付けられているクライアント。クライアント値が '' の場合は、すべてのクライアントにとってグローバルなプロパティを表します。
name	VARCHAR(255)。プライマリ・キー。プロパティ名。
modifiers	INTEGER。プロパティを詳しく説明するためのビット・フィールド。現在は、先頭ビットのみが、同期できないプロパティを指定するために使用されています。他のすべてのビット・フィールドは、今後の使用のために確保されています。
value	VARCHAR(32767)。プロパティの値。
last_modified	TIMESTAMP。値が最後に変更された日時。プロパティをクライアントといつ同期する必要があるかを指定するために必要です。

備考

このテーブルには、クライアントと名前で構成された複合プライマリ・キーがあります。

このテーブルの所有者は ml_qa_user_group です。

このテーブルの内容は修正しないでください。

ml_qa_global_props_client

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。また、このテーブルは ml_qa_global_props と同じです。必要に応じて、QAnywhere Agent によってリモート・データベースに作成されます。

このテーブルの内容は修正しないでください。

カラム	説明
name	VARCHAR(255)。プライマリ・キー。プロパティ名。
modifiers	INTEGER。プロパティを詳しく説明するためのビット・フィールド。現在は、先頭ビットのみが、同期できないプロパティを指定するために使用されています。他のすべてのビット・フィールドは、今後の使用のために確保されています。
value	VARCHAR(32767)。プロパティの値。

備考

このテーブルの所有者は ml_qa_user_group です。

このテーブルの内容は修正しないでください。

参照

- 「ml_qa_global_props」632 ページ

ml_qa_notifications

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。また、同期の開始を通知する QAnywhere ユーザを判別するために Notifier によって使用されます。

カラム	説明
user_id	INTEGER。プライマリ・キー。
name	VARCHAR(128)。Mobile Link ユーザ名。これによって、リモート・データベースがユニークに識別されます。

備考

このテーブルの所有者は ml_qa_user_group です。

このテーブルの内容は修正しないでください。

ml_qa_repository

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。このテーブルには、メッセージとそのプロパティが格納されます。

カラム	説明
seqno	BIGINT。メッセージ全体の順序を指定するために使用されます。正確なキューイングに必要です。
msgid	VARCHAR(255)。プライマリ・キー。グローバルにユニークなメッセージ識別子。
originator	VARCHAR(255)。送信元の Mobile Link ユーザの名前。
priority	INTEGER。**0** ～ **9** の数字。優先度が高い数字が付いているメッセージが配信されてから、優先度の低い数字が付いているメッセージが配信されます。デフォルトは **4** です。
expires	TIMESTAMP。有効期限の日時で、これを過ぎるとメッセージが配信されなくなります。
kind	INTEGER。メッセージがバイナリ (**1**) であるか、テキスト (**2**) であるかを示します。
contentsize	BIGINT。メッセージのサイズ。バイナリ・メッセージの場合はバイト数です。テキスト・メッセージの場合は文字数です。
props	LONG BINARY。メッセージのプロパティのエンコード。プロパティの詳細については、「ml_qa_repository_props」639 ページ を参照してください。
content	LONG BINARY。メッセージの内容。テキスト・メッセージは、UTF-8 でエンコードされます。

備考

このテーブルの所有者は ml_qa_user_group です。

このテーブルの内容は修正しないでください。

ml_qa_repository_client

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。また、このテーブルは ml_qa_repository と同じです。必要に応じて、QAnywhere Agent によってリモート・データベースに作成されます。

カラム	説明
seqno	BIGINT。メッセージ全体の順序を指定するために使用されます。正確なキューイングに必要です。
msgid	VARCHAR(255)。プライマリ・キー。グローバルにユニークなメッセージ識別子。
syncstatus	INTEGER。このメッセージに関する、クライアントとサーバ間の同期の状態。**0** は同期していない状態を表し、**1** は同期している状態を表します。**2** はメッセージが同期できないことを表します。
originator	VARCHAR(255)。送信元の Mobile Link ユーザの名前。
priority	INTEGER。**0** ～ **9** の数字。優先度が高い数字が付いているメッセージが配信されてから、優先度の低い数字が付いているメッセージが配信されます。デフォルトは **4** です。
expires	TIMESTAMP。有効期限の日時で、これを過ぎるとメッセージが配信されなくなります。
kind	INTEGER。メッセージがバイナリ (**1**) であるか、テキスト (**2**) であるかを示します。
contentsize	BIGINT。メッセージのサイズ。バイナリ・メッセージの場合はバイト数です。テキスト・メッセージの場合は文字数です。
props	LONG BINARY。メッセージのプロパティのエンコード。プロパティの詳細については、「ml_qa_repository_props」639 ページ を参照してください。
content	LONG BINARY。メッセージの内容。テキスト・メッセージは、UTF-8 でエンコードされます。

備考

このテーブルの所有者は ml_qa_user_group です。

このテーブルの内容は修正しないでください。

参照

- 「ml_qa_repository」635 ページ

ml_qa_repository_content_client

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。また、このテーブルは ml_qa_repository_content と同じです。必要に応じて、QAnywhere Agent によってリモート・データベースに作成されます。

このテーブルの所有者は ml_qa_user_group です。

このテーブルの内容は修正しないでください。

ml_qa_repository_props

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。また、これは ml_qa_repository テーブルの props カラムを拡張したものです。必要に応じて、プロパティは転送ルールのエンジンによってのみ拡張されます。関連付けられたルールがない場合は、プロパティは拡張されません。

カラム	説明
msgid	VARCHAR(255)。プライマリ・キー。グローバルにユニークなメッセージ識別子。
name	VARCHAR(255)。プライマリ・キー。プロパティ名。プロパティ名が Unicode で指定されている場合は、データベースのネイティブの文字セットに変換されます。
value	VARCHAR(32767)。プロパティの値。

備考

このテーブルの所有者は ml_qa_user_group です。

このテーブルの内容は修正しないでください。

このテーブルには、msgid と名前で構成された複合プライマリ・キーがあります。

ml_qa_repository_props_client

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。また、このテーブルは ml_qa_repository_props と同じです。必要に応じて、QAnywhere Agent によってリモート・データベースに作成されます。

このテーブルの内容は修正しないでください。

カラム	説明
msgid	VARCHAR(255)。プライマリ・キー。グローバルにユニークなメッセージ識別子。
name	VARCHAR(255)。プライマリ・キー。プロパティ名。プロパティ名が Unicode で指定されている場合は、データベースのネイティブの文字セットに変換されます。
value	VARCHAR(32767)。プロパティの値。

備考

このテーブルの所有者は ml_qa_user_group です。

参照

- 「ml_qa_repository_props」639 ページ

ml_qa_repository_staging

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。これには、QAnywhere クライアントに送信されるメッセージが格納されます。

カラム	説明
seqno	BIGINT。メッセージ全体の順序を指定するために使用されます。正確なキューイングに必要です。
msgid	VARCHAR(255)。プライマリ・キー。グローバルにユニークなメッセージ識別子。
destination	VARCHAR(255)。メッセージのアドレス。
originator	VARCHAR(255)。送信元の Mobile Link ユーザの名前。
status	VARCHAR(255)。メッセージのステータス。**pending**、**receiving**、**received**、**unreceivable**、**expired**、または **cancelled** に指定できます。デフォルトは **pending** です。
statustime	TIMESTAMP。ステータスが最後に変更された日時。
uploaded	BIT。送信者と受信者のメッセージのステータスが同期しているかどうかを示します。**0** は同期していない状態を示し、**1** は同期している状態を示します。デフォルトは **0** です。
expires	TIMESTAMP。有効期限の日時で、これを過ぎるとメッセージが配信されなくなります。
priority	INTEGER。**0** ～ **9** の数字。常に、優先度が高い数字が付いているメッセージが配信されてから、優先度の低い数字が付いているメッセージが配信されます。デフォルトは **4** です。
props	LONG BINARY。メッセージのプロパティのエンコード。
kind	INTEGER。メッセージがバイナリ (**1**) であるか、テキスト (**2**) であるかを示します。
content	LONG BINARY。メッセージの内容。テキスト・メッセージは、UTF-8 でエンコードされます。

カラム	説明
contentsize	BIGINT。メッセージのサイズ。バイナリ・メッセージの場合はバイト数です。テキスト・メッセージの場合は文字数です。
mluser	VARCHAR(128)。Mobile Link ユーザ名。これによって、リモート・データベースがユニークに識別されます。

備考

このテーブルの所有者は ml_qa_user_group です。

このテーブルの内容は修正しないでください。

ml_qa_status_staging

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。また、これは送信元のクライアントとの同期ステータスが変化したときに使用される中間テーブルです。

カラム	説明
msgid	VARCHAR(255)。プライマリ・キー。グローバルにユニークなメッセージ識別子。
status	VARCHAR(255)。メッセージのステータス。**pending**、**receiving**、**received**、**unreceivable**、**expired**、または **cancelled** に指定できます。デフォルトは **pending** です。
statustime	TIMESTAMP。ステータスが最後に変更された日時。
mluser	VARCHAR(128)。Mobile Link ユーザ名。これによって、リモート・データベースがユニークに識別されます。

備考

このテーブルの所有者は ml_qa_user_group です。

このテーブルの内容は修正しないでください。

ml_script

すべてのスクリプトのテキストを格納します。

カラム	説明
script_id	INTEGER。プライマリ・キー。script_id カラムは、スクリプトを識別するユニークな整数を格納します。
script	TEXT。script カラムはスクリプトのテキストを格納します。
script_language	VARCHAR(128)。script_language カラムは、スクリプトに使用されるスクリプト言語を格納します。スクリプト言語は、**sql**、**java**、または **dnet** が使用できます。

ml_script_version

各スクリプト・バージョンと関連づけられたスクリプトの名前、ID、コメントを格納します。

カラム	説明
version_id	INTEGER。プライマリ・キー。version_id カラムは、バージョンを識別するユニークな整数を格納します。
name	VARCHAR(128)。name カラムは、バージョンに指定された任意の名前を格納します。
description	TEXT。description カラムは、バージョンに指定された任意の説明を格納します。この説明は Mobile Link が使用するものではありませんが、アプリケーション固有のコメントとして役立ちます。たとえば、指定したスクリプト・バージョンの目的を説明します。

ml_scripts_modified

スクリプト・テーブルが最後に変更された時刻を格納します。Mobile Link サーバは、新しいスクリプトをロードするかどうか決定するためにこのテーブルを確認します。

カラム	説明
last_modified	DATETIME。プライマリ・キー。last_modified カラムは、ml_script_version、ml_script、または ml_connection_script システム・テーブルが最後に変更された時刻を格納します。

ml_subscription

Adaptive Server Anywhere リモート・データベースの各サブスクリプションに対するログ・オフセット (progress) を追跡します。

カラム	説明
user_id	INTEGER。プライマリ・キー。user_id カラムは、ml_user テーブルの user_id カラムを参照します。これは、パブリケーションにサブスクライブしたユーザの ID です。
subscription_id	VARCHAR(128)。プライマリ・キー。subscription_id は、リモートの Adaptive Server Anywhere が生成する番号です。
publication_name	VARCHAR(128)。プライマリ・キー。publication_name は、ユーザが定義したパブリケーションの名前を格納します。
progress	NUMERIC(20,0)。progress はオフセットまたはステータスとも呼ばれ、サブスクリプションのすべての操作がアップロードされたところまでのリモート・トランザクション・ログの位置を示します。このカラムは、バージョン 8.0 以降のデータベースに使用されます。バージョン 7 のデータベースでは、progress は ml_user テーブルに格納されます。
last_download_time	TIMESTAMP。指定したユーザまたはサブスクリプションのダウンロードが、統合データベースに最後に適用された時刻を示します。デフォルトは January 1, 1900, 00:00:00 です。
last_upload_time	TIMESTAMP。指定したユーザまたはサブスクリプションのアップロードが、統合データベースに最後に適用された時刻を示します。デフォルトは January 1, 1900, 00:00:00 です。

備考

「**progress**」は、「**オフセット**」または「**ステータス**」とも呼ばれ、リモート・データベースのトランザクション・ログの位置を示します。これは、サブスクリプションに対するすべての操作がアップロードされ確認された点を示します。dbmlsync は、オフセットを使用してどのデータをアップロードするか決定します。

リモート・データベースでは、オフセットは SYS.SYSSYNC システム・テーブルの progress カラムに格納されます。

統合データベースでは、バージョン 7.x データベースのオフセットは ml_user テーブルの progress カラムに格納され、バージョン 8.0 以降のデータベースのオフセットは ml_subscription テーブルの progress カラムに格納されます。

ml_table

リモート・テーブルの名前を格納します。このリストには、Sybase Central で同期テーブルとしてマーク付けされたすべてのテーブルが含まれます。

カラム	説明
table_id	INTEGER。プライマリ・キー。table_id カラムは、テーブルを識別するユニークな整数を格納します。
name	VARCHAR(128)。name カラムは、テーブルに指定された任意の名前を格納します。

ml_table_script

指定されたスクリプト・バージョンにおいて、テーブル・スクリプトを指定されたテーブルとイベントに関連付けます。

カラム	説明
version_id	INTEGER。プライマリ・キー。version_id カラムは、ml_script_version テーブルの version_id カラムを参照します。
table_id	INTEGER。プライマリ・キー。table_id カラムは、ml_table システム・テーブルの table_id カラムを参照します。
event	VARCHAR(128)。プライマリ・キー。event カラムはイベントの名前を格納します。
script_id	INTEGER。script_id カラムは、ml_script テーブルの script_id カラムを参照します。スクリプトは ml_script テーブルに格納されます。

備考

ml_table_script Mobile Link システム・テーブルの内容を簡単に表示できるビュー (ml_table_scripts) があります。

ml_user

すべての登録済み Mobile Link ユーザと、そのユーザのパスワードと同期ステータスを格納します。このテーブルの progress は、Ultra Light のリモートまたは Adaptive Server Anywhere バージョン 7 のリモートにだけ使用されます。

カラム	説明
user_id	INTEGER。プライマリ・キー。user_id カラムは、ユーザを識別するユニークな整数を格納します。この値は Mobile Link が内部でのみ使用します。
name	VARCHAR(128)。name カラムは、ユーザに指定された任意の名前を格納します。
commit_state	INTEGER。commit_state カラムはステータスを格納します。
progress	NUMERIC(20,0)。progress はログ・オフセットまたはステータスとも呼ばれ、サブスクリプションのすべての操作がアップロードおよび確認されたところまでのトランザクション・ログの位置を示します。このカラムは、バージョン 7 データベースに使用されます。バージョン 8.0 以降のデータベースでは、progress は ml_subscription テーブルに格納されます。
hashed_password	BINARY(20)。hashed_password カラムは、Mobile Link ユーザのパスワードを難読化した形式で格納します。パスワードがない場合、この値は NULL です (これはおすすめしません)。
last_download_time	TIMESTAMP。指定したユーザまたはサブスクリプションのダウンロードが、統合データベースに最後に適用された時刻を示します。デフォルトは January 1, 1900, 00:00:00 です。
last_upload_time	TIMESTAMP。指定したユーザまたはサブスクリプションのアップロードが、統合データベースに最後に適用された時刻を示します。デフォルトは January 1, 1900, 00:00:00 です。

備考

「**progress**」は、「**ログ・オフセット**」または「**ステータス**」とも呼ばれ、リモート・データベースのトランザクション・ログの位置を示します。これは、サブスクリプションに対するすべての操作がアップロードされ確認された点を示します。dbmlsync は、オフセットを使用してどのデータをアップロードするか決定します。

リモート・データベースでは、オフセットは SYS.SYSSYNC システム・テーブルの progress カラムに格納されます。

統合データベースでは、バージョン 7.x データベースのオフセットは ml_user テーブルの progress カラムに格納され、バージョン 8.0 以降のデータベースのオフセットは ml_subscription テーブルの progress カラムに格納されます。

第 19 章

データ型の変換

この章の内容

この章では、Mobile Link 同期サーバが、Adaptive Server Anywhere 以外の統合データベースと通信する場合に発生するデータ型の変換についての情報を提供します。以下の表は、データ型の変換を示します。

.NET 言語または Java で同期スクリプトを作成する場合、SQL データ型を Java や .NET データ型にマップする方法を把握している必要があります。詳細については、『Mobile Link 管理ガイド』> 「SQL データ型と .NET データ型」および 『Mobile Link 管理ガイド』> 「SQL データ型と Java データ型」を参照してください。

注意
この章では、サポートされているデータ型についてのみ説明します。

Adaptive Server Enterprise データのマッピング

次の表は、Adaptive Server Anywhere または Ultra Light のデータ型がどのように Adaptive Server Enterprise データ型にマップされるかを示します。

Adaptive Server Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型
bit	bit
tinyint	tinyint
smallint	smallint
int	int
integer	integer
decimal [デフォルトは p=30, s=6]	numeric(30,6)
numeric [デフォルトは 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)

Adaptive Server Anywhere または Ultra Light のデータ型	**Adaptive Server Enterprise データ型**
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)
uniqueidentifier	varchar(n) [1]

[1] *n* は 36 より大きな値。

IBM DB2 データのマッピング

次の表は、Adaptive Server Anywhere または Ultra Light のデータ型がどのように IBM DB2 データ型にマップされるかを示します。

Adaptive Server Anywhere または Ultra Light のデータ型	**IBM DB2 データ型**
bit	smallint
tinyint	smallint
smallint	smallint
int	int
integer	int
bigint	decimal(20,0)
char(1-4000)	varchar(n)
char(4001-32767)	long varchar
character(1-4000)	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 または CLOB(n)
long varchar	long varchar または CLOB(n)
text	long varchar
binary(1-4000)	varchar (bit データの場合) または BLOB(n)
binary(4001-32767)	long varchar (bit データの場合) または BLOB(n)

Adaptive Server Anywhere または Ultra Light のデータ型	IBM DB2 データ型
long binary	long varchar (bit データの場合) または BLOB(n)
image	long varchar (bit データの場合) または BLOB(n)
decimal [デフォルトは p=30, s=6]	decimal(30,6)
numeric [デフォルトは p=30 s=6]	decimal(30,6)
real	real
float	float
double	float
smallmoney	decimal(10,4)
money	decimal(19,4)
date	date
time	time
smalldatetime	timestamp
datetime	timestamp
timestamp	timestamp
uniqueidentifier	varchar(n) [1]

[1] *n* は 36 より大きな値。

Oracle データのマッピング

次の表は、Adaptive Server Anywhere または Ultra Light のデータ型がどのように Oracle データ型にマップされるかを示します。

Adaptive Server Anywhere または Ultra Light のデータ型	**Oracle データ型**
bit	number(1,0)
tinyint	number(3,0)
smallint	number(5,0)
int	number(11,0)
integer	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(10,4)
money	number(19,4)
date	date
time	date
timestamp	date
smalldatetime	date
datetime	date
char(n)	varchar(n) または CLOB(n)
varchar(n)	varchar(n) または CLOB(n)

Adaptive Server Anywhere または Ultra Light のデータ型	**Oracle データ型**
long varchar	CLOB
binary(n)	raw(n) または BLOB(n)
varbinary(n)	raw(n) または BLOB(n)
long binary	BLOB
uniqueidentifier	varchar2(n) [1]

[1] *n* は 36 より大きな値。

LONG データ型は、Oracle 8、8i、9i では使用されなくなりました。

Oracle の LONG データ型を正常に同期させるには、[ODBC データ・ソース設定] ダイアログで Oracle の [**LONG 型コラムの強制検索**] の ODBC オプションをチェックします。

Microsoft SQL Server データのマッピング

次の表は、Adaptive Server Anywhere または Ultra Light のデータ型がどのように Microsoft SQL Server データ型にマップされるかを示します。

Adaptive Server Anywhere または Ultra Light のデータ型	**Microsoft SQL Server データ型**
bit	bit
tinyint	tinyint
smallint	smallint
int	int
integer	int
bigint	numeric(20,0) または bigint (SQL Server 2000 のみ)
decimal [デフォルトは p=30, s=6]	decimal(30, 6)
numeric [デフォルトは p=30 s=6]	numeric(30,6)
float	float
real	real
smallmoney	smallmoney
money	money
date	datetime
time	datetime
timestamp	datetime
smalldatetime	datetime
datetime	datetime
char(n)	varchar(n) または text
character(n)	varchar(n)
varchar(n)	varchar(n) または text

Adaptive Server Anywhere または Ultra Light のデータ型	**Microsoft SQL Server データ型**
long varchar	text
binary(n)	binary(n) または image
long binary	image
double	float
uniqueidentifier	uniqueidentifier

第 20 章

文字セットの考慮事項

この章の内容

この章では、Mobile Link アプリケーションで国際言語の問題を処理する方法について説明します。

文字セットの考慮事項

テキストの各文字は、1 バイトまたはそれ以上のバイトで表現されます。文字からバイナリ・コードへのマッピングを「**文字セット・エンコード**」といいます。ヨーロッパ言語など、小規模なアルファベット体系を持つ言語で使われる文字セットには、シングルバイト表現が使用されるものがあります。ユニコードなどのように 2 バイト表現を使用する文字セットもあります。2 バイト文字セットは、各文字について記憶領域を 2 倍使用するため、はるかに多くの文字を表現できます。

変換エラーが起きたり、データが失われたりするのは、1 つの文字セットを使用しているテキストを別の文字セットに変換しなければならないときです。すべての文字がすべての文字セットで表示できるわけではありません。特に、シングルバイト文字セットは、使用可能なコードの数が限られているので、表示できる文字はマルチバイト・システムより少なくなります。

Mobile Link リモート・データベースの文字セットがユーザの統合データベースと同じ場合は、文字変換の問題は起こりません。

ディレクトリのリストのようにインデックスを構築したり順序リストを準備したりするには、テキストをソートする必要がよくあります。「**ソート順**」は、文字の順序を特定します。たとえば、一般的にソート順では文字 "a" は文字 "b" の前に位置し、文字 "b" は文字 "c" の前に位置します。

各データベースには、「**照合順**」があります。照合順は、データベースを作成するときに設定します。設定方法は、データベースのシステムによって異なります。照合順は、データベースの文字セットとソート順の両方を定義します。

ヒント

可能な場合は常に、使用しているリモート・データベースの照合順を、使用している統合データベースの照合順と同じになるように定義してください。この方法により、間違った変換をする可能性が減ります。

詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「Ultra Light の文字セット」および『ASA データベース管理ガイド』>「国際言語と文字セット」を参照してください。

同期中の文字セットの変換：Windows

同期中は、ある文字セットの文字を別の文字セットに変換しなければならないことがあります。次の変換は、文字がリモート・アプリケーションと統合データベースの間で渡されるときに起こります。

アップロード中の文字セットの変換

Mobile Link クライアントは、リモート・データベースの文字セットを使用して Mobile Link 同期サーバにデータを送ります。

1. Mobile Link 同期サーバは、Unicode ODBC API を使用して統合データベースと通信します。通信することにより、Mobile Link 同期サーバはリモート・データベースから受け取ったすべての文字をユニコードに変換します。

2. 必要に応じて、統合データベース・サーバの ODBC ドライバは文字をユニコードから統合データベースの文字セットに変換します。この変換を制御できるのは、使用している統合データベース・システムの ODBC ドライバだけです。したがって、2 つの異なるデータベース・システム (特に、異なる製造業者によって作成されたシステム) では、動作が異なる場合があります。Mobile Link 同期は、多くのデータベース・システムで動作します。詳細については、使用している統合サーバと ODBC ドライバのマニュアルを参照してください。

ダウンロード中の文字セットの変換

1. 統合データベース・システムの ODBC ドライバは、文字を統合データベースのコードで受け取ります。このドライバは、これらの文字をユニコードに変換し、Unicode API を通して Mobile Link 同期サーバに渡します。この変換を制御できるのは、使用している統合データベース・システムの ODBC ドライバだけです。詳細については、使用している統合サーバと ODBC ドライバのマニュアルを参照してください。

2. Mobile Link 同期サーバは、Unicode ODBC API を通して文字を受け取ります。リモート・データベースで異なる文字セットを使用している場合は、ダウンロードする前に Mobile Link 同期サーバが文字を変換します。

例

- Windows CE デバイス上の Ultra Light アプリケーションでは、ユニコード文字セットを使用します。

  Windows CE アプリケーションを同期するときは、Mobile Link 同期サーバ内で文字変換は行われません。Mobile Link 同期サーバは、アプリケーションから送信されたデータがすでにユニコードであると判断し、それを ODBC ドライバに直接渡します。同様に、データのダウンロード時も文字セットの変換は必要ありません。

- Windows CE 以外のプラットフォーム上にあるすべての Adaptive Server Anywhere データベースとすべての Ultra Light アプリケーションは、リモート・データベースの照合順によって決定された文字セットを使用します。

  リモート・データベースを同期するときは、Mobile Link 同期サーバがリモート・データベースの文字セットとユニコードの間の文字セット変換を実行します。

ODBC ドライバの文字セット変換の制御

統合データベースはユニコードを使用していない場合が多いので、使用している統合データベース・システムの ODBC ドライバがデータをユニコードに変換する方法とユニコードから変換する方法について理解しておくことが重要です。ODBC ドライバの中には、Mobile Link を実行しているマシンの言語設定を使って、使用する文字セットを決定するものがあります。この場合、Mobile Link 同期サーバを実行しているマシンの言語設定とコード・ページ設定が、統合データベースの設定と一致するのが最も望ましいことです。

Sybase Adaptive Server Enterprise のドライバなどその他の ODBC ドライバでは、各接続で特定の文字セットを使用できます。変換のエラーを避けるために、Mobile Link で使用する文字セットが統合データベースの文字セットと一致するように設定してください。

使用している統合データベース・サーバの ODBC ドライバで文字セット変換がどのように行われるかについては、各製品の ODBC ドライバのマニュアルを参照してください。

同期中の文字セットの変換：Windows 以外

iAnywhere Solutions が Windows 以外のプラットフォームで提供する ODBC ドライバには、Unicode ODBC API が含まれません。Mobile Link 同期サーバは、リモート・データベースの照合順で決定された文字セットを使用して、ODBC ドライバとデータを交換します。

リモート・データベースが Windows CE で実行中の Ultra Light アプリケーションである場合は、Mobile Link 同期サーバが、ODBC で使用されている文字セットとユニコード間の文字セット変換を実行します。

第 21 章

iAnywhere Solutions ODBC ドライバ

この付録の内容

この付録では、Mobile Link で使用できる ODBC ドライバについて説明します。

Mobile Link でサポートされる ODBC ドライバ

次の表のとおり、Mobile Link 同期サーバは各種の統合データベースや ODBC ドライバと連携させることができます。ドライバによっては、Mobile Link と互換性があっても、使用に関する機能上の制約がある場合があります。

最新の情報と完全な機能仕様については、http://www.ianywhere.com/developer/technotes/odbc_mobilink.html (英文) を参照してください。

ODBC ドライバの設定の詳細については、『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』> 「iAnywhere Solutions ODBC ドライバの概要」を参照してください。UNIX では、このマニュアルのスタンドアロン・バージョンが、SQL Anywhere Studio インストール・ディレクトリの *drivers* サブディレクトリにインストールされます。

データベース	**ODBC ドライバ**
Oracle 8i	iAnywhere Solutions 8 － Oracle 8、8i、9i ODBC ドライバ Merant 製 Oracle 用 DataDirect Connect ODBC ドライバ
Oracle 9i	iAnywhere Solutions 8 － Oracle 8、8i、9i ODBC ドライバ Oracle 9i ODBC ドライバ
Microsoft SQL Server 7、Microsoft SQL Server 2000	Microsoft SQL Server ODBC ドライバ Merant 製 DataDirect SQL Server Wire Protocol ODBC ドライバ

データベース	ODBC ドライバ
Sybase Adaptive Server Enterprise 11.5 以降	iAnywhere Solutions 8 - Sybase ASE ODBC ドライバ Merant 製 DataDirect Sybase Wire Protocol ODBC ドライバ Merant 製 Sybase ASE 用 DataDirect Connect ODBC ドライバ Merant 製 Sybase ASE 用 Connect ODBC ドライバ
IBM DB2 UDB 7.1、7.2	IBM DB2 UDB 7.1 ODBC ドライバ IBM DB2 UDB 7.2 ODBC ドライバ Merant 製 DataDirect DB2 Wire Protocol ODBC ドライバ
Sybase Adaptive Server Anywhere 9	Adaptive Server Anywhere 9.0 ODBC ドライバ

第 22 章

Mobile Link アプリケーションの配備

この付録の内容

この付録では、Mobile Link サーバと Mobile Link クライアントを運用環境に配備する方法について説明します。配備に必要なファイルを示します。

ライセンス契約の確認

ファイルの再配布はライセンス契約に従います。このマニュアル内の記述は、ライセンス契約のどの条項にも優先しません。配備について検討する前にライセンス契約を確認してください。

配備の概要

Mobile Link アプリケーションを配備するときは、以下の作業を行います。

- 運用設定への Mobile Link サーバの配備
- Adaptive Server Anywhere Mobile Link クライアントの配備
- Ultra Light Mobile Link クライアントの配備

この章では、アプリケーションのインストール・プログラムに含める必要があるファイルについて、これらの項目ごとに説明します。

Mobile Link サーバの配備

Mobile Link 同期サーバを運用環境に配備するには、SQL Anywhere Studio のライセンス取得済みコピーを運用マシンにインストールするのが最も簡単です。

しかし、(ライセンス契約に応じて) Mobile Link 同期サーバを別のインストール・プログラムで再配布する場合は、ファイルのサブセットだけを含めてもかまいません。この場合、インストール環境に次のファイルを含める必要があります。

注意

- 再配布する前にクリーンなマシンでテストしてください。
- ファイルは、SQL Anywhere Studio インストール・ディレクトリにインストールしてください。
- 特に指定がないかぎり、ファイルを同じディレクトリにインストールしてください。
- ロケーションが指定されている場合、ファイルは同じ名前のディレクトリにコピーしてください。
- UNIX の場合は、SQL Anywhere アプリケーションおよびライブラリを配置できるように、システムに環境変数を設定してください。必要な環境変数を設定するためのテンプレートとして、*asa_config.sh* と *asa_config.csh*（*/opt/sybase/SYBSsa9/bin* ディレクトリ内) のいずれかのうち、シェルに適したファイルを使用することをおすすめします。asa_config ファイルで設定される環境変数には、PATH、LD_LIBRARY_PATH、ASANY9、ASANYSH9 があります。
- Java 同期論理を使用し、グラフィカルな管理ツール (Sybase Central および Mobile Link モニタ) を使用するには、JRE 1.4.2 をインストールします。

Windows アプリケーション

説明	Windows ファイル
Mobile Link 同期サーバ	◆ *charsets¥unicode* ◆ *win32¥dbmlsv9.dll* ◆ *win32¥dbmlsrv9.exe* ◆ *win32¥dbmsql9.dll* ◆ *win32¥dbunic9.dll*
言語ライブラリ	◆ *win32¥dblgen9.dll*[1]
Windows パフォーマンス・モニタのサポート	◆ *win32¥dbmlctr9.dll*[2] ◆ *win32¥dbmlctr9.h* ◆ *win32¥dbmlctr9.ini*
同期ストリーム・ライブラリ (使用するライブラリを配備)	◆ *win32¥dbmlhttp9.dll* ◆ *win32¥dbmlsock9.dll*
Java 同期論理	◆ *java¥activation.jar*[3] ◆ *java¥imap.jar*[3] ◆ *java¥jodbc.jar* ◆ *java¥log4j.jar*[3] ◆ *java¥mailapi.jar*[3] ◆ *java¥mlscript.jar* ◆ *java¥mlsupport.jar* ◆ *java¥pop3.jar*[3] ◆ *java¥smtp.jar*[3] ◆ *win32¥dbmjava9.dll* ◆ *win32¥mljodbc9.dll*
.NET 同期論理	◆ *MobiLink¥setup¥dnet¥mlDomConfig.xml* ◆ *win32¥dbmdnet9.dll* ◆ *win32¥dnetodbc9.dll* ◆ *win32¥iAnywhere.MobiLink.dll* ◆ *win32¥iAny-where.MobiLink.Script.dll* ◆ *win32¥iAny-where.MobiLink.Script.xml* ◆ *win32¥mlDomConfig.xsd*

説明	Windows ファイル
セキュリティ・オプション [4]	◆ *win32¥dbmlhttps.dll* ◆ *win32¥dbmlhttpsfips9.dll* ◆ *win32¥dbmlrsafips9.dll* ◆ *win32¥dbmljrsa9.dll* ◆ *win32¥dbmljtls9.dll* ◆ *win32¥dbmlrsa9.dll* ◆ *win32¥dbmltls9.dll*
スクリプト・ファイル (統合データベース用のファイルを配備)	◆ *MobiLink¥setup* ◆ *MobiLink¥upgrade*
iAnywhere Solutions ODBC ドライバ	◆ *drivers*
dbmluser ユーティリティ	◆ *win32¥dbmluser.exe*
dbmlstop ユーティリティ	◆ *win32¥dbmlstop.exe*
エラー名	◆ *h¥sserror.h*
Mobile Link モニタ	◆ *java¥mlmon.jar* ◆ *shared¥java¥HelpManager11.jar* ◆ *shared¥java¥JComponents142.jar* ◆ *shared¥java¥jsyblib142.jar* ◆ *shared¥win32¥jsyblib142.dll* ◆ *ultralite¥java¥lib¥ulrt.jar* ◆ *win32¥dbmlmon.exe*
Mobile Link プラグインおよびモニタのオンライン・ヘルプ	◆ *java¥dbmaen9.jar*[1]
Mobile Link リダイレクタ	◆ *MobiLink¥redirector*
Sybase Central	◆ *shared¥Sybase Central 4.3*
Sybase Central プラグイン	◆ *win32¥dbmlput9.dll*

説明	Windows ファイル
Notifier	◆ *java¥activation.jar*[3] ◆ *java¥jodbc.jar* ◆ *java¥log4j.jar* ◆ *java¥mailapi.jar*[3] ◆ *java¥mlnotif.jar* ◆ *java¥mlscript.jar* ◆ *java¥smtp.jar*[3]
QAnywhere	◆ *java¥log4j.jar* ◆ *java¥qaconnector.jar*

*注意：*ソフトウェアのバージョンによっては、ファイルが *win32* または *ia64* ディレクトリにある場合があります。

[1] フランス語、ドイツ語、日本語、中国語の版では、**en** をそれぞれ **fr**、**de**、**ja**、**zh** と置き換えます。

[2] セットアップ・プログラムはこのファイルを自己登録しなければなりません。

[3] アプリケーションを再配布する場合は、これらのファイルを Sun から直接入手してください。

[4] トランスポート・レイヤ・セキュリティを使用するには、別途ライセンスを取得できる SQL Anywhere Studio セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。このコンポーネントのご注文については、『SQL Anywhere Studio の紹介』> 「別途ライセンスが入手可能なコンポーネント」を参照してください。

UNIX、Linux、Macintosh のアプリケーション

説明	UNIX ファイル
Mobile Link 同期サーバ	◆ *bin/dbmlsrv9* ◆ *charsets/unicode* ◆ *lib/libdbodm9_r.so*[3] ◆ *lib/libdbmsql9_r.so*[3] ◆ *lib/libdbtasks9_r.so*[3] ◆ *lib/libdbunic9_r.so*[3]
言語ライブラリ	◆ *res/dblgen9.res*[1]
同期ストリーム・ライブラリ (使用するライブラリを配備)	◆ *lib/libdbmlhttp9_r.so*[3] ◆ *lib/libdbmlsock9_r.so*[3]
Java 同期論理	◆ *java/activation.jar*[2] ◆ *java/imap.jar*[2] ◆ *java/jodbc.jar* ◆ *java/log4j.jar*[2] ◆ *java/mailapi.jar*[2] ◆ *java/mlscript.jar* ◆ *java/mlsupport.jar* ◆ *java/pop3.jar*[2] ◆ *java/smtp.jar*[2] ◆ *lib/libdbmjava9_r.so*[3] ◆ *lib/libmljodbc9.so*[3]
.NET 同期論理	◆ なし
セキュリティ・オプション [4]	◆ *lib/libdbmlhttps9_r.so*[3] ◆ *lib/libdbmlrsa9_r.so*[3] ◆ *lib/libdbmltls9_r.so*[3]
スクリプト・ファイル (統合データベース用のファイルを配備)	◆ *MobiLink/setup* ◆ *MobiLink/upgrade*
iAnywhere Solutions ODBC ドライバ	◆ *drivers*

説明	UNIX ファイル
dbmluser ユーティリティ	◆ *bin/dbmluser*
dbmlstop ユーティリティ	◆ *bin/dbmlstop*
エラー名	◆ *h/sserror.h*
Mobile Link モニタ	◆ *bin/dbmlmon* ◆ *java/mlmon.jar* ◆ *shared/java/HelpManager11.jar* ◆ *shared/java/JComponents142.jar* ◆ *shared/java/jsyblib142.jar* ◆ *ultralite/java/lib/ulrt.jar*
Mobile Link リダイレクタ	◆ *Mobilink/redirector/redirector.config* ◆ *MobiLink/redirector/java*
Mobile Link プラグインおよび Mobile Link モニタのオンライン・ヘルプ	◆ *java/dbmaen9.jar*
Sybase Central	◆ *shared/sybcentral43*
Sybase Central プラグイン	◆ *lib/libdbmlput9_r.so*[3]
Notifier	◆ *java/activation.jar*[2] ◆ *java/jodbc.jar* ◆ *java/log4j.jar* ◆ *java/mailapi.jar*[2] ◆ *java/mlnotif.jar* ◆ *java/mlscript.jar* ◆ *java/smtp.jar*[2]
QAnywhere	◆ *java/log4j.jar* ◆ *java/qaconnector.jar*

[1] フランス語、ドイツ語、日本語、中国語の版では、**en** をそれぞれ **fr**、**de**、**ja**、**zh** と置き換えます。

[2] アプリケーションを再配布する場合は、これらのファイルを Sun から直接入手してください。

[3] Solaris および Linux のファイル拡張子は *.so* です。AIX のファイル拡張子は *.a* です。HP のファイル拡張子は *.sl* です。Macintosh のファイル拡張子は *.dylib* です。

[4] トランスポート・レイヤ・セキュリティを使用するには、別途ライセンスを取得できる SQL Anywhere Studio セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。このコンポーネントのご注文については、『SQL Anywhere Studio の紹介』> 「別途ライセンスが入手可能なコンポーネント」を参照してください。

Adaptive Server Anywhere Mobile Link クライアントの配備

Adaptive Server Anywhere クライアントの場合、Adaptive Server Anywhere データベース・サーバと Mobile Link クライアントを配備する必要があります。

Adaptive Server Anywhere データベースの配備については、『ASA プログラミング・ガイド』> 「データベースとアプリケーションの配備」を参照してください。

(ライセンス契約に応じて) Mobile Link 同期クライアントを再配布する場合は、Adaptive Server Anywhere データベースに必要なファイルの他に、以下のファイルをインストール環境に含める必要があります。

特に指定がないかぎり、ファイルを同じディレクトリにインストールしてください。

Windows アプリケーション

説明	Windows ファイル
Mobile Link 同期クライアント	◆ *win32¥dblgen9.dll*[1] ◆ *win32¥dbmlsync.exe* ◆ *win32¥dbtool9.dll* *注意* : dbtools インタフェースを使用している場合を除き、Windows CE では *dbtool9.dll* は必要ありません。
dbmlsync 統合コンポーネント	◆ Mobile Link 同期クライアント・ファイル ◆ ビジュアル・コンポーネント用の *dbmlsynccomg.dll* または非ビジュアル・コンポーネント用の *dbmlsynccom.dll*
同期ストリーム・ライブラリ (使用するライブラリを配備)	◆ *win32¥dbmlhttp9.dll* ◆ *win32¥dbmlsock9.dll*

説明	Windows ファイル
セキュリティ・オプション [2]	◆ *dbmlhttps9.dll* ◆ *dbmlhttpsfips9.dll*[3] ◆ *dbmlrsafips9.dll*[3] ◆ *dbmlrsa9.dll* ◆ *dbmltls9.dll*
ActiveSync ユーティリティと HotSync ユーティリティ	◆ *win32¥dbasinst.exe* ◆ *win32¥dbcon9.exe*
Listener	◆ *win32¥dblgen9.dll*[1] ◆ *win32¥dblsn.exe* ◆ *win32¥lsn_udp.dll* ◆ *win32¥lsn_swi510.dll* ◆ *win32¥maac555.dll* ◆ *win32¥maac750.dll* ◆ *win32¥maac750r3.dll* ◆ *win32¥mabridge.dll*

[1] フランス語、ドイツ語、日本語、中国語の版では、**en** をそれぞれ **fr**、**de**、**ja**、**zh** と置き換えます。

[2] トランスポート・レイヤ・セキュリティを使用するには、別途ライセンスを取得できる SQL Anywhere Studio セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。このコンポーネントのご注文については、『SQL Anywhere Studio の紹介』> 「別途ライセンスが入手可能なコンポーネント」を参照してください。

[3] FIPS セキュリティは、Windows CE には該当しません。

UNIX、Linux、Macintosh のアプリケーション

説明	UNIX ファイル
Mobile Link 同期クライアント	◆ *bin/dbmlsync* ◆ *res/dblgen9.res* ◆ *lib/libdbtool9.so*[1] ◆ *lib/libdbtool9_r.so*[1]

説明	UNIX ファイル
同期ストリーム・ライブラリ (使用するライブラリを配備)	◆ *lib/libdbmlhttp9_r.so*[1] ◆ *lib/libdbmlsock9_r.so*[1]
セキュリティ・オプション [2]	◆ *lib/libdbmlhttps9_r.so*[1] ◆ *lib/libdbmlrsa9_r.so*[1] ◆ *lib/libdbmltls9_r.so*[1]

[1] Solaris および Linux のファイル拡張子は *.so* です。AIX のファイル拡張子は *.a* です。HP のファイル拡張子は *.sl* です。Macintosh のファイル拡張子は *.dylib* です。

[2] トランスポート・レイヤ・セキュリティを使用するには、別途ライセンスを取得できる SQL Anywhere Studio セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。このコンポーネントのご注文については、『SQL Anywhere Studio の紹介』> 「別途ライセンスが入手可能なコンポーネント」を参照してください。

Ultra Light Mobile Link クライアントの配備

Ultra Light クライアントの場合、Ultra Light ランタイム・ライブラリまたは Ultra Light コンポーネントには、必要な同期ストリーム機能が含まれています。Ultra Light ランタイム・ライブラリは、アプリケーションにコンパイルされます。配備はライセンス契約に応じて決まります。

詳細については、Ultra Light コンポーネントまたは使用している開発モデルのマニュアルを参照してください。

Palm アプリケーション

説明	**Palm ファイル**
Windows Listener 設定ユーティリティ	◆ *dblsncfg.exe*
Treo 180 用のリスナ	◆ *palm¥Treo180¥lsnT180.prc*
Kyocera 6035 用のリスナ	◆ *palm¥Kyocera6035¥lsnK6035.prc*

QAnywhere アプリケーションの配備

QAnywhere アプリケーションには、クライアント・ファイルとサーバ・ファイルが必要です。

QAnywhere クライアントには、以下にリストされているファイルのほかに、dbmlsync によって使用されるすべてのファイルが必要です。

dbmlsync の詳細については、「Adaptive Server Anywhere Mobile Link クライアントの配備」682 ページを参照してください。

Windows アプリケーション

説明	Windows ファイル
QAnywhere クライアント	◆ *qanywhere.db* ◆ *win32¥iAnywhere.QAnywhere.Client.dll* または *ce¥iAnywhere.QAnywhere.Client.dll* (C# インタフェースの場合のみ必要) ◆ *win32¥qaagent.exe* または *ce¥arm.30¥qaagent.exe* または *ce¥x86.30¥qaagent.exe* ◆ *win32¥qany9.dll* または *ce¥arm.30¥qany9.dll* または *ce¥x86.30¥qany9.dll*
サーバ側	◆ *java¥log4j.jar* ◆ *java¥qaconnector.jar*

UNIX、Linux、Macintosh のアプリケーション

説明	UNIX ファイル
サーバ側	◆ *java/log4j.jar* ◆ *java/qaconnector.jar*

QAnywhere .NET API dll の登録

QAnywhere .NET API dll (*win32¥iAnywhere.QAnywhere.Client.dll*) は、Windows (Windows CE を除く) 上の Global Assembly Cache に登録する必要があります。Global Assembly Cache には、マシンに登録されているすべてのプログラムがリストされています。SQL Anywhere Studio をインストールすると、インストール・プログラムによって登録されます。Windows CE の場合、この DLL を登録する必要はありません。

QAnywhere を配備する場合は、.NET Framework に含まれている gacutil ユーティリティを使用して、QAnywhere .NET API dll (*win32¥iAnywhere.QAnywhere.Client.dll*) を登録してください。

索引

記号

.NET
- Mobile Link .NET API の利点 31
- Mobile Link API リファレンス 381
- Mobile Link データ型 361
- Mobile Link 同期スクリプトの説明 353

.NET CLR
- Mobile Link オプション 259

.NET Mobile Link API
- API リファレンス 381
- 利点 35

.NET からの情報の出力
- Mobile Link .NET 同期論理 366

.NET クラス
- .NET 同期論理のためのインスタンス化 360

.NET での Mobile Link サーバ・エラーの処理
- Mobile Link .NET 同期論理 367

.NET での同期スクリプトの作成
- 説明 353

.NET 同期のサンプル
- Mobile Link .NET 同期論理 377

.NET 同期論理
- .NET クラスのインスタンス化 360
- API 381
- DBCommand 381
- DBConnection 383
- DBConnectionContext 383
- DBParameter 385
- DBParameterCollection 387
- DBRowReader 390
- InOutInteger 399
- LogCallback 391
- LogMessage 391
- MessageType 392
- Mobile Link のパフォーマンス 133
- ServerContext 392
- ServerException 395
- ShutdownCallback 395
- SQLType 396
- サポートされる言語 354
- サンプル 377
- 設定 355
- 説明 31
- デバッグ 368
- メソッド 363

.NET 同期論理の作成
- 説明 360

.NET 同期論理の実行
- 説明 355

.NET 同期論理の設定
- 説明 355

.NET 同期論理のデバッグ
- 説明 368

.NET の同期の方法
- 説明 371

@data オプション
- Mobile Link [dbmlsrv9] 242
- Mobile Link [dbmlstop] 606
- Mobile Link [dbmluser] 608

@EmployeeID 変数
- Mobile Link プライマリ・キー・プールで使用 77

@LastDownload タイムスタンプ変数 60

1 つの SQL 文で複数のエラーが処理される場合
- Mobile Link 317

A

active
- Mobile Link 同期プロパティ 158

Adaptive Server Anywhere
- Mobile Link 統合データベース 46

Adaptive Server Enterprise
begin_connection_autocommit イベント 431
Mobile Link での DDL の使用 431
Mobile Link 同期 47
Mobile Link 統合データベースとして使用 47
Mobile Link におけるデータ型の変換 654
StaticCursorLongColBuffLen 47
Apache
Apache リダイレクタの設定 186
Mobile Link に対するサーブレット・リダイレクタの設定 180
Apache Tomcat
サーブレット・リダイレクタ 180
Apache Tomcat サーバ用のサーブレット・リダイレクタの設定 180
Apache Web サーバ
Apache リダイレクタの設定 186
Apache Web サーバ用の Apache リダイレクタの設定
説明 186
Apache リダイレクタ
設定 186
API
Mobile Link .NET 381
Mobile Link Java 344
API リファレンス
Mobile Link .NET API 381
Mobile Link Java API 344
ASA Mobile Link クライアントの配備 682
ASA 統合データベースの設定 46
auth_status 同期パラメータ
説明 420
authenticate_parameters
接続イベント 417
authenticate_user
接続イベント 420
authenticate_user_hashed
接続イベント 425
AvantGo xi
-a オプション
Mobile Link [dbmlsrv9] 242

B

backlog オプション
HTTP 同期 272, 274
TCP/IP 同期 270
-bc オプション
Mobile Link [dbmlsrv9] 244
begin_connection
接続イベント 429
begin_connection_autocommit
接続イベント 431
begin_download
接続イベント 432
テーブル・イベント 434
begin_download_deletes
テーブル・イベント 437
begin_download_rows
テーブル・イベント 440
begin_publication
接続イベント 443
begin_sync
Mobile Link 同期プロパティ 158
begin_synchronization
接続イベント 447
テーブル・イベント 449
begin_upload
接続イベント 452
テーブル・イベント 455
begin_upload_deletes
テーブル・イベント 458
begin_upload_rows
テーブル・イベント 461
BLOB
ASE からダウンロードされた BLOB 47
BLOB キャッシュ・サイズ
Mobile Link のパフォーマンス 132
-bn オプション
Mobile Link [dbmlsrv9] 244
buffer_size オプション
HTTPS 同期 274
HTTP 同期 272
-b オプション
Mobile Link [dbmlsrv9] 243

C

C# プログラミング言語
Mobile Link .NET でのサポート 354
Mobile Link オプション 259
C++ プログラミング言語
Mobile Link .NET でのサポート 354
certificate_password オプション
HTTPS 同期 274
certificate オプション
HTTPS 同期 274
CHAR データ型
Mobile Link と他の DBMS 44
-classic オプション
Mobile Link [dbmlsrv9] -sl java 261
-classpath オプション
Mobile Link [dbmlsrv9] -sl java 261
CLASSPATH 環境変数
Mobile Link Java 同期論理 323
CLR
Mobile Link オプション 259
-clrConGC オプション
Mobile Link [dbmlsrv9] -sl dnet 259
-clrFlavor オプション
Mobile Link [dbmlsrv9] -sl dnet 259
-clrVersion オプション
Mobile Link [dbmlsrv9] -sl dnet 259
-cn オプション
Mobile Link [dbmlsrv9] 246
Common Language Runtime
Mobile Link オプション 259
completed
Mobile Link 同期プロパティ 158
conflicted_deletes
Mobile Link 同期プロパティ 158
conflicted_inserts
Mobile Link 同期プロパティ 158
connection_retries
Mobile Link 同期プロパティ 158
conflicted_updates
Mobile Link 同期プロパティ 158
contd_timeout オプション
HTTPS 同期 274
HTTP 同期 272
contd_timeout プロトコル・オプション
Mobile Link リダイレクタ 166
-cp オプション
Mobile Link [dbmlsrv9] -sl java 261
-cr オプション
Mobile Link [dbmlsrv9] 246
-ct オプション
Mobile Link [dbmlsrv9] 246
-c オプション
Mobile Link [dbmlsrv9] 245
Mobile Link [dbmluser] 608
Mobile Link [gencert] 612

D

DB2
IBM の識別子最大長 620
Mobile Link 統合データベースとして使用 51
Mobile Link におけるデータ型の変換 656
セッション全体の変数 52
DBCommand
Mobile Link .NET API 381
DBConnection
Mobile Link .NET API 383
DBConnectionContext
Mobile Link .NET API 383
Mobile Link Java API 344
コンストラクタ 362
dbmlmon.exe
Mobile Link のモニタ 143
dbmlmon
起動 145
dbmlsrv9
スクリプト生成の自動化 289
Notifier 253
QAnywhere 251
オプション 236
起動 13
出力ログにエラー・コンテキストをレポート 254
dbmlsrv9 オプション
アルファベット順のリスト 242

dbmlstop ユーティリティ
オプション 606
構文 606
使用 14
dbmluser ユーティリティ
オプション 608
構文 608
DBMS 依存の同期スクリプト
Mobile Link 43
DBParameter
Mobile Link .NET API 385
DBParameterCollection
Mobile Link .NET API 387
DBRowReader
Mobile Link .NET API 390
DDL
Mobile Link リモート・データベース 100
-dd オプション
Mobile Link [dbmlsrv9] 247
DECIMAL データ型
Mobile Link と ASE 48
-dl オプション
Mobile Link [dbmlsrv9] 248
Mobile Link [dbmluser] 608
-DMLStartClasses
Java ユーザ定義起動クラス 333
Mobile Link [dbmlsrv9] -sl java 261
download
Mobile Link 同期プロパティ 158
download_bytes
Mobile Link 同期プロパティ 158
download_cursor
子テーブルの分割 69
重複のある分割 67
ストアド・プロシージャ・コールの使用 97
ストアド・プロシージャ・コールの使用例 97
切断分割 65
タイムスタンプベースの同期 61
テーブル・イベント 464
パフォーマンス 134
download_cursor スクリプトの作成
Mobile Link 309
download_delete_cursor
子テーブルの分割 69
重複のある分割 67
ストアド・プロシージャ・コールの使用 97
ストアド・プロシージャ・コールの使用例 97
切断分割 65
説明 311
テーブル・イベント 468
パフォーマンス 134
download_delete_cursor スクリプトの作成
Mobile Link 311
download_delete_cursor スクリプトを使用したローの削除 311
download_delete_cursor タイムスタンプベースの同期 61
download_deleted_rows
Mobile Link 同期プロパティ 158
download_errors
Mobile Link 同期プロパティ 158
download_fetched_rows
Mobile Link 同期プロパティ 158
download_filtered_rows
Mobile Link 同期プロパティ 158
download_statistics
接続イベント 472
テーブル・イベント 476
download_warnings
Mobile Link 同期プロパティ 158
-ds オプション
Mobile Link [dbmlsrv9] 248
duration
Mobile Link 同期プロパティ 158
-d オプション
Mobile Link [dbmlsrv9] 247
Mobile Link [dbmlsrv9] -sl java 261
Mobile Link [dbmluser] 608

E

ecc_tls
dbmlsrv9 -x 271

end_connection
接続イベント 480
end_download
接続イベント 482
テーブル・イベント 485
end_download_deletes
テーブル・イベント 488
end_download_rows
テーブル・イベント 491
end_publication
接続イベント 494
end_sync
Mobile Link 同期プロパティ 158
end_synchronization
接続イベント 497
テーブル・イベント 499
end_upload
接続イベント 502
テーブル・イベント 504
end_upload_deletes
テーブル・イベント 508
end_upload_rows
テーブル・イベント 511
-et オプション
Mobile Link [dbmlsrv9] 249
example_download_cursor
説明 292
example_upload_cursor
説明 292
テーブル・イベント 514
example_upload_delete
テーブル・イベント 515
example_upload_insert
テーブル・イベント 516
example_upload_update
テーブル・イベント 517
-e オプション
Mobile Link [dbmlsrv9] 248

F

FIPS
Mobile Link 同期サーバ通信 273
Mobile Link 同期サーバの -x オプション 269
説明 207
FIPS 140-2 承認
説明 207
-fr オプション
Mobile Link [dbmlsrv9] 250
-f オプション
Mobile Link [dbmlsrv9] 249
Mobile Link [dbmlstop] 606
Mobile Link [dbmluser] 608

G

gencert ユーティリティ
オプション 612
構文 612
getServerContext メソッド
DBConnectionContext クラス (Mobile Link Java API) 345
DBConnectionContext クラス (MobiLink .NET API) 384
GLOBAL_DATABASE_ID オプション
Mobile Link 設定 73
GLOBAL_DATABASE_ID 値の設定
Mobile Link 73
GUID xi

H

handle_error
接続イベント 518
同期スクリプト 316
handle_odbc_error
接続イベント 522
host オプション
HTTPS 同期 274
HTTP 同期 272
TCP/IP 同期 270
host プロトコル・オプション
Mobile Link リダイレクタ 166

-hotspot オプション
Mobile Link [dbmlsrv9] -sl java 261
HTTP
dbmlsrv9 -x コマンド・ライン・オプション 272
Mobile Link 同期サーバ 269
httpd.conf
Apache ネイティブ・リダイレクタ 186
HTTPS
dbmlsrv9 -x コマンド・ライン・オプション 273
Mobile Link 同期サーバ 269
Mobile Link トランスポート・レイヤ・セキュリティ 225
HTTPS_FIPS
dbmlsrv9 -x コマンド・ライン・オプション 273
Mobile Link 同期サーバ 269
HTTPS におけるトランスポート・レイヤ・セキュリティ
Mobile Link 225
TCP/IP におけるトランスポート・レイヤ・セキュリティ
Mobile Link 227
-h オプション
Mobile Link [dbmlstop] 606

I

iAnywhere Solutions ODBC ドライバ
サポート 669
iaredirect.dll
ISAPI リダイレクタの設定 177
NSAPI リダイレクタの設定 172
IBM DB2
Mobile Link 統合データベースとして使用 51
Mobile Link におけるデータ型の変換 656
識別子最大長 620
セッション全体の変数 52
IBM DB2 統合データベースの設定 51
ignored_deletes
Mobile Link 同期プロパティ 158
ignored_inserts
Mobile Link 同期プロパティ 158
ignored_updates
Mobile Link 同期プロパティ 158
ignore オプション
TCP/IP 同期 270
IIS
ISAPI 用の設定 177
InOutByteArray
Mobile Link Java API 345
InOutInteger
Mobile Link Java API 345
InOutString
Mobile Link Java API 346
iPlanet
NSAPI リダイレクタの設定 172
ISAPI リダイレクタ
設定 177
呼び出し 177

J

Java
Mobile Link Java API の利点 31
Mobile Link Java API リファレンス 344
Mobile Link データ型 327
Mobile Link 用の同期スクリプト 321
同期論理 31
Javadoc
Mobile Link 344
Java Mobile Link API
利点 34
Java VM
Mobile Link オプション 261
Java クラス
Java 同期論理のためのインスタンス化 326
Java クラスのデバッグ
Mobile Link Java 同期論理 330
Java での Mobile Link サーバ・エラーの処理
Mobile Link Java 同期論理 331
Java 同期スクリプトの作成
Mobile Link Java 同期論理の例 338

Java 同期の例
Mobile Link Java 同期論理 337
Java 同期論理
API 344
DBConnectionContext 344
InOutByteArray 345
InOutInteger 345, 351
InOutString 346
Java クラスのインスタンス化 326
LogListener 346
LogMessage 346
Mobile Link サーバのコマンド・ライン指定 324
Mobile Link のパフォーマンス 133
ServerContext 347
ServerException 350
ShutdownListener 351
設定 323, 329
説明 31
例 337
Java 同期論理と SQL 同期論理
Mobile Link のパフォーマンス 133
Java 同期論理の作成
説明 326
Java 同期論理の実行
説明 323
Java 同期論理の設定
説明 323
Java による同期スクリプトの作成
説明 321
-jrepath オプション
Mobile Link [dbmlsrv9] -sl java 261

K

keep partial download 同期パラメータ
再起動可能なダウンロード 94

L

last_download_timestamp
スクリプト・パラメータ 297
liveness_timeout オプション
TCP/IP 同期 271
LOG_LEVEL
リダイレクタのプロパティ 169
LogCallback
Mobile Link .NET API 391
LogListener
Mobile Link Java API 346
LogMessage
Mobile Link .NET API 391
Mobile Link Java API 346
LONG データ型
Oracle の同期 659

M

M-Business Anywhere
同期の設定 191
M-Business Anywhere リダイレクタ
設定 191
MessageType
Mobile Link .NET API 392
Microsoft SQL Server
Mobile Link 統合データベースとして使用 54
Mobile Link におけるデータ型の変換 660
Microsoft SQL Server 統合データベースの設定 54
Microsoft Web サーバ用の ISAPI リダイレクタの設定
説明 177
ML
リダイレクタのプロパティ 169
ml_add_connection_script ストアド・プロシージャ
SQL 構文 594
ml_add_dnet_connection_script ストアド・プロシージャ
SQL 構文 597
ml_add_dnet_table_script ストアド・プロシージャ
SQL 構文 598
ml_add_java_connection_script ストアド・プ

ロシージャ
SQL 構文 599
ml_add_java_table_script ストアド・プロシージャ
SQL 構文 600
ml_add_property ストアド・プロシージャ
SQL 構文 602
ml_add_table_script ストアド・プロシージャ
SQL 構文 595
ML_CLIENT_TIMEOUT
リダイレクタのプロパティ 169
ml_connection_script
Mobile Link システム・テーブル 621
ml_device
Mobile Link システム・テーブル 622
ml_device_address
Mobile Link システム・テーブル 624
ml_listening
Mobile Link システム・テーブル 626
ml_property
Mobile Link システム・テーブル 628
ml_qa_delivery
Mobile Link システム・テーブル 630
ml_qa_delivery_client
Mobile Link システム・テーブル 631
ml_qa_global_props
Mobile Link システム・テーブル 632
ml_qa_global_props_client
Mobile Link システム・テーブル 633
ml_qa_notifications
Mobile Link システム・テーブル 634
ml_qa_repository
Mobile Link システム・テーブル 635
ml_qa_repository_client
Mobile Link システム・テーブル 636
ml_qa_repository_content_client
Mobile Link システム・テーブル 638
ml_qa_repository_props
Mobile Link システム・テーブル 639
ml_qa_repository_props_client
Mobile Link システム・テーブル 640
ml_qa_repository_staging
Mobile Link システム・テーブル 641
ml_qa_status_staging
Mobile Link システム・テーブル 643
ml_script
Mobile Link システム・テーブル 644
ml_script_version
Mobile Link システム・テーブル 645
ml_scripts_modified
Mobile Link システム・テーブル 646
ml_subscription
Mobile Link システム・テーブル 647
ml_table
Mobile Link システム・テーブル 649
ml_table_script
Mobile Link システム・テーブル 650
ml_user
Mobile Link システム・テーブル 651
Mobile Link ユーザ認証 [dbmluser] 608
ml_username
スクリプト・パラメータ 297
-MLAutoLoadPath オプション
Mobile Link [dbmlsrv9] -sl dnet 259
説明 373
mlDomConfig.xml
説明 375
-MLDomConfigFile オプション
Mobile Link [dbmlsrv9] -sl dnet 259
説明 373
mlMonitorSettings
Mobile Link モニタの設定 152
mlscript.jar
Mobile Link の Java 同期論理 323
-MLStartClasses
.NET ユーザ定義起動クラス 364
Mobile Link [dbmlsrv9] -sl dnet 259
Mobile Link .NET API リファレンス
アルファベット順のリスト 381
Mobile Link
.NET 同期論理 353
dbmlsrv9 オプション 235
dbmlsrv9 の接続パラメータ 269
Java 同期論理 321

Mobile Link サーバの停止　14
ODBC ドライバのサポート　670
Web サーバの設定　161
アーキテクチャ　10
アプリケーションの配備　673
イベントのアルファベット順のリスト　401
イベントの概要　402
開発のヒント　59
起動　13
機能　4
競合の解決　80
クライアント　17
現在のセッション以外での実行　195
システム・テーブル　619
処理の概要　18
スクリプト　285
ストアド・プロシージャ　593
データ型　653
同期の基本　9
同期の方法　57
同期論理の作成オプション　31
統合データベース　39
トランスポート・レイヤ・セキュリティ　205
パフォーマンス　129
ファイルベースのダウンロード　105
複数の同期サーバ　163
文字セットの考慮事項　663
モニタ　143
モニタの接続パラメータ　145
ユーティリティ　605
リダイレクタ　161
Mobile Link Java API リファレンス
アルファベット順のリスト　344
Mobile Link アップロード・ストリーム
処理　26
定義　18
Mobile Link アプリケーションの配備
説明　673
Mobile Link イベント
説明　20
リスト　401
Mobile Link イベントの概要　402
Mobile Link クライアント
説明　17
配備　682
Mobile Link クライアントとサーバのリダイレクタ設定　166
Mobile Link サーバ起動時のトラブルシューティング
説明　204
Mobile Link サーバの配備　675
Mobile Link システム・テーブル
ml_connection_script　621
ml_device　622
ml_device_address　624
ml_listening　626
ml_property　628
ml_qa_delivery_client　631
ml_qa_global_props　632
ml_qa_global_props_client　633
ml_qa_notifications　634
ml_qa_repository　635
ml_qa_repository_client　636
ml_qa_repository_content_client　638
ml_qa_delivery　630
ml_qa_repository_props　639
ml_qa_repository_props_client　640
ml_qa_repository_staging　641
ml_qa_status_staging　643
ml_script　644
ml_script_version　645
ml_scripts_modified　646
ml_subscription　647
ml_table　649
ml_table_script　650
ml_user　651
説明　619
統合データベースに作成　42
Mobile Link 証明書生成ユーティリティ [gencert]
構文　612
Mobile Link 証明書読み込みユーティリティ [readcert]
構文　611
Mobile Link スクリプト
説明　20

Mobile Link 接続
デバッグ 204
Mobile Link ダウンロード・ストリーム
定義 18
Mobile Link 停止ユーティリティ [dbmlstop]
構文 606
Mobile Link データ型
.NET と SQL 361
Java と SQL 327
Mobile Link 同期
.NET クラスの作成 363
.NET 同期論理 353
Java クラスの作成 329
Java 同期論理 321
Web サーバの設定 161
イベントの概要 402
クライアント 17
再起動可能なダウンロード 92
統合データベース 39
パフォーマンス 129
ファイルベースのダウンロード 105
Mobile Link 同期サーバ
オプション 236
起動 13
構文 236
説明 13
停止ユーティリティ 606
配備 675
複数のインスタンス 169
Mobile Link 同期サーバ動作のロギング
説明 15
Mobile Link 同期サーバのオプション
説明 235
Mobile Link 同期サーバの確認
Mobile Link トランスポート・レイヤ・セキュリティ 222
Mobile Link 同期サーバの実行
説明 13
Mobile Link 同期サーバの停止 14
Mobile Link 同期処理
説明 4
Mobile Link 同期スクリプト
.NET クラスの構成 362
.NET クラスの作成 363
.NET でのデータベース・トランザクションの保存 361
Java クラスの構成 328
Java クラスの作成 329
Java クラスのデバッグ 330
Java でのデータベース・トランザクションの保存 327
スクリプトのアルファベット順のリスト 401
説明 285
データベース・トランザクションと .NET クラス 361
データベース・トランザクションと Java クラス 327
Mobile Link 同期について 3
Mobile Link 同期論理
.NET 353
.NET と SQL のデータ型 361
Java 321
Java と SQL に対応するデータ型 327
スクリプトのアルファベット順のリスト 401
スクリプトの作成 285
同期の方法 57
Mobile Link 統合データベース
IBM DB2 51
Mobile Link 統合データベースとしての ASA 46
Mobile Link 統合データベースとしての ASE 47
Oracle 49
SQL Server 54
説明 39
Mobile Link トランスポート・レイヤ・セキュリティ
概要 206
説明 205
Mobile Link の機能
説明 4
Mobile Link のセキュリティ
カスタム・ユーザ認証 425
Mobile Link の世代番号
ファイルベースのダウンロード 115

Mobile Link の統計のプロパティ
Mobile Link モニタ 158
Mobile Link のパフォーマンス
主要な要因 135
説明 129
モニタ 141
Mobile Link のパフォーマンスに影響を与える主要な要因 135
Mobile Link のパフォーマンスのチューニング 136
Mobile Link のパフォーマンスのモニタ
概要 141
Mobile Link モニタ
MS Excel で表示 154
ウォッチの指定 156
ウォッチ・マネージャ 156
オプション 152
[概要] ウィンドウ枠 152
起動 145
使用 148
[詳細テーブル] ウィンドウ枠 149
説明 143
[チャート] ウィンドウ枠 150
データの保存 154
デフォルトのリストア 152
統計のプロパティ 158
プロパティ 153
ユーザ・インタフェース 148
Mobile Link モニタの起動
説明 145
Mobile Link モニタの使用 148
Mobile Link ユーザ
Mobile Link ユーザ認証 [dbmluser] 608
Mobile Link ユーザ認証ユーティリティ [dbmluser]
構文 608
Mobile Link ユーザ名
スクリプト・パラメータ 297
Mobile Link ユーティリティ
Mobile Link 証明書ジェネレータ [gencert] 612
Mobile Link 証明書読み込み [readcert] 611
Mobile Link 停止ユーティリティ [dbmlstop] 606
Mobile Link ユーザ認証 [dbmluser] 608
Mobile Link リモート・データベース
スキーマの変更 100
mod_iaredirect.dll
Apache リダイレクタの設定 186
M-Business Anywhere リダイレクタの設定 191
mod_iaredirect.so
M-Business Anywhere リダイレクタの設定 191
modify_error_message
接続イベント 525
modify_last_download_timestamp
接続イベント 527
modify_next_last_download_timestamp
接続イベント 530
modify_user
接続イベント 533
-m オプション
Mobile Link [dbmlsrv9] 251

N

Netscape/Sun Web サーバ用の NSAPI リダイレクタの設定
説明 172
Netscape Web サーバ
NSAPI リダイレクタの設定 172
new_row_cursor
テーブル・イベント 536
-notifier オプション
Mobile Link [dbmlsrv9] 253
NSAPI リダイレクタ
設定 172
NUMERIC データ型
Mobile Link と ASE 48

O

ODBC

Mobile Link での複数のエラー 317
ODBC ドライバ
Mobile Link でサポートされる ODBC ドライバ 670
Mobile Link 文字セットの変換 666
old_row_cursor
テーブル・イベント 539
-on オプション
Mobile Link [dbmlsrv9] 255
-oq オプション
Mobile Link [dbmlsrv9] 256
Oracle
Mobile Link 同期でのシーケンス 49
Mobile Link 統合データベースとして使用 49
Mobile Link におけるデータ型の変換 658
ODBC 設定 659
データ型 658
同期する LONG データ 659
Oracle 統合データベースの設定 49
-os オプション
Mobile Link [dbmlsrv9] 256
Mobile Link [dbmluser] 608
-ot オプション
Mobile Link [dbmlsrv9] 257
Mobile Link [dbmluser] 608
-o オプション
Mobile Link [dbmlsrv9] 254
Mobile Link [dbmluser] 608

P

partial download retained 同期パラメータ
再起動可能なダウンロード 94
-pc オプション
Mobile Link [dbmluser] 608
port オプション
HTTPS 同期 274
HTTP 同期 272
TCP/IP 同期 271
port プロトコル・オプション
Mobile Link リダイレクタ 166
ファイアウォール経由の同期 166
preload_upload
Mobile Link 同期プロパティ 158
prepare_for_download
Mobile Link 同期プロパティ 158
接続イベント 542
-ps オプション
Mobile Link [dbmlsrv9] 257
-p オプション
Mobile Link [dbmluser] 608

Q

QAnywhere
Mobile Link システム・テーブル 619
配備 686
プロパティ 251
QAnywhere クライアントの配備 686
-q オプション
Mobile Link [dbmlsrv9] 258
Mobile Link [dbmlstop] 606
Mobile Link [gencert] 612

R

-rd オプション
Mobile Link [dbmlsrv9] 258
readcert ユーティリティ
オプション 611
構文 611
redirector.config
設定 169
例 171
ロケーション 169
report_error
構文 316
接続イベント 544
report_odbc_error
接続イベント 547
reqtool
Mobile Link トランスポート・レイヤ・セキュリティ 215

使用 215
resolve_conflict
使用 83
テーブル・イベント 550
resume partial download 同期パラメータ
再起動可能なダウンロード 94
rsa_tls
dbmlsrv9 -x 271
rsa_tls_fips
dbmlsrv9 -x 271
RSA 証明書
生成 612
読み込み 611
-r オプション
Mobile Link [dbmlsrv9] 258
Mobile Link [gencert] 612

S

ServerContext
Mobile Link .NET API 392
Mobile Link Java API 347
ServerException
Mobile Link .NET API 395
Mobile Link Java API 350
-server オプション
Mobile Link [dbmlsrv9] -sl java 261
ShutdownCallback
Mobile Link .NET API 395
ShutdownListener
Mobile Link Java API 351
-sl dnet オプション
-MLAutoLoadPath の使用 358
-MLDomConfigFile の使用 373
Mobile Link [dbmlsrv9] 259
ユーザ定義起動クラス 364
SLEEP
リダイレクタのプロパティ 169
-sl java オプション
Mobile Link [dbmlsrv9] 261
ユーザ定義起動クラス 333
SQL Anywhere Studio
マニュアル xii
SQL Server
Mobile Link 統合データベースとして使用 54
SQLType
Mobile Link .NET API 396
SQL 構文
Mobile Link サーバ [dbmlsrv9] 236
SQL データ型と .NET データ型
Mobile Link .NET 同期論理 361
SQL データ型と Java データ型
説明 327
SQL 同期論理
Mobile Link 285
Mobile Link のパフォーマンス 133
代替手段 31
SSL (セキュア・ソケット・レイヤ)
Mobile Link 同期 205
start_time
Mobile Link 同期プロパティ 158
StaticCursorLongColBuffLen
ASE 47
STOP SYNCHRONIZATION DELETE 文
使用方法 311
Sun One
NSAPI リダイレクタの設定 172
Sun Web サーバ
NSAPI リダイレクタの設定 172
Sybase Adaptive Server Enterprise
Mobile Link におけるデータ型の変換 654
Sybase ASE 統合データベースの設定 47
sync.conf
M-Business Anywhere リダイレクタ 191
sync
Mobile Link 同期プロパティ 158
sync_deadlocks
Mobile Link 同期プロパティ 158
sync_errors
Mobile Link 同期プロパティ 158
sync_tables
Mobile Link 同期プロパティ 158
sync_warnings
Mobile Link 同期プロパティ 158
syncasa.sql

説明 46
syncase.sql
説明 47
syncase125.sql
説明 47
syncdb2long.sql
説明 51
synchronization_statistics
接続イベント 553
テーブル・イベント 556
SynchronizationException
Mobile Link .NET API 399
Mobile Link Java API 351
syncmss.sql
説明 54
syncora.sql
説明 49
-s オプション
Mobile Link [dbmlsrv9] 259
Mobile Link [gencert] 612

T

TCP/IP
ASA クライアントの Mobile Link TLS 227
dbmlsrv9 -x コマンド・ライン・オプション 270
Mobile Link 同期サーバ 269
HTTPS
Ultra Light クライアントの Mobile Link TLS 229
TCP/IP
Ultra Light クライアントの Mobile Link TLS 231
time_statistics
接続イベント 559
テーブル・イベント 562
TLS xi
Mobile Link 205
TLS を使用する Adaptive Server Anywhere クライアント
Mobile Link 225
Tomcat
サーブレット・リダイレクタの設定 180
リダイレクタのサポートされているバージョン 180
trusted_certificates パラメータ
Mobile Link トランスポート・レイヤ・セキュリティ 224
-tt オプション
Mobile Link [dbmlsrv9] 263
-t オプション
Mobile Link [dbmlsrv9] 263
Mobile Link [dbmlstop] 606

U

UDB
Mobile Link 統合データベースとして使用 51
-ud オプション
Mobile Link [dbmlsrv9] 264
ULRollbackPartialDownload 関数
再起動可能なダウンロード 94
Ultra Light
Mobile Link サンプル・スクリプト 292
Mobile Link トランスポート・レイヤ・セキュリティ 228
配備 685
Ultra Light Mobile Link クライアントの配備 685
Ultra Light 用のサンプル・スクリプト
Mobile Link 292
unknown_timeout オプション
HTTPS 同期 275
unknown_timeout プロトコル・オプション
ファイアウォール経由の同期 166
UPDATE の競合
Mobile Link 80
upload
Mobile Link 同期プロパティ 158
upload_bytes
Mobile Link 同期プロパティ 158
upload_cursor
テーブル・イベント 565

upload_deadlocks
Mobile Link 同期プロパティ 158
upload_delete
.NET 同期論理での使用 371
テーブル・イベント 568
upload_deleted_rows
Mobile Link 同期プロパティ 158
upload_delete スクリプトの作成
Mobile Link 306
upload_errors
Mobile Link 同期プロパティ 158
upload_fetch
競合検出 81
テーブル・イベント 570
upload_fetch スクリプトの作成
Mobile Link 306
upload_insert
.NET 同期論理での使用 371
テーブル・イベント 572
upload_inserted_rows
Mobile Link 同期プロパティ 158
upload_insert スクリプトの作成
Mobile Link 305
upload_new_row_insert
テーブル・イベント 575
ユーザ名の格納 89
upload_old_row_insert
テーブル・イベント 577
ユーザ名の格納 89
upload_statistics
接続イベント 579
テーブル・イベント 583
upload_update
.NET 同期論理での使用 371
競合検出 81
使用 86
テーブル・イベント 587
upload_updated_rows
Mobile Link 同期プロパティ 158
upload_update スクリプトの作成
Mobile Link 306
upload_warnings
Mobile Link 同期プロパティ 158
-urc オプション
Mobile Link のパフォーマンスの利点 134
url_suffix オプション
HTTPS 同期 275
HTTP 同期 273
url_suffix プロトコル・オプション
Mobile Link リダイレクタ 166
user
Mobile Link 同期プロパティ 158
-us オプション
Mobile Link [dbmlsrv9] 265
UUID
Mobile Link 同期アプリケーション 70
UUID を使用したユニークなプライマリ・キーの管理
Mobile Link 70
-u オプション
Mobile Link [dbmlsrv9] 264
Mobile Link [dbmluser] 608

V

VARCHAR データ型
Mobile Link と他の DBMS 44
-verbose オプション
Mobile Link [dbmlsrv9] -sl java 261
verify_upload
Mobile Link 同期プロパティ 158
version
Mobile Link 同期プロパティ 158
version オプション
HTTPS 同期 275
HTTP 同期 273
Visual Basic
Mobile Link .NET でのサポート 354
-v オプション
Mobile Link [dbmlsrv9] 265
Mobile Link [dbmlsync] のパフォーマンス 133

W

Web サーバ
ISAPI Microsoft の同期設定 177
Mobile Link クライアント 166
Mobile Link の設定オプション 163
Mobile Link リダイレクタ 161
同期の設定 169
同期用の Apache の設定 186
同期用の M-Business Anywhere の設定 191
同期用の NSAPI の設定 172
Web サーバを経由した同期
Mobile Link 161
Web サーバを使用した場合のオプション
Mobile Link 163
Windows 版 Mobile Link 同期サーバをサービスとして実行する 197
worker
Mobile Link 同期プロパティ 158
-wu オプション
Mobile Link [dbmlsrv9] 268
-w オプション
Mobile Link [dbmlsrv9] 266
Mobile Link [dbmlstop] 606

X

X509 証明書
生成 612
読み込み 611
Xusage.txt
ロケーション 261
-x オプション
Mobile Link [dbmlsrv9] 269
Mobile Link [dbmlsrv9] -sl java 261

Z

-za オプション
Mobile Link [dbmlsrv9] 275
-ze オプション
Mobile Link [dbmlsrv9] 277
-zp オプション
Mobile Link [dbmlsrv9] 278
-zs オプション
Mobile Link [dbmlsrv9] 278
-zt オプション
Mobile Link [dbmlsrv9] 279
-zu オプション
Mobile Link [dbmlsrv9] 279
-zwd オプション
Mobile Link [dbmlsrv9] 280
-zwe オプション
Mobile Link [dbmlsrv9] 281
-zw オプション
Mobile Link [dbmlsrv9] 279

あ

アイコン
マニュアルで使用 xvii
アセンブリ
Mobile Link .NET 同期論理での配置 355
Mobile Link での実装 373
アセンブリのロード
Mobile Link .NET 同期論理 373
新しいリモートの同期
Mobile Link ファイルベースのダウンロード 109
アップグレード
Mobile Link リモート・データベースのスキーマ 100
アップロード・イベント
Mobile Link 同期 410
説明 305
アップロード・キャッシュ・サイズ
Mobile Link のパフォーマンス 132
アップロード・ストリーム
Mobile Link トランザクション 23
Mobile Link の処理 26
イベント 305
定義 18
アップロード・ストリームの処理方法 26
アップロード専用の同期

説明　30
アップロード専用の同期とダウンロード専用の同期
説明　30
アップロード中のイベント
説明　410
ローをアップロードするスクリプトの作成　305
アプリケーション
Mobile Link アプリケーションの配備　673
Mobile Link スクリプトの区別　298
アプリケーションのアップグレード
複数の Mobile Link スクリプト・バージョンの使用　298
暗号
Mobile Link トランスポート・レイヤ・セキュリティ　205
暗号化
Mobile Link　205
暗号方式
Mobile Link パブリック・キー　205

い

一度に複数のサービスを実行する
Mobile Link　201
イベント
Mobile Link　20, 401
Mobile Link イベントの概要　286
Mobile Link 同期の説明　402
Mobile Link の説明　285
インデックス
Mobile Link のパフォーマンス　134

う

ウィザード
[サービス追加]　197
[接続スクリプト追加]　301
[同期テーブル・スクリプト追加]　302
[同期テーブル追加]　302
[バージョン追加]　299

え

エラー
Mobile Link modify_error_message 接続イベント　525
Mobile Link 同期中の処理　316
記録　316
エラー処理
Mobile Link 同期中　316
エラーのレポート
Mobile Link 同期　316
エラー・ログ
Mobile Link サーバ [dbmlsrv9]　248
エラーを処理するスクリプトの作成
Mobile Link　316
エンタープライズ・ルート証明書
Mobile Link トランスポート・レイヤ・セキュリティ　209, 211, 213
エンタープライズ・ルート証明書の作成
Mobile Link トランスポート・レイヤ・セキュリティ　213

お

オブジェクト
Mobile Link .NET API　381
Mobile Link Java API　344
オプション
dbmlsrv9　236
Mobile Link サーバ [dbmlsrv9]　236
Mobile Link 証明書ジェネレータ [gencert]　612
Mobile Link 証明書読み込み [readcert]　611
Mobile Link 停止ユーティリティ [dbmlstop]　606
Mobile Link ユーザ認証 [dbmluser]　608
[オプション] ダイアログ
Mobile Link モニタ　152

か

カーソル・スクリプト
定義 295
解決
Mobile Link 競合 80
Mobile Link 競合解決 80
開発のヒント
Mobile Link 同期 59
[概要] ウィンドウ枠
Mobile Link モニタ 152
カスケード削除
Mobile Link 同期中 27
カスタム検証
Mobile Link ファイルベースのダウンロード 116
カラム
Mobile Link リモート・データベースへの追加 101
完全なイベント・モデル
Mobile Link 402
簡単な同期スクリプト 288

き

キー・プール
Mobile Link 同期アプリケーション 76
キー・プールを使用したユニークなプライマリ・キーの管理
Mobile Link 76
擬似コード
Mobile Link イベント 402
規則
表記 xv
起動
Mobile Link 同期サーバ 13
Mobile Link モニタ [dbmlmon] 145
Notifier 253
起動クラス
.NET の MLStartClasses オプション 259
Java の DMLStartClasses オプション 261
Mobile Link .NET 同期論理 364
Mobile Link Java 同期論理 333
基本的な規則
Mobile Link 59
キャッシュ・サイズ
Mobile Link の BLOB パフォーマンス 132
Mobile Link のアップロード・パフォーマンス 132
Mobile Link のダウンロード・パフォーマンス 132
競合
Mobile Link 80
Mobile Link 競合解決 80
Mobile Link のデフォルト動作 80
Mobile Link のパフォーマンス 130
Mobile Link のパフォーマンスの説明 136
Mobile Link 文ベースのアップロード 81
競合解決
Mobile Link 80
Mobile Link 競合検出 81
Mobile Link での強制 88
Mobile Link のデフォルト動作 80
Mobile Link 文ベースのアップロード 81
resolve_conflict スクリプト 83
upload_update スクリプト 86
ユーザ固有の論理 89
競合検出
Mobile Link 81
Mobile Link 文ベースのアップロード 81
競合の解決
Mobile Link 80
resolve_conflict スクリプト 83
upload_update スクリプト 86
競合の検出
Mobile Link 文ベースのアップロード 81
競合を検出する方法
Mobile Link 81
強制的な競合解決
Mobile Link 88
共有アセンブリ
Mobile Link での実装 373
強力な暗号化
Mobile Link 205

く

クライアント
Mobile Link 同期 17
パブリック証明書を信用するように設定 218
クライアント・イベント・フック・プロシージャ xi
クライアント・セキュリティ・パラメータ
Mobile Link トランスポート・レイヤ・セキュリティ 224
クラス・インスタンス
Java 同期論理 326
Mobile Link .NET 同期論理 360
グローバル・アセンブリ・キャッシュ
Mobile Link での実装 373
グローバル・オートインクリメント
Mobile Link 用の GLOBAL_DATABASE_ID の設定 73
Mobile Link 用のユニークな値の生成 71
アルゴリズム 74
宣言 73
グローバル・オートインクリメントを使用したユニークなプライマリ・キーの管理
Mobile Link 71
グローバル証明書
Mobile Link トランスポート・レイヤ・セキュリティにおける reqtool の使用 215
Mobile Link トランスポート・レイヤ・セキュリティのサーバ証明書として使用 216
グローバル証明書をサーバ証明書として使用する
Mobile Link トランスポート・レイヤ・セキュリティ 216
グローバル署名証明書
Mobile Link トランスポート・レイヤ・セキュリティ 214

け

現在のセッション以外での Mobile Link の実行
説明 195
検証
Mobile Link ファイルベースのダウンロード 112
検証チェック
Mobile Link ファイルベースのダウンロード 112

こ

高可用性
Mobile Link リダイレクタ 162
構文
Mobile Link サーバ [dbmlsrv9] 236
Mobile Link 証明書ジェネレータ [gencert] 612
Mobile Link 証明書読み込み [readcert] 611
Mobile Link スクリプト 401
Mobile Link ストアド・プロシージャ 593
Mobile Link 停止ユーティリティ [dbmlstop] 606
Mobile Link 同期ユーティリティ 605
Mobile Link ユーザ認証 [dbmluser] 608
子テーブルの分割
Mobile Link 69
コマンド・ライン
dbmlsrv9 の起動 236
コマンド・ライン・ユーティリティ
Mobile Link 証明書ジェネレータ [gencert] 612
Mobile Link 停止ユーティリティ [dbmlstop] 606
Mobile Link ユーザ認証 [dbmluser] 608
readcert の構文 611
コンストラクタ
Mobile Link .NET 同期論理 362
Mobile Link Java 同期論理 328

さ

サーバ
Mobile Link 同期 [dbmlsrv9] 13
サーバ証明書

Mobile Link トランスポート・レイヤ・セキュリティにおけるグローバル証明書の使用 216
サーバ・ストアド・プロシージャ
Mobile Link 593
サーバ認証
Mobile Link トランスポート・レイヤ・セキュリティ 222
サービス
Mobile Link の実行 195
依存性 201
サービスとしての Mobile Link の実行 197
削除 197
設定 198
複数実行 201
[サービス追加] ウィザード
使用 197
サービスの依存性
Mobile Link 201
サービスの追加、修正、削除
Mobile Link 197
サーブレット
Apache Web サーバ用のリダイレクタのインストール 180
サーブレット・リダイレクタ
Apache Tomcat 180
Apache Web サーバ 180
サーブレット・リダイレクタの設定
Apache Web サーバ 180
再起動可能なダウンロード
Mobile Link 92
最終ダウンロード・タイムスタンプ
modify_last_download_timestamp 接続イベント 527
modify_next_last_download_timestamp 接続イベント 530
説明 60
最終変更カラム
説明 60
削除
Mobile Link .NET 接続スクリプト 597
Mobile Link .NET テーブル・スクリプト 598
Mobile Link Java 接続スクリプト 599
Mobile Link Java テーブル・スクリプト 600
Mobile Link SQL 接続スクリプト 594
Mobile Link SQL テーブル・スクリプト 595
Mobile Link のプロパティ 602
Mobile Link リモート・データベースのロー 311
削除の処理
同期の方法 91
作成
.NET 同期論理 353
Java 同期論理 321
Mobile Link 統合データベース 42
Mobile Link ファイルベースのダウンロード用のダウンロード・ファイル 108
新しい証明書 612
ファイル定義データベース 107
サポート
ニュースグループ xx
サポートされている DBMS スクリプトの作成方法 43
Mobile Link 52
参照整合性
Mobile Link 同期中 27
参照整合性と同期
Mobile Link クライアント 27
サンプル
.NET 同期論理 377
サンプル・スクリプト
生成 290
サンプル・スクリプトの生成
Mobile Link 290
サンプル同期スクリプトの生成 290
サンプルのドメイン設定ファイル
説明 375

し

シーケンス
Mobile Link 同期におけるプライマリ・キーの一意性 49
識別子
IBM DB2 の最大長 620

自己署名証明書
Mobile Link トランスポート・レイヤ・セキュリティ 209
Mobile Link トランスポート・レイヤ・セキュリティ用に作成 210
自己署名証明書の新規作成
Mobile Link トランスポート・レイヤ・セキュリティ 210
自己署名証明書の設定
Mobile Link トランスポート・レイヤ・セキュリティ 210
システム・テーブル
Mobile Link 同期 620
Mobile Link 統合データベースに作成 42
システム・プロシージャ
Mobile Link ストアド・プロシージャ 593
失敗したダウンロード
同期の方法 92
失敗したダウンロードの再開
Mobile Link 92
失敗したダウンロードの処理
同期の方法 92
自動検証
Mobile Link ファイルベースのダウンロード 113
重複
分割 65
照合順
Mobile Link 同期 664
[詳細テーブル] ウィンドウ枠
Mobile Link モニタ 149
冗長性
Mobile Link [dbmlsrv9] -v オプション 265
Mobile Link のパフォーマンス 133
証明書
Mobile Link トランスポート・レイヤ・セキュリティのデジタル証明書 209
RSA 証明書の生成 612
RSA 証明書の読み込み 611
楕円曲線暗号化証明書の生成 612
楕円曲線暗号化の証明書の読み込み 611
証明書生成ユーティリティ [gencert]
構文 612
証明書チェーン
Mobile Link トランスポート・レイヤ・セキュリティ 211
証明書チェーンの使用
Mobile Link トランスポート・レイヤ・セキュリティ 211
証明書フィールドの確認
Mobile Link トランスポート・レイヤ・セキュリティ 223
証明書読み込みユーティリティ [readcert]
構文 611
署名
楕円曲線証明書と RSA 証明書 612
署名付き証明書
Mobile Link トランスポート・レイヤ・セキュリティでの作成 214
署名付き証明書の作成
Mobile Link トランスポート・レイヤ・セキュリティ 214

す

スイッチ
Mobile Link サーバ [dbmlsrv9] 236
Mobile Link 証明書ジェネレータ [gencert] 612
Mobile Link 証明書読み込み [readcert] 611
Mobile Link ユーザ認証 [dbmluser] 608
スキーマ
Mobile Link リモート・テーブルと関係する統合テーブル 40
スキーマの変更
Mobile Link リモート・データベース 100
スクリプト
.NET 接続スクリプトの追加と削除 597
.NET テーブル・スクリプトの追加と削除 598
Java 接続スクリプトの追加と削除 599
Java テーブル・スクリプトの追加と削除 600
Mobile Link イベント 401
Mobile Link での統合データベースへの追加

301
Mobile Link 同期の自動化 289
Mobile Link の説明 20, 285
SQL 接続スクリプトの追加と削除 594
SQL テーブル・スクリプトの追加と削除 595
共通パラメータ 297
サポートされている DBMS スクリプトの作成方法 43
接続スクリプト 295
テーブル・スクリプト 295
バージョン 298
ローをアップロードするスクリプトの作成 305
ローをダウンロードするスクリプトの作成 308
スクリプト構文のテスト
Mobile Link 318
スクリプトと同期処理 293
スクリプトの自動化
Mobile Link 同期 289
スクリプトの自動生成
Mobile Link 同期 289
スクリプトの種類
説明 295
スクリプトの直接挿入
Mobile Link 303
スクリプトの追加または削除
Mobile Link 301
スクリプトのパラメータ
説明 297
スクリプト・バージョン
Mobile Link 同期 298
追加 299
スクリプト・バージョンの追加
Mobile Link 299
スクリプトを追加または削除するためのストアド・プロシージャ
Mobile Link サーバ 594
ストアド・プロシージャ
ml_add_connection_script SQL 構文 594
ml_add_dnet_connection_script SQL 構文 597
ml_add_dnet_table_script SQL 構文 598
ml_add_java_connection_script SQL 構文 599
ml_add_java_table_script SQL 構文 600
ml_add_property SQL 構文 602
ml_add_table_script SQL 構文 595
Mobile Link 593
Mobile Link ストアド・プロシージャのソース・コード 303
データのダウンロードに使用 97
同期スクリプトの追加と削除に使用 303
ストアド・プロシージャ・コールからの結果セットのダウンロード
同期の方法 97
ストアド・プロシージャを使用した同期スクリプトの追加と削除 303
スナップショットの例
Mobile Link ファイルベースのダウンロード 117
スナップショットを使った同期
説明 62

せ

生成
RSA 証明書 612
楕円曲線暗号化の証明書 612
制約エラー xi
セキュリティ
FIPS 207
Mobile Link 205
Mobile Link の概要 36
Mobile Link のカスタム・ユーザ認証 420
世代番号
Mobile Link ファイルベースのダウンロード 115
セッション全体の変数
Mobile Link 同期での IBM DB2 52
Oracle パッケージ 52
セッション・プロパティ
Mobile Link モニタ 153
接続
Mobile Link 同期サーバの -x オプション 269
接続スクリプト

.NET スクリプトの削除 597
.NET スクリプトの追加 597
Java スクリプトの削除 599
Java スクリプトの追加 599
Mobile Link スクリプトのアルファベット順のリスト 401
SQL スクリプトの削除 594
SQL スクリプトの追加 594
Sybase Central を使用した追加 301
説明 295
定義 295
[接続スクリプト追加] ウィザード
使用 301
接続の確立
Mobile Link 同期サーバの -x オプション 269
接続パラメータ
Mobile Link サーバの -x オプション 269
接続プロパティ
Mobile Link サーバの -x オプション 269
切断分割
定義 65
同期 65
設定
Apache Web サーバ 186
Apache Web サーバ用のサーブレット・リダイレクタ 180
M-Business Anywhere 191
Microsoft Web サーバ 177
Mobile Link .NET 同期論理 355
Mobile Link Java 同期論理 323
Mobile Link 統合データベース 42
NSAPI Web サーバ 172
Tomcat 180
トランスポート・レイヤ・セキュリティを使用する Mobile Link Adaptive Server Anywhere クライアント 225
リダイレクタ (すべてのバージョン共通) 169
設定スクリプト
Mobile Link 統合データベース 42
前回のタイムスタンプ
スクリプト・パラメータ 297

そ

挿入
Mobile Link のスクリプト 303
ソート順
文字と Mobile Link 同期 664
組織単位
Mobile Link トランスポート・レイヤ・セキュリティでの確認 223

た

タイムスタンプベースの同期
download_cursor スクリプト 61
download_delete_cursor スクリプト 61
説明 60
タイムスタンプベースの例
Mobile Link ファイルベースのダウンロード 123
ダウンロード
Mobile Link でのファイルベース 105
ダウンロード・イベント
Mobile Link 同期 414
ダウンロード確認
Mobile Link のパフォーマンス 131
ダウンロード・キャッシュ・サイズ
Mobile Link のパフォーマンス 132
ダウンロード・ストリーム
Mobile Link トランザクション 23
Mobile Link のパフォーマンス 134
イベント 308
失敗したダウンロード 92
定義 18
ダウンロード専用の同期
説明 30
ダウンロード中のイベント
説明 414
ローをダウンロードするスクリプトの作成 308
ダウンロードの失敗
Mobile Link 再起動可能なダウンロード 92
ダウンロード・バッファ

Mobile Link のパフォーマンス 131
ダウンロード・ファイル
Mobile Link ファイルベースのダウンロード用に作成 108
ダウンロード・ファイルの作成
Mobile Link ファイルベースのダウンロード 108
楕円曲線暗号化の証明書
生成 612
読み込み 611
多対多関係
同期 67
分割 67
断片化 xi

ち

[チャート] ウィンドウ枠
Mobile Link モニタ 150
重複のある分割
Mobile Link 67

つ

追加
Mobile Link .NET 接続スクリプト 597
Mobile Link .NET テーブル・スクリプト 598
テーブル
Mobile Link ASA リモート・データベースに対する追加 100
追加
Mobile Link ASA リモート・データベースに対するテーブルの追加 100
Mobile Link Java 接続スクリプト 599
Mobile Link Java テーブル・スクリプト 600
Mobile Link SQL 接続スクリプト 594
Mobile Link SQL テーブル・スクリプト 595
Mobile Link のプロパティ 602
Mobile Link のユーザ名 608
Mobile Link リモート・データベースに対するカラム 101
Sybase Central を使用した同期スクリプトの追加 301
楕円曲線証明書と RSA 証明書 612
通称
Mobile Link トランスポート・レイヤ・セキュリティでの確認 224
通信
Mobile Link 同期サーバの -x オプション 269
通信フォールト
Mobile Link 同期リカバリ 24

て

停止
Mobile Link 停止ユーティリティ [dbmlstop] 606
Mobile Link 同期サーバ 14
停止ユーティリティ [dbmlstop]
構文 606
データ移動テクノロジ
Mobile Link 同期 9
データ・エントリ
同期の方法 90
データ型
Mobile Link .NET と SQL 361
Mobile Link Java と SQL 327
Mobile Link 統合データベースの変換 653
Mobile Link における ASE データ型への変換 654
Mobile Link における IBM DB2 データ型への変換 656
Mobile Link における Microsoft SQL Server データ型への変換 660
Mobile Link における Oracle データ型への変換 658
Mobile Link における変換 653
データ型の変換
Mobile Link 653
Mobile Link 統合データベース 653
データのダウンロード
Mobile Link でのファイルベースのダウンロード 105

データベース
Mobile Link 統合データベース 39
Mobile Link との同期 9
データベース・スキーマ
Mobile Link リモート・テーブルと関係する統合テーブル 40
データベース接続
Mobile Link のパフォーマンス 132, 139
テーブル
Mobile Link リモート・テーブルと関係する統合テーブル 40
スクリプト・パラメータ 297
分割 65
テーブル・スクリプト
.NET スクリプトの削除 598
.NET スクリプトの追加 598
Java スクリプトの削除 600
Java スクリプトの追加 600
Mobile Link スクリプトのアルファベット順のリスト 401
SQL スクリプトの削除 595
SQL スクリプトの追加 595
Sybase Central を使用した追加 302
説明 295
定義 287, 295
[テーブル・スクリプトを追加] ウィザード
使用 302
テーブル分割
例 65
デーモン
デーモンとしての Mobile Link の実行 195
デーモンとしての UNIX 版 Mobile Link サーバの稼働 196
テクニカル・サポート
ニュースグループ xx
デジタル証明書
Mobile Link トランスポート・レイヤ・セキュリティ 209
デジタル証明書の使用
Mobile Link トランスポート・レイヤ・セキュリティ 209
デジタル署名
Mobile Link トランスポート・レイヤ・セキュリティ 222
テスト
同期スクリプト 318
デッドロック
Mobile Link アップロード・ストリームの処理 26
デバッグ
.NET 同期論理 368
Java クラスを使用した Mobile Link 同期 330
Mobile Link 接続 204
Mobile Link 同期サーバのログ 15
デフォルト値の選択方法
Mobile Link グローバル・オートインクリメント 74
デフォルトのグローバル・オートインクリメント
宣言 73
デフォルトのグローバル・オートインクリメントの宣言
Mobile Link 73

と

同期
.NET での Mobile Link スクリプトの作成 353
bmlsrv9 のプロトコル・オプション 269
Java での Mobile Link スクリプトの作成 321
Mobile Link システム・テーブル 620
Mobile Link システムのアーキテクチャ 10
Mobile Link 処理の概要 18
Mobile Link ストアド・プロシージャ 593
Mobile Link 同期サーバの実行 195
Mobile Link トランザクション 23
Mobile Link における ASE データ型への変換 654
Mobile Link における IBM DB2 データ型への変換 656
Mobile Link における Microsoft SQL Server データ型への変換 660
Mobile Link における Oracle データ型への変換 658

Mobile Link におけるデータ型の変換 653
Mobile Link の説明 9
Mobile Link のタイムスタンプ 24
Mobile Link パフォーマンス 26
Mobile Link 文字セット 664
Mobile Link ユーティリティ 605
Web サーバの設定 161
Windows 以外のプラットフォームでの Mobile Link 文字セットの変換 667
Windows 環境での Mobile Link 文字セットの変換 665
競合解決 80
再起動可能なダウンロード 92
スクリプトのアルファベット順のリスト 401
スクリプトの作成 285
スナップショット 62
多対多関係 67
同期論理の作成オプション 31
統合データベース 39
トランスポート・レイヤ・セキュリティ 205
パフォーマンスに関するヒント 129
プロセス 293
方法 57
モニタの接続パラメータ 145
ローの削除 311
ローのダウンロード 308
同期イベント
authenticate_parameters 417
authenticate_user 420
authenticate_user_hashed 425
begin_connection 429
begin_connection_autocommit 431
begin_download 432, 434
begin_download_deletes 437
begin_download_rows 440
begin_publication 443
begin_synchronization 447, 449
begin_upload 452, 455
begin_upload_deletes 458
begin_upload_rows 461
download_cursor 464
download_delete_cursor 468
download_statistics 472, 476
end_connection 480
end_download 482, 485
end_download_deletes 488
end_download_rows 491
end_publication 494
end_synchronization 497, 499
end_upload 502, 504
end_upload_deletes 508
end_upload_rows 511
example_upload_cursor 514
example_upload_delete 515
example_upload_insert 516
example_upload_update 517
handle_error 518
handle_odbc_error 522
Mobile Link アップロード 410
Mobile Link ダウンロード 414
Mobile Link 同期の説明 402
modify_error_message 525
modify_last_download_timestamp 527
modify_next_last_download_timestamp 530
modify_user 533
new_row_cursor 536
old_row_cursor 539
prepare_for_download 542
report_error 544
report_odbc_error 547
resolve_conflict 550
synchronization_statistics 553, 556
time_statistics 559, 562
upload_cursor 565
upload_delete 568
upload_fetch 570
upload_insert 572
upload_new_row_insert 575
upload_old_row_insert 577
upload_statistics 579, 583
upload_update 587
説明 401
同期エラー
Mobile Link の処理 316
トラブルシューティング 248
同期クライアント
Mobile Link 用の ASA または Ultra Light 17
同期サーバ

Mobile Link の説明　13
同期サブスクリプション　xi
同期システム
コンポーネント　10
同期システムの要素　10
同期処理
説明　18
同期処理のトランザクション　23
同期スクリプト
.NET　353
.NET メソッド　363
.NET 用に実装　355
DBMS 依存性　43
download_cursor　309
handle_error イベント　316
Java　321
Java メソッド　329
Java 用の実装　323
Mobile Link イベント　401
Mobile Link のバージョン　298
report_error　316
Sybase Central を使用した追加　301
Ultra Light 用のサンプル・スクリプト　292
共通パラメータ　297
サポートされている DBMS スクリプトの作成方法　43
サンプル生成　290
実行　293
自動生成　289
種類　295
ストアド・プロシージャを使用した追加と削除　303
接続スクリプト　295
説明　285
追加と削除　301
テーブル・スクリプト　295
テスト　318
例　288
ローをアップロードするスクリプトの作成　305
ローをダウンロードするスクリプトの作成　308
同期スクリプトの概要　285
同期スクリプトの作成
SQL　285
サポートされている DBMS スクリプトの作成方法　43
同期スクリプトの自動生成　289
同期スクリプトの追加
ストアド・プロシージャの使用　303
同期中のエラーの記録　316
[同期テーブル追加] ウィザード
使用　302
同期のアップロード・ストリーム
Mobile Link 処理　26
同期の基本　9
Mobile Link の説明　9
同期の障害処理の方法　24
同期の方法
失敗したダウンロード　92
スナップショットベースの同期　62
説明　57
タイムスタンプベースの同期　60
ダウンロードするためのストアド・プロシージャ　97
データ・エントリ　90
プライマリ・キー・プール　76
分割　65
ローのアップロード　305
ローの削除　91
同期パラメータ
HTTPS 同期　269
HTTP 同期　269
TCP/IP 同期　269
同期プロパティ
Mobile Link モニタ　153
同期ユーザ
Mobile Link ユーザ認証 [dbmluser]　608
同期論理
Mobile Link　285
作成オプション　31
同期論理の作成オプション　31
統計
Mobile Link　158
統計のカスタマイズ

Mobile Link もにた　156
統計のプロパティ
Mobile Link　158
統合データベース
ASA 以外のデータベース　43
DBMS 依存性　43
Mobile Link　12
Mobile Link アプリケーションの互換性の問題　52
Mobile Link システム・テーブル　620
Mobile Link 統合データベースとして使用する IBM DB2　51
Mobile Link 統合データベースとして使用する Oracle　49
Mobile Link 統合データベースとして使用する SQL Server　54
Mobile Link 統合データベースとしての ASA　46
Mobile Link 統合データベースとしての ASE　47
Mobile Link 統合データベースの作成　42
Mobile Link におけるデータ型の変換　653
Mobile Link リモート・テーブルと関係するテーブル　40
同期スクリプトの追加　301
統合データベースでのスクリプトの追加と削除
Mobile Link　301
統合データベースでの変更
Mobile Link ファイルベースのダウンロード　108
統合データベースの作成
説明　42
統合データベースの設定
説明　42
同時実行性
Mobile Link アップロード・ストリームの処理　26
Mobile Link のパフォーマンス　130
ドメイン設定ファイル
説明　375
ドライバ
Mobile Link でサポートされるドライバ　670
トラブルシューティング
Mobile Link サーバ起動時　204
Mobile Link 再起動可能なダウンロード　92
Mobile Link 同期サーバのログ　15
失敗したダウンロードの処理　92
同期エラー　248
トランザクション
Mobile Link .NET 同期論理　361
Mobile Link Java 同期論理　327
Mobile Link 同期中　23, 24
トランザクション・レベルのアップロード
Mobile Link [dbmlsrv9] -us オプション　265
トランスポート・レイヤ・セキュリティ
Mobile Link　205
Mobile Link でサポートされるプラットフォーム　206
Mobile Link での設定　208
Mobile Link での動作　206
Mobile Link 内での効率性　206
トランスポート・レイヤ・セキュリティの設定
Mobile Link　208
トランスポート・レイヤ・セキュリティを使用する Mobile Link クライアントの設定
説明　222
トランスポート・レイヤ・セキュリティを使用する Mobile Link 同期サーバの起動
説明　219
トランスポート・レイヤ・セキュリティを使用する Ultra Light クライアントの設定
説明　228

に

ニュースグループ
テクニカル・サポート　xx
認証
Mobile Link dbmluser ユーティリティ　608
認証局

Mobile Link トランスポート・レイヤ・セキュリティ 218

ね

ネットワーク・パラメータ
Mobile Link サーバの -x オプション 269
ネットワーク・プロトコル
HTTPS_FIPS を使用した dbmlsrv9 273
HTTPS を使用した dbmlsrv9 273
HTTP を使用した dbmlsrv9 272
Mobile Link 同期サーバ 269
TCP/IP を使用した dbmlsrv9 270

は

バージョン
Mobile Link 同期スクリプト 298
スクリプト・バージョンの追加 299
[バージョン追加] ウィザード
使用 299
配備
ASA Mobile Link クライアント 682
Mobile Link 673
Mobile Link アプリケーション 673
Mobile Link 同期サーバ 675
Mobile Link のパフォーマンス 129
QAnywhere アプリケーション 686
Ultra Light アプリケーション 685
アプリケーションとデータベース 673
配備の概要
Mobile Link 674
配布可能なダウンロード
Mobile Link でのファイルベースのダウンロード 105
パスワード
Mobile Link dbmluser ユーティリティ 608
パッケージ
セッション全体の情報 52
パッケージ化されたダウンロード
Mobile Link でのファイルベースのダウンロード 105
パフォーマンス
Mobile Link 129
Mobile Link アップロード・ストリームの処理 26
ダウンロード 134
パフォーマンスに関するヒント
Mobile Link 130
パブリック・キー暗号方式
Mobile Link 205
パブリック証明書を信用するようにクライアントを設定する
Mobile Link トランスポート・レイヤ・セキュリティ 218
パラメータ
ml_username 297
Mobile Link テーブル名 297
Mobile Link ユーザ名 297
最終ダウンロード・タイムスタンプ 297
同期スクリプト 297

ひ

表記
規則 xv
ヒント
Mobile Link のパフォーマンス 129
同期の方法 59

ふ

ファイアウォール
Mobile Link クライアントの設定 166
Mobile Link 同期サーバの設定 166
Mobile Link 要求のルート指定 162
ファイル
Mobile Link でのファイルベースのダウンロード 105
ファイル定義データベース
作成 107
定義済み 107

ファイル定義データベースの作成
Mobile Link 107
ファイルベースのダウンロード
説明 105
例 117
ファイルベースのダウンロードの設定
説明 107
フィードバック
提供 xx
マニュアル xx
フェールオーバ
Mobile Link リダイレクタ 162
フォールト
Mobile Link 同期リカバリ 24
負荷分散
Mobile Link リダイレクタ 162
リダイレクタの例 171
複数のアプリケーション
Mobile Link スクリプトの区別 298
フック xi
プライベート・アセンブリ
Mobile Link での実装 373
プライマリ・キー
Mobile Link と ASE 48
Mobile Link プライマリ・キー・プール 76
Mobile Link 用のユニークな値の生成 70
Oracle のシーケンス 49
同期における一意性 70
プライマリ・キー・プール
Mobile Link の例 77
Mobile Link 用にデフォルトのグローバル・オートインクリメントを使用したユニークな値の生成 71
同期 76
ブロードキャスト・ダウンロード
Mobile Link でのファイルベースのダウンロード 105
プロキシ Web サーバ
Mobile Link 162
プロシージャ
Mobile Link 593
プロトコル
HTTPS_FIPS を使用した dbmlsrv9 273
HTTPS を使用した dbmlsrv9 273
HTTP を使用した dbmlsrv9 272
Mobile Link 同期 10
Mobile Link 同期サーバ 269
TCP/IP を使用した dbmlsrv9 270
プロパティ
QAnywhere サーバ 251
プロパティを追加または削除するためのストアド・プロシージャ
Mobile Link サーバ 602
分割
切断 65
定義 65
分散データベース
Mobile Link 同期 9
文ベースのアップロード
競合検出 81
文ベースのスクリプト
ローのアップロード 305

へ

変換
Mobile Link データ型 653
Mobile Link における ASE データ型への変換 654
Mobile Link における IBM DB2 データ型への変換 656
Mobile Link における Microsoft SQL Server データ型への変換 660
Mobile Link における Oracle データ型への変換 658
ODBC ドライバによる文字セットの変換 666

ほ

ボトルネック
Mobile Link のパフォーマンス 135

ま

マニュアル
SQL Anywhere Studio xii

み

民間の認証局
Mobile Link トランスポート・レイヤ・セキュリティ 209

め

メソッド
Mobile Link .NET 同期論理 363
Mobile Link Java 同期論理 329
メッセージ
Mobile Link QAnywhere システム・テーブル 619
メッセージ・プロパティ・ファイル
QAnywhere の -m オプションで dbmlsrv9 を起動 251

も

文字セット
Mobile Link 同期 664
文字セット間の変換
Windows 以外のプラットフォームでの Mobile Link 同期 667
Windows 環境での Mobile Link 同期 665
文字セットの考慮事項
Mobile Link 663
文字セットの変換
ODBC ドライバによる変換 666
Windows 以外のプラットフォームでの Mobile Link 同期中 667
Windows 環境での Mobile Link 同期中 665
戻り値
.NET 同期 363
Java 同期 329
モニタ
Mobile Link のパフォーマンス 141
Mobile Link モニタ 143
モニタのデータの保存
Mobile Link モニタ 154
モニタリング
Mobile Link での同期 143

ゆ

ユーザ固有の競合解決 89
ユーザ定義起動クラス
Mobile Link .NET 同期論理 364
Mobile Link Java 同期論理 333
ユーザ認証ユーティリティ [dbmluser]
構文 608
ユーザ名
Mobile Link ユーザ認証ユーティリティ [dbmluser] 608
ユーザ名の格納
競合解決中 89
優先同期
Mobile Link のパフォーマンス 134
ユーティリティ
Mobile Link サーバ [dbmlsrv9] 236
Mobile Link 証明書ジェネレータ [gencert] 612
Mobile Link 証明書読み込み [readcert] 611
Mobile Link 停止ユーティリティ [dbmlstop] 606
Mobile Link ユーザ認証 [dbmluser] 608
Mobile Link リダイレクタ 161
ユニークなプライマリ・キー
Mobile Link 70
Mobile Link 用にキー・プールを使用した生成 76
Mobile Link 用にグローバル・オートインクリメントを使用した生成 71
UUID を使用して Mobile Link 用に生成 70
ユニークなプライマリ・キーの管理
Mobile Link 70

よ

要求
Mobile Link でのルート指定 161
要求のルート指定
Mobile Link 同期 161
読み込み
RSA 証明書 611
楕円曲線暗号化の証明書 611

り

リダイレクタ
Apache Web サーバ用のサーブレット・リダイレクタ 180
Apache ネイティブ 186
ISAPI 177
M-Business Anywhere 191
Microsoft Web サーバ 177
Mobile Link クライアントとサーバの設定 166
NSAPI バージョン 172
Tomcat 用のサーブレット・リダイレクタの設定 180
使用するタイミング 163
使用方法 162
設定 (すべてのバージョン共通) 169
説明 161
負荷分散の例 171
ロケーションの指定 169
リダイレクタの設定
概要 165
リダイレクタのプロパティの設定 (すべてのバージョン共通) 169
リバース・プロキシ
定義 162
リモート・データベース
Mobile Link リモート・テーブルと関係する統合テーブル 40
リモート・データベース間でローを分割する 65
リモート・データベースでのスキーマの変更
Mobile Link 100
リモート・データベースの配備
説明 673
リモート・テーブル
Mobile Link でのローの削除 311
リモート・テーブルと統合データベースの関係 40

る

ルート証明書
Mobile Link トランスポート・レイヤ・セキュリティ 209
Mobile Link トランスポート・レイヤ・セキュリティのクライアント検証 218

れ

例
Java 同期論理 337
Mobile Link ファイルベースのダウンロード 117
Ultra Light 同期スクリプトの生成 292

ろ

ロー
Mobile Link リモート・データベースでの削除 311
分割 65
ローのアップロード
.NET の同期の方法 371
Mobile Link のパフォーマンス 134
スクリプトの作成 305
ローの削除
Mobile Link リモート・データベース 311
同期の方法 91
ローのダウンロード
同期スクリプト 308

ローをアップロードするスクリプトの作成
Mobile Link 305
ローをダウンロードするスクリプトの作成
Mobile Link 308
ロギング
Mobile Link 同期サーバの動作 15
Mobile Link のパフォーマンス 133
ログ・ファイル
Mobile Link 同期サーバ 15

わ

ワーカ・スレッド
Mobile Link 136
Mobile Link のパフォーマンス 130