SYBASE iAnywhere

Mobile Link
クライアント管理

2009 年 2 月

バージョン 11.0.1

版権と商標

目次

はじめに

このマニュアルの内容

このマニュアルでは、Mobile Link クライアントを設定、構成、同期する方法について説明します。Mobile Link クライアントには、SQL Anywhere または Ultra Light のいずれかのデータベースを使用できます。また、dbmlsync API についても説明します。dbmlsync API を使用すると、同期を C++ または .NET のクライアント・アプリケーションにシームレスに統合できます。

対象読者

このマニュアルは、情報システムに同期を追加したいと考えている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、「データ交換テクノロジの概要」『SQL Anywhere 11 - 紹介』を参照してください。

SQL Anywhere のマニュアルについて

SQL Anywhere の完全なマニュアルは 4 つの形式で提供されており、いずれも同じ情報が含まれています。

- **HTML ヘルプ**　オンライン・ヘルプには、SQL Anywhere 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。

  Microsoft Windows オペレーティング・システムを使用している場合は、オンライン・ヘルプは HTML ヘルプ (CHM) 形式で提供されます。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 11] - [マニュアル] - [オンライン・マニュアル]** を選択します。

  管理ツールのヘルプ機能でも、同じオンライン・マニュアルが使用されます。

- **Eclipse**　UNIX プラットフォームでは、完全なオンライン・ヘルプは Eclipse 形式で提供されます。マニュアルにアクセスするには、SQL Anywhere 11 インストール環境の *bin32* または *bin64* ディレクトリから *sadoc* を実行します。

- **DocCommentXchange**　DocCommentXchange は、SQL Anywhere マニュアルにアクセスし、マニュアルについて議論するためのコミュニティです。

  DocCommentXchange は次の目的に使用できます (現在のところ、日本語はサポートされておりません)。

  ○ マニュアルを表示する

  ○ マニュアルの項目について明確化するために、ユーザによって追加された内容を確認する

  ○ すべてのユーザのために、今後のリリースでマニュアルを改善するための提案や修正を行う

  http://dcx.sybase.com を参照してください。

- **PDF**　SQL Anywhere の完全なマニュアル・セットは、Portable Document Format (PDF) 形式のファイルとして提供されます。内容を表示するには、PDF リーダが必要です。Adobe Reader をダウンロードするには、http://get.adobe.com/reader/ にアクセスしてください。

  Microsoft Windows オペレーティング・システムで PDF 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 11] - [マニュアル] - [オンライン・マニュアル - PDF]** を選択します。

  UNIX オペレーティング・システムで PDF マニュアルにアクセスするには、Web ブラウザを使用して *install-dir/documentation/ja/pdf/index.html* を開きます。

マニュアル・セットに含まれる各マニュアルについて

SQL Anywhere のマニュアルは次の構成になっています。

- **『SQL Anywhere 11 - 紹介』**　このマニュアルでは、データの管理および交換機能を提供する包括的なパッケージである SQL Anywhere 11 について説明します。SQL Anywhere を使用す

ると、サーバ環境、デスクトップ環境、モバイル環境、リモート・オフィス環境に適したデータベース・ベースのアプリケーションを迅速に開発できるようになります。

- **『SQL Anywhere 11 - 変更点とアップグレード』** このマニュアルでは、SQL Anywhere 11 とそれ以前のバージョンに含まれる新機能について説明します。
- **『SQL Anywhere サーバ - データベース管理』** このマニュアルでは、SQL Anywhere データベースを実行、管理、構成する方法について説明します。データベース接続、データベース・サーバ、データベース・ファイル、バックアップ・プロシージャ、セキュリティ、高可用性、Replication Server を使用したレプリケーション、管理ユーティリティとオプションについて説明します。
- **『SQL Anywhere サーバ - プログラミング』** このマニュアルでは、C、C++、Java、PHP、Perl、Python、および Visual Basic や Visual C# などの .NET 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。ADO.NET や ODBC などのさまざまなプログラミング・インタフェースについても説明します。
- **『SQL Anywhere サーバ - SQL リファレンス』** このマニュアルでは、システム・プロシージャとカタログ (システム・テーブルとビュー) に関する情報について説明します。また、SQL Anywhere での SQL 言語の実装 (探索条件、構文、データ型、関数) についても説明します。
- **『SQL Anywhere サーバ - SQL の使用法』** 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。
- **『Mobile Link - クイック・スタート』** 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。
- **『Mobile Link - クライアント管理』** このマニュアルでは、Mobile Link クライアントを設定、構成、同期する方法について説明します。Mobile Link クライアントには、SQL Anywhere または Ultra Light のいずれかのデータベースを使用できます。また、dbmlsync API についても説明します。dbmlsync API を使用すると、同期を C++ または .NET のクライアント・アプリケーションにシームレスに統合できます。
- **『Mobile Link - サーバ管理』** このマニュアルでは、Mobile Link アプリケーションを設定して管理する方法について説明します。
- **『Mobile Link - サーバ起動同期』** このマニュアルでは、Mobile Link サーバ起動同期について説明します。この機能により、Mobile Link サーバは同期を開始したり、リモート・デバイス上でアクションを実行することができます。
- **『QAnywhere』** このマニュアルでは、モバイル・クライアント、ワイヤレス・クライアント、デスクトップ・クライアント、およびラップトップ・クライアント用のメッセージング・プラットフォームである、QAnywhere について説明します。
- **『SQL Remote』** 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・システムについて説明します。このシステムによって、SQL Anywhere の統合データベースと複数の SQL Anywhere リモート・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。

- **『Ultra Light - データベース管理とリファレンス』** このマニュアルでは、小型デバイス用 Ultra Light データベース・システムの概要を説明します。
- **『Ultra Light - C/C++ プログラミング』** このマニュアルでは、Ultra Light C および Ultra Light C++ のプログラミング・インタフェースについて説明します。Ultra Light を使用すると、ハンドヘルド・デバイス、モバイル・デバイス、埋め込みデバイスのデータベース・アプリケーションを開発し、これらのデバイスに配備できます。
- **『Ultra Light - M-Business Anywhere プログラミング』** このマニュアルは、Ultra Light for M-Business Anywhere について説明します。Ultra Light for M-Business Anywhere を使用すると、Palm OS、Windows Mobile、または Windows を搭載しているハンドヘルド・デバイス、モバイル・デバイス、または埋め込みデバイスの Web ベースのデータベース・アプリケーションを開発し、これらのデバイスに配備できます。
- **『Ultra Light - .NET プログラミング』** このマニュアルでは、Ultra Light.NET について説明します。Ultra Light.NET を使用すると、PC、ハンドヘルド・デバイス、モバイル・デバイス、または埋め込みデバイスのデータベース・アプリケーションを開発し、これらのデバイスに配備できます。
- **『Ultra Light J』** このマニュアルでは、Ultra Light J について説明します。Ultra Light J を使用すると、Java をサポートしている環境用のデータベース・アプリケーションを開発し、配備することができます。Ultra Light J は、BlackBerry スマートフォンと Java SE 環境をサポートしており、iAnywhere Ultra Light データベース製品がベースになっています。
- **『エラー・メッセージ』** このマニュアルでは、SQL Anywhere エラー・メッセージの完全なリストを示し、その診断情報を説明します。

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表記規則について説明します。

オペレーティング・システム

SQL Anywhere はさまざまなプラットフォームで稼働します。ほとんどの場合、すべてのプラットフォームで同じように動作しますが、いくつかの相違点や制限事項があります。このような相違点や制限事項は、一般に、基盤となっているオペレーティング・システム (Windows、UNIX など) に由来しており、使用しているプラットフォームの種類 (AIX、Windows Mobile など) またはバージョンに依存していることはほとんどありません。

オペレーティング・システムへの言及を簡素化するために、このマニュアルではサポートされているオペレーティング・システムを次のようにグループ分けして表記します。

- **Windows** Microsoft Windows ファミリを指しています。これには、主にサーバ、デスクトップ・コンピュータ、ラップトップ・コンピュータで使用される Windows Vista や Windows XP、およびモバイル・デバイスで使用される Windows Mobile が含まれます。

  特に記述がないかぎり、マニュアル中に Windows という記述がある場合は、Windows Mobile を含むすべての Windows ベース・プラットフォームを指しています。

- **UNIX**　特に記述がないかぎり、マニュアル中に UNIX という記述がある場合は、Linux および Mac OS X を含むすべての UNIX ベース・プラットフォームを指しています。

ディレクトリとファイル名

ほとんどの場合、ディレクトリ名およびファイル名の参照形式はサポートされているすべてのプラットフォームで似通っており、それぞれの違いはごくわずかです。このような場合は、Windows の表記規則が使用されています。詳細がより複雑な場合は、マニュアルにすべての関連形式が記載されています。

ディレクトリ名とファイル名の表記を簡素化するために使用されている表記規則は次のとおりです。

- **大文字と小文字のディレクトリ名**　Windows と UNIX では、ディレクトリ名およびファイル名には大文字と小文字が含まれている場合があります。ディレクトリやファイルが作成されると、ファイル・システムでは大文字と小文字の区別が維持されます。

  Windows では、ディレクトリおよびファイルを参照するとき、大文字と小文字は**区別されません**。大文字と小文字を混ぜたディレクトリ名およびファイル名は一般的に使用されますが、参照するときはすべて小文字を使用するのが通常です。SQL Anywhere では、*Bin32* や *Documentation* などのディレクトリがインストールされます。

  UNIX では、ディレクトリおよびファイルを参照するとき、大文字と小文字は**区別されます**。大文字と小文字を混ぜたディレクトリ名およびファイル名は一般的に使用されません。ほとんどの場合は、すべて小文字の名前が使用されます。SQL Anywhere では、*bin32* や *documentation* などのディレクトリがインストールされます。

  このマニュアルでは、ディレクトリ名に Windows の形式を使用しています。ほとんどの場合、大文字と小文字が混ざったディレクトリ名をすべて小文字に変換すると、対応する UNIX 用のディレクトリ名になります。

- **各ディレクトリおよびファイル名を区切るスラッシュ**　マニュアルでは、ディレクトリの区切り文字に円記号を使用しています。たとえば、PDF 形式のマニュアルは *install-dir¥Documentation¥ja¥pdf* にあります。これは Windows の形式です。

  UNIX では、円記号をスラッシュに置き換えます。PDF マニュアルは *install-dir/documentation/ja/pdf* にあります。

- **実行ファイル**　マニュアルでは、実行ファイルの名前は、Windows の表記規則が使用され、*.exe* や *.bat* などの拡張子が付きます。UNIX では、実行ファイルの名前に拡張子は付きません。

  たとえば、Windows でのネットワーク・データベース・サーバは *dbsrv11.exe* です。UNIX では *dbsrv11* です。

- ***install-dir***　インストール・プロセス中に、SQL Anywhere をインストールするロケーションを選択します。このロケーションを参照する環境変数 SQLANY11 が作成されます。このマニュアルでは、そのロケーションを *install-dir* と表します。

  たとえば、マニュアルではファイルを install-dir¥readme.txt のように参照します。これは、Windows では、%SQLANY11%¥readme.txt に対応します。UNIX では、$SQLANY11/readme.txt または ${SQLANY11}/readme.txt に対応します。

install-dir のデフォルト・ロケーションの詳細については、「SQLANY11 環境変数」『SQL Anywhere サーバ - データベース管理』を参照してください。

- ***samples-dir*** インストール・プロセス中に、SQL Anywhere に含まれるサンプルをインストールするロケーションを選択します。このロケーションを参照する環境変数 SQLANYSAMP11 が作成されます。このマニュアルではそのロケーションを *samples-dir* と表します。

  Windows エクスプローラ・ウィンドウで *samples-dir* を開くには、**[スタート] - [プログラム] - [SQL Anywhere 11] - [サンプル・アプリケーションとプロジェクト]** を選択します。

  samples-dir のデフォルト・ロケーションの詳細については、「SQLANYSAMP11 環境変数」『SQL Anywhere サーバ - データベース管理』を参照してください。

コマンド・プロンプトとコマンド・シェル構文

ほとんどのオペレーティング・システムには、コマンド・シェルまたはコマンド・プロンプトを使用してコマンドおよびパラメータを入力する方法が、1 つ以上あります。Windows のコマンド・プロンプトには、コマンド・プロンプト (DOS プロンプト) および 4NT があります。UNIX のコマンド・シェルには、Korn シェルおよび bash があります。各シェルには、単純コマンドからの拡張機能が含まれています。拡張機能は、特殊文字を指定することで起動されます。特殊文字および機能は、シェルによって異なります。これらの特殊文字を誤って使用すると、多くの場合、構文エラーや予期しない動作が発生します。

このマニュアルでは、一般的な形式のコマンド・ラインの例を示します。これらの例に、シェルにとって特別な意味を持つ文字が含まれている場合、その特定のシェル用にコマンドを変更することが必要な場合があります。このマニュアルではコマンドの変更について説明しませんが、通常、その文字を含むパラメータを引用符で囲むか、特殊文字の前にエスケープ文字を記述します。

次に、プラットフォームによって異なるコマンド・ライン構文の例を示します。

- **カッコと中カッコ** 一部のコマンド・ライン・オプションは、詳細な値を含むリストを指定できるパラメータを要求します。リストは通常、カッコまたは中カッコで囲まれています。このマニュアルでは、カッコを使用します。次に例を示します。

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

  カッコによって構文エラーになる場合は、代わりに中カッコを使用します。

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

  どちらの形式でも構文エラーになる場合は、シェルの要求に従ってパラメータ全体を引用符で囲む必要があります。

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

- **引用符** パラメータの値として引用符を指定する必要がある場合、その引用符はパラメータを囲むために使用される通常の引用符と競合する可能性があります。たとえば、値に二重引用符を含む暗号化キーを指定するには、キーを引用符で囲み、パラメータ内の引用符をエスケープします。

  ```
  -ek "my ¥"secret¥" key"
  ```


多くのシェルでは、キーの値は my "secret" key のようになります。

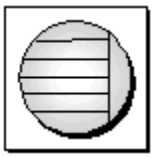
- **環境変数**　マニュアルでは、環境変数設定が引用されます。Windows のシェルでは、環境変数は構文 *%ENVVAR%* を使用して指定されます。UNIX のシェルでは、環境変数は構文 *$ENVVAR* または *${ENVVAR}* を使用して指定されます。

グラフィック・アイコン

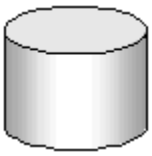
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション。

- SQL Anywhere などのデータベース・サーバ。



- データベース。ハイレベルの図では、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 サーバ、SQL Remote Message Agent などが挙げられます。

- プログラミング・インタフェース。

ドキュメンテーション・チームへのお問い合わせ

このヘルプに関するご意見、ご提案、フィードバックをお寄せください。

SQL Anywhere ドキュメンテーション・チームへのご意見やご提案は、弊社までご連絡ください。頂戴したご意見はマニュアルの向上に役立たせていただきます。ぜひとも、ご意見をお寄せください。

DocCommentXchange

DocCommentXchange を使用して、ヘルプ・トピックに関するご意見を直接お寄せいただくこともできます。DocCommentXchange (DCX) は、SQL Anywhere マニュアルにアクセスしたり、マニュアルについて議論するためのコミュニティです。DocCommentXchange は次の目的に使用できます (現在のところ、日本語はサポートされておりません)。

- マニュアルを表示する
- マニュアルの項目について明確化するために、ユーザによって追加された内容を確認する
- すべてのユーザのために、今後のリリースでマニュアルを改善するための提案や修正を行う

http://dcx.sybase.com を参照してください。

詳細情報の検索／テクニカル・サポートの依頼

詳しい情報やリソースについては、iAnywhere デベロッパー・コミュニティ (http://www.iAnywhere.jp/developers/index.html) 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 Sybase iAnywhere 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 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。バージョンおよびビルド番号を調べるには、コマンド **dbeng11 -v** を実行します。

ニュースグループは、ニュース・サーバ *forums.sybase.com* にあります。

以下のニュースグループがあります。

- ianywhere.public.japanese.general

Web 開発に関する問題については、http://groups.google.com/group/sql-anywhere-web-development を参照してください。

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 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。

Mobile Link クライアントの紹介

この項では、Mobile Link 同期に使用できるクライアントについて紹介するとともに、あらゆる種類の Mobile Link クライアントに共通する情報を示します。

Mobile Link クライアント

目次

SQL Anywhere クライアント

SQL Anywhere データベースを Mobile Link クライアントとして使用するには、データベースに同期オブジェクトを追加します。追加する必要があるオブジェクトは、パブリケーション、Mobile Link ユーザ、パブリケーションをユーザに結び付けるサブスクリプションです。次の項を参照してください。

- 「リモート・データベースの作成」 100 ページ
- 「データのパブリッシュ」 105 ページ
- 「Mobile Link ユーザの作成」 113 ページ
- 「同期サブスクリプションの作成」 116 ページ

dbmlsync というコマンド・ライン・ユーティリティを実行すると、同期を開始します。このユーティリティは、リモート・データベースに接続し、通常はリモート・データベースのトランザクション・ログに含まれる情報を使用してアップロードを準備します(または、トランザクション・ログを使用しないで、スクリプト化されたアップロードを起動できます)。dbmlsync ユーティリティは、同期パブリケーションと同期サブスクリプションに保管された情報を使用して Mobile Link サーバに接続し、データを交換します。

「同期の開始」 119 ページを参照してください。

SQL Anywhere クライアントの詳細については、「SQL Anywhere クライアント」 99 ページを参照してください。

dbmlsync コマンド・ライン・オプションの詳細については、「Mobile Link SQL Anywhere クライアント・ユーティリティ (dbmlsync)」 137 ページを参照してください。

同期のカスタマイズ

「dbmlsync の同期のカスタマイズ」 130 ページを参照してください。

Ultra Light クライアント

Ultra Light アプリケーションは、アプリケーションに適切な同期機能の呼び出しが含まれていると自動的に Mobile Link が有効になります。

Ultra Light のアプリケーションとライブラリは、アプリケーション側での同期アクションを処理します。Ultra Light アプリケーションは、同期をほとんど考慮しないで記述できます。Ultra Light ランタイムは、前回の同期以後に加えられた変更を追跡します。

TCP/IP、HTTP、HTTPS、または ActiveSync を使用している場合には、同期関数を 1 回呼び出すと、アプリケーションから同期が開始されます。HotSync 用のインタフェースは、若干異なります。同期がアプリケーションまたは HotSync から開始されると、Mobile Link サーバと Ultra Light ランタイムが同期中の動作を制御します。

参照

- Ultra Light データベース管理とリファレンス
- 「Ultra Light クライアント」『Ultra Light データベース管理とリファレンス』
- 「Ultra Light J クライアント」 6 ページ
- 「Ultra Light での ActiveSync と HotSync の使用」『Ultra Light データベース管理とリファレンス』
- 「Ultra Light 同期パラメータとネットワーク・プロトコル・オプション」『Ultra Light データベース管理とリファレンス』

Ultra Light J クライアント

Ultra Light J は、Java アプリケーションに、堅牢な同期を行うために変更とステータスの追跡機能を含む Mobile Link 同期クライアントを提供します。Ultra Light J アプリケーションは、アプリケーションに適切な同期機能の呼び出しが含まれていると自動的に Mobile Link が有効になります。

Ultra Light J のアプリケーションとライブラリは、アプリケーション側での同期アクションを処理します。Ultra Light J アプリケーションは、同期をほとんど考慮しないで記述できます。Ultra Light J ランタイムは、前回の同期以後に加えられた変更を追跡します。

HTTP または HTTPS を使用している場合には、同期関数を 1 回呼び出すと、アプリケーションから同期が開始されます。

Mobile Link ファイル転送ユーティリティ (mlfiletransfer) と SQL パススルー機能は、Ultra Light J クライアントでは使用できません。

参照

- 「データの同期」 『Ultra Light J』
- 「Mobile Link クライアントとしての Ultra Light J の使用」 『Ultra Light J』
- Ultra Light データベース管理とリファレンス
- 「Ultra Light クライアント」 『Ultra Light データベース管理とリファレンス』
- 「Ultra Light 同期パラメータとネットワーク・プロトコル・オプション」 『Ultra Light データベース管理とリファレンス』

クライアントのネットワーク・プロトコルの指定

Mobile Link サーバでは、-x コマンド・ライン・オプションを使用して、同期クライアントが Mobile Link サーバに接続するための 1 つ以上のネットワーク・プロトコルを指定します。クライアントが使用する同期プロトコルと一致するネットワーク・プロトコルを選択してください。

mlsrv11 コマンド・ライン・オプションの構文は次のとおりです。

mlsrv11 -c "*connection-string***" -x** *protocol***(** *options* **)**

次の例では、TCP/IP プロトコルが選択されますが、プロトコル・オプションが指定されていません。

```
mlsrv11 -c "dsn=SQL Anywhere 11 Demo" -x tcpip
```

プロトコルは、次の形式のオプションを使用して設定できます。

(*keyword=value***;...)**

次に例を示します。

```
mlsrv11 -c "dsn=SQL Anywhere 11 Demo" -x tcpip(
  host=localhost;port=2439)
```

参照

Mobile Link ネットワーク・プロトコル・オプションの詳細については、次の表を参照してください。

項目	参照先
Mobile Link サーバのネットワーク・オプションの設定方法	「-x オプション」 『Mobile Link - サーバ管理』
Mobile Link クライアント・アプリケーションで使用可能なすべてのネットワーク・プロトコル・オプション	「Mobile Link クライアント・ネットワーク・プロトコル・オプションの一覧」 37 ページ
SQL Anywhere クライアントのオプションの設定方法	「CommunicationAddress (adr) 拡張オプション」 197 ページ 「CommunicationType (ctp) 拡張オプション」 199 ページ
Ultra Light クライアントのオプションの設定方法	「Stream Parameters 同期パラメータ」 『Ultra Light データベース管理とリファレンス』 「Stream Type 同期パラメータ」 『Ultra Light データベース管理とリファレンス』 「Ultra Light 同期ユーティリティ (ulsync)」 『Ultra Light データベース管理とリファレンス』

Mobile Link のシステム・テーブル

Mobile Link サーバのシステム・テーブル

統合データベースとして使用するデータベースを設定すると、Mobile Link サーバに必要な Mobile Link システム・テーブルが作成されます。

「Mobile Link サーバのシステム・テーブル」 『Mobile Link - サーバ管理』を参照してください。

Ultra Light のシステム・テーブル

「Ultra Light のシステム・テーブル」 『Ultra Light データベース管理とリファレンス』を参照してください。

SQL Anywhere システム・テーブル

SQL Anywhere システム・テーブルは直接アクセスできませんが、システム・ビューを使用してアクセスできます。「システム・ビュー」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

次の SQL Anywhere システム・ビューは、Mobile Link のユーザにとって特に関心のあるものです。

- 「SYSSYNC システム・ビュー」 『SQL Anywhere サーバ - SQL リファレンス』
- 「SYSPUBLICATION システム・ビュー」 『SQL Anywhere サーバ - SQL リファレンス』
- 「SYSSUBSCRIPTION システム・ビュー」 『SQL Anywhere サーバ - SQL リファレンス』
- 「SYSSYNCSCRIPT システム・ビュー」 『SQL Anywhere サーバ - SQL リファレンス』

SQL Anywhere は、システム・ビューに対してクエリを実行して必要な情報を提供する統合ビューも備えています。「統合ビュー」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

Mobile Link ユーザ

目次

Mobile Link ユーザの概要

「Mobile Link ユーザ」とは、Mobile Link サーバに接続するときに認証に使用される名前で、「同期ユーザ」とも呼ばれます。

ユーザを同期システムに含めるための条件は次のとおりです。

- リモート・データベース上に Mobile Link ユーザ名を作成してください。
- Mobile Link ユーザ名を Mobile Link サーバに登録してください。

Mobile Link ユーザの名前とパスワードは、データベース・ユーザの名前とパスワードとは異なります。Mobile Link ユーザ名は、リモート・データベースから Mobile Link サーバへの接続を認証するために使用されます。

ユーザ名を使用して、Mobile Link サーバの動作を制御することもできます。そのためには、同期スクリプト内で **username** パラメータを使用します。

「スクリプトでのリモート ID と Mobile Link ユーザ名の使用」 16 ページを参照してください。

Mobile Link ユーザ名は、統合データベースの Mobile Link システム・テーブル ml_user の名前カラムに格納されます。

Mobile Link ユーザ名は、同期システム内でユニークである必要はありません。セキュリティ上問題がない場合は、各リモート・データベースに同じ Mobile Link ユーザ名を割り当てることもできます。

Ultra Light ユーザ認証

Ultra Light と Mobile Link のユーザ認証スキームは異なりますが、Ultra Light ユーザ ID の値を Mobile Link ユーザ名と共有にして簡素化できます。このように簡素化できるのは、Ultra Light アプリケーションを単一ユーザが使用している場合のみです。

「Ultra Light ユーザ認証」 『Ultra Light データベース管理とリファレンス』を参照してください。

Mobile Link ユーザの作成と登録

Mobile Link ユーザをリモート・データベースに作成し、統合データベースに登録します。

リモート・データベースの Mobile Link ユーザの作成

リモート・データベース側にユーザを追加する場合、次のオプションがあります。

- SQL Anywhere リモート・データベースの場合は、Sybase Central または CREATE SYNCHRONIZATION USER 文を使用します。

  「Mobile Link ユーザの作成」 113 ページを参照してください。

- Ultra Light リモート・データベースの場合は、User Name と Password 同期パラメータを設定します。

「User Name 同期パラメータ」『Ultra Light データベース管理とリファレンス』と「Password 同期パラメータ」『Ultra Light データベース管理とリファレンス』を参照してください。

統合データベースへの Mobile Link ユーザ名の追加

リモート・データベースでユーザ名が作成された後、次の方法のいずれかを使用して、統合データベースにユーザ名を登録できます。

- mluser ユーティリティを使用する。

  「Mobile Link ユーザ認証ユーティリティ (mluser)」『Mobile Link - サーバ管理』を参照してください。

- Sybase Central を使用する。
- authenticate_user イベント用または authenticate_user_hashed イベント用のスクリプトを実装する。これらのスクリプトのどちらかを起動すると、Mobile Link サーバによって、認証が正常に行われるユーザが自動的に追加されます。

  「authenticate_user 接続イベント」『Mobile Link - サーバ管理』または「authenticate_user_hashed 接続イベント」『Mobile Link - サーバ管理』を参照してください。

- mlsrv11 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。このオプションは、開発時には便利ですが、配備されたアプリケーションへの使用はおすすめできません。

  「-zu オプション」『Mobile Link - サーバ管理』を参照してください。

ユーザの最初のパスワードを設定する

各ユーザのパスワードは、ユーザ名とともに ml_user テーブルに格納されます。最初のパスワードは、Sybase Central または mluser コマンド・ライン・ユーティリティを使用して指定できます。

Sybase Central は、個々のユーザとパスワードを追加する場合に便利です。mluser ユーティリティは、バッチで追加する場合に便利です。

パスワードがないユーザを作成した場合、そのユーザは Mobile Link で認証されず、接続や同期のためにパスワードが必要ありません。

♦ 1 ユーザに最初の Mobile Link パスワードを設定するには、次の手順に従います (Sybase Central の Admin モードの場合)。

1. Sybase Central から、Mobile Link プラグインを使用して統合データベースに接続します。
2. **[モード]** - **[管理]** を選択します。
3. **[ユーザ]** をクリックします。
4. **[ファイル]** - **[新規]** - **[ユーザ]** を選択します。
5. **Mobile Link ユーザ作成ウィザード**の指示に従います。

♦ 最初の Mobile Link パスワードを設定するには、次の手順に従います (コマンド・ラインの場合)。

1. 各行に 1 人分ずつ、ユーザ名とパスワードを空白スペースで区切って入力したファイルを作成します。
2. コマンド・プロンプトを開き、mluser コマンド・ライン・ユーティリティを実行します。次に例を示します。

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

   このコマンド・ラインでは、-c オプションで、統合データベースへの ODBC 接続を指定します。-f オプションで、ユーザ名とパスワードを含むファイルを指定します。

   「Mobile Link ユーザ認証ユーティリティ (mluser)」 『Mobile Link - サーバ管理』を参照してください。

新しいユーザからの同期

通常、Mobile Link サーバに接続するには、各 Mobile Link クライアントが有効な Mobile Link ユーザ名とパスワードを指定します。

Mobile Link サーバの起動時に -zu+ オプションを指定すると、サーバは未登録のユーザからの同期要求を受け付けて応答できます。ml_user テーブルにリストされていないユーザからの要求を受信すると、要求は処理され、ユーザは ml_user テーブルに追加されます。

Mobile Link クライアントが、現在の ml_user テーブルにないユーザ名で同期を実行した場合に -zu+ を使用すると、Mobile Link はデフォルトで次のアクションを取ります。

- **パスワードを持たない新しいユーザ** ユーザがパスワードを入力しない場合、デフォルトでは、ml_user テーブルに NULL パスワードでユーザ名が追加されます。このユーザはパスワードなしで同期できます。
- **パスワードを持つ新しいユーザ** ユーザがパスワードを入力すると、ユーザ名とパスワードの両方が ml_user テーブルに追加され、Mobile Link システム内で新しいユーザ名が認識されるようになります。これ以降、このユーザは同期するために同じパスワードの指定が必要になります。
- **新しいパスワードを持つ新しいユーザ** 新しいユーザは、[新しいパスワード] フィールドまたは [パスワード] フィールドに情報を入力できます。いずれの場合も、新しいパスワード設定が古いパスワード設定に上書きされ、新しいパスワードを使用して、新しいユーザが Mobile Link システムに追加されます。これ以降、このユーザは同期するために同じパスワードの指定が必要になります。

「-zu オプション」 『Mobile Link - サーバ管理』を参照してください。

未知のユーザによる同期の防止

デフォルトでは、Mobile Link サーバは ml_user テーブルに登録されているユーザのみ認識します。このデフォルト設定には 2 つの利点があります。第 1 に、Mobile Link サーバへの不正なア

クセスによるリスクが減少します。第 2 に、すでに登録されているユーザが間違ったユーザ名やスペルミスのあるユーザ名で誤って接続するのを防ぐことができます。Mobile Link システムに予期しない動作が発生する危険性があるため、誤って接続するような事態は避けてください。

エンド・ユーザに対するパスワード入力の要求

Mobile Link サーバでユーザ認証を無効にするように選択しないかぎり、Mobile Link クライアントから同期を行うすべてのエンド・ユーザは、毎回 Mobile Link ユーザ名とパスワードを入力しなければなりません。

♦ **エンド・ユーザに Mobile Link パスワードの入力を要求するには、次の手順に従います。**

- ユーザ名とパスワードを入力するメカニズムは、Ultra Light クライアントと SQL Anywhere クライアントで異なります。
  - **Ultra Light**　Ultra Light クライアントでは同期時に、同期構造体の password フィールドに有効な値を指定します。組み込みの Mobile Link 同期の場合、有効なパスワードとは、ml_user Mobile Link システム・テーブルにある値と一致するものです。

    アプリケーションでは、エンド・ユーザに対して Mobile Link ユーザ名とパスワードの入力を要求してから、同期を行ってください。

    「Ultra Light 同期パラメータとネットワーク・プロトコル・オプション」『Ultra Light データベース管理とリファレンス』を参照してください。
  - **SQL Anywhere**　ユーザは、dbmlsync コマンド・ラインで有効なパスワードを入力できます。コマンド・ラインに入力しないと、dbmlsync 接続ウィンドウに入力するように要求されます。コマンド・ラインは、同じコンピュータで実行中の他のプロセスから参照できるため、ダイアログで入力する方が安全です。

    認証が失敗すると、ユーザ名とパスワードを再入力するように要求されます。

    「-c オプション」 151 ページを参照してください。

パスワードの変更

Mobile Link では、エンド・ユーザが自身のパスワードを変更するメカニズムが用意されています。インタフェースは、Ultra Light クライアントと SQL Anywhere クライアントで異なります。

♦ **エンド・ユーザに Mobile Link パスワードを入力させるには、次の手順に従います。**

- ユーザ名とパスワードを入力するメカニズムは、Ultra Light クライアントと SQL Anywhere クライアントで異なります。
  - **SQL Anywhere**　dbmlsync コマンド・ラインまたは dbmlsync 接続ウィンドウ (コマンド・ライン・パラメータを指定しない場合) で、有効な既存のパスワードと新しいパスワードを入力します。

    「-mp オプション」 167 ページと「-mn オプション」 166 ページを参照してください。

- **Ultra Light**　アプリケーションでは、同期構造体の password フィールドに既存のパスワードを、new_password フィールドに新しいパスワードを同期時に指定します。

  「Password 同期パラメータ」『Ultra Light データベース管理とリファレンス』と「New Password 同期パラメータ」『Ultra Light データベース管理とリファレンス』を参照してください。

最初のパスワードは、統合サーバで、または最初の同期で設定できます。「ユーザの最初のパスワードを設定する」 11 ページと「新しいユーザからの同期」 12 ページを参照してください。

パスワードをいったん割り当てると、クライアント側でパスワードを空の文字列にリセットすることはできません。

リモート ID

「リモート ID」により、Mobile Link 同期システムのリモート・データベースがユニークに識別されます。

SQL Anywhere または Ultra Light データベースを作成したときは、リモート ID は NULL です。データベースを Mobile Link と同期する場合、Mobile Link は NULL のリモート ID がないかどうかをチェックします。NULL のリモート ID が見つかると、GUID をリモート ID として割り当てます。リモート ID を一度設定すると、手動で変更されるまで、データベースでは同じリモート ID を保持します。

SQL Anywhere リモート・データベースでは、Mobile Link サーバが、リモート ID とサブスクリプションによって同期の進行状況を追跡します。Ultra Light データベースでは、Mobile Link サーバがリモート ID とパブリケーションによって同期の進行状況を追跡します。この情報は、ml_subscription システム・テーブルに保存されます。リモート ID は、同期ごとに Mobile Link サーバ・ログにも記録されます。

複数の同時同期で同じリモート ID が使用されると、Mobile Link サーバはエラーを発行します。

参照

- 「ml_subscription」 『Mobile Link - サーバ管理』

Mobile Link リモート ID の設定

リモート ID は GUID として作成されますが、意味のある名前に変更することもできます。SQL Anywhere と Ultra Light データベースの場合、リモート ID は、ml_remote_id と呼ばれるプロパティとしてデータベースに保存されます。

SQL Anywhere クライアントについては、「リモート ID の設定」 102 ページを参照してください。

Ultra Light クライアントについては、「Ultra Light ml_remote_id オプション」 『Ultra Light データベース管理とリファレンス』を参照してください。

スターター・データベースを複数のロケーションに配備する場合は、リモート ID に NULL が設定されているデータベースを配備するのが最も安全です。事前に移植するようにデータベースを同期した場合は、配備前にリモート ID を NULL に設定し直すことができます。この方法により、リモート・データベースが初めて同期したときに、ユニークなリモート ID が割り当てられるので、リモート ID はユニークになります。また、リモート ID はリモート・セットアップ手順として設定できますが、ユニークでなければなりません。

例

リモート・データベースあたり 1 ユーザに Mobile Link 設定を定義する場合の管理作業を簡素化するには、各リモート・データベースで Mobile Link の 3 つの識別子すべてに同じ番号を使用することが必要な場合があります。たとえば、SQL Anywhere リモート・データベースでは、次のように設定できます。

```
-- Set the MobiLink user name:
  CREATE SYNCHRONIZATION USER "1" ... ;
```

```
-- Set the partition number for DEFAULT GLOBAL AUTOINCREMENT:
  SET OPTION PUBLIC.GLOBAL_DATABASE_ID = '1';

-- Set the MobiLink remote ID:
  SET OPTION PUBLIC.ml_remote_id = '1';
```

スクリプトでのリモート ID と Mobile Link ユーザ名の使用

Mobile Link ユーザ名はユーザを識別し、認証に使用されます。リモート ID は Mobile Link リモート・データベースをユニークに識別します。

多くの同期スクリプトでは、リモート・データベースの識別にオプションでリモート ID (名前付きパラメータ s.remote_id) または Mobile Link ユーザ名 (s.remote_id) を使用できます。リモート ID を使用するといくつかの利点があります (特に Ultra Light の場合)。

リモート・データベースと Mobile Link ユーザが 1 対 1 の関係になるように配備し、Mobile Link ユーザ名がリモート・データベースをユニークに識別する場合は、リモート ID を無視できます。この場合、Mobile Link イベント・スクリプトでは、username パラメータを参照できます。このパラメータは、認証に使用する Mobile Link ユーザ名です。

Mobile Link ユーザが別のデータベースでデータを同期し、各リモートのデータが同じ場合は、同期スクリプトは、Mobile Link ユーザ名を参照できます。Mobile Link ユーザが別のデータベースで別のデータ・セットを同期する場合は、同期スクリプトは、リモート ID を参照する必要があります。

Mobile Link サーバはリモート ID によって同期の進行状況を追跡するため、Ultra Light データベースでは、前のアップロード状態が不明な場合でも、異なるユーザによって同じデータベースを同期できます。この場合、別のユーザごとのローの一部は失われてダウンロードされることがない可能性があるため、タイムスタンプベースのダウンロード・スクリプトでは Mobile Link ユーザ名は参照できなくなります。これを防ぐには、同じリモート・データベースを使用して、統合データベースのテーブルを各ユーザのローにマッピングする必要があります。現在の同期に対するリモート ID に基づいたテーブルのマッピングと統合テーブルのジョインによって、すべてのユーザのすべてのデータを確実にダウンロードできます。

別のスクリプト・バージョンを使用して、異なるデータを別のリモート・データベースに同期することもできます。「スクリプト・バージョン」 『Mobile Link - サーバ管理』を参照してください。

ユーザ認証メカニズムの選択

ユーザの認証は、データを保護するためのセキュリティ・システムの一部です。

Mobile Link では、ユーザ認証メカニズムを選択することができます。インストール環境全体にわたって 1 つのメカニズムを使用する必要はありません。Mobile Link には、インストール環境内の各スクリプト・バージョンが異なる認証メカニズムを使用できるという柔軟性があります。

- **Mobile Link ユーザ認証なし**　パスワード保護が必要でないデータの場合は、インストール環境でユーザ認証を使用しないように選択できます。この場合、Mobile Link ユーザ名は ml_user テーブルに含まれていなければなりませんが、hashed_password カラムは NULL です。
- **組み込みの Mobile Link ユーザ認証**　Mobile Link では、ml_user Mobile Link システム・テーブルに格納されているユーザ名とパスワードを使用して、認証が実行されます。

  組み込みのメカニズムについては、次の項で説明します。
- **カスタム認証**　Mobile Link スクリプトの authenticate_user を使用して、組み込みの Mobile Link ユーザ認証システムを、独自の別のシステムに置き換えることができます。たとえば、使用する統合データベースの管理システムによって、Mobile Link システムの代わりにデータベースのユーザ認証を使用できます。

  「カスタム・ユーザ認証」 21 ページを参照してください。

Mobile Link と関連製品の他のセキュリティ関連機能については、次の項を参照してください。

- 「Mobile Link クライアント／サーバ通信の暗号化」『SQL Anywhere サーバ - データベース管理』
- Ultra Light クライアント：「Ultra Light データベースの保護」『Ultra Light データベース管理とリファレンス』
- SQL Anywhere クライアント：「安全なデータの管理」『SQL Anywhere サーバ - データベース管理』

ユーザ認証アーキテクチャ

Mobile Link のユーザ認証システムは、ユーザ名とパスワードに依存します。組み込みのメカニズムを使用して、Mobile Link サーバに対してユーザ名とパスワードを認証させることも、独自のカスタム・ユーザ認証メカニズムを実装することもできます。

組み込みの認証システムでは、ユーザ名とパスワードの両方が、統合データベース内の ml_user Mobile Link システム・テーブルに格納されます。パスワードは、ハッシュされた状態で格納されます。これは、Mobile Link サーバ以外のアプリケーションからは、ml_user テーブルを読み込んでオリジナルのフォームのパスワードを再構成できないようにするためです。統合データベースにユーザ名とパスワードを追加するには、Sybase Central または mluser ユーティリティを使用するか、Mobile Link サーバの起動時に -zu+ オプションを指定します。

「Mobile Link ユーザの作成と登録」 10 ページを参照してください。

Mobile Link クライアントは、Mobile Link サーバに接続するときに、次の値を提供します。

- **ユーザ名**　Mobile Link ユーザ名。必須です。同期を実行するには、ユーザ名を ml_user システム・テーブルに格納する必要があります。または、Mobile Link サーバを -zu+ オプションを使用して起動し、ml_user テーブルに新しいユーザを追加する必要があります。
- **パスワード**　Mobile Link パスワード。この値は、ユーザが不明な場合、または ml_user Mobile Link システム・テーブル内の対応するパスワードが NULL の場合にのみオプションになります。
- **新しいパスワード**　新しい Mobile Link パスワード。この値はオプションであり、Mobile Link ユーザはこの値を設定することでパスワードを変更できます。

カスタム認証

オプションで、ユーザ認証メカニズムを独自のものに置き換えることができます。

「カスタム・ユーザ認証」 21 ページを参照してください。

認証処理

次に、認証中に発生するイベントの順序を説明します。

1. リモート・アプリケーションは、リモート ID、Mobile Link ユーザ名を使用し、オプションでパスワードと新しいパスワードを使用して、同期要求を開始します。Mobile Link サーバは新しいトランザクションを開始し、begin_connection_autocommit と begin_connection イベントをトリガします。
2. Mobile Link は、リモート ID が現在同期を実行中ではなく、authentication_status が 4000 に設定されていることを確認します。
3. authenticate_user スクリプトを定義している場合は、次のイベントが発生します。
   a. authenticate_user スクリプトを SQL で作成した場合、このスクリプトは authentication_status が 4000 に事前に設定され、Mobile Link ユーザ名が指定されて、オプションでパスワードと新しいパスワードが使用されます。

      authenticate_user スクリプトを Java または .NET で作成した場合、SQL 文が返されてから、この SQL 文は authentication_status が 4000 に事前に設定され、Mobile Link ユーザ名が指定されて、オプションでパスワードと新しいパスワードが使用されます。
   b. authenticate_user スクリプトで例外が発生したり、スクリプトを実行したときにエラーが発生した場合、同期処理は停止します。

   authenticate_user スクリプトまたは返された SQL 文は、2 ～ 4 つの引数を取るストアド・プロシージャの呼び出しでなければいけません。事前に設定した authentication_status 値が最初のパラメータとして渡され、このストアド・プロシージャによって更新されます。最初のパラメータで返された値は、authenticate_user スクリプトからの authentication_status です。
4. authenticate_user_hashed スクリプトが存在すれば、次のイベントが発生します。
   a. パスワードが指定されている場合、ハッシュされた値が計算されます。新しいパスワードが指定されている場合、ハッシュされた値が計算されます。
   b. authenticate_user_hashed スクリプトが、authentication_status の現在値 (authenticate_user が存在しない場合は事前に設定された authentication_status、または authenticate_user スクリプトから返された authentication_status) とハッシュされたパスワードで呼び出されます。動作は、手順 3 と同じです。最初のパラメータで返された値は、authenticate_user_hashed スクリプトの authentication_status として使用されます。
5. Mobile Link サーバは、authenticate_user スクリプトと authenticate_user_hashed スクリプトが存在する場合は、より大きい値を使用し、どちらのスクリプトも存在しない場合は、事前に設定された authentication_status を使用します。
6. Mobile Link サーバは、指定された Mobile Link ユーザ名を ml_user テーブルで問い合わせます。
   a. カスタム・スクリプト authenticate_user または authenticate_user_hashed のいずれかが呼び出されますが、指定された Mobile Link ユーザ名が ml_user テーブルになく、authentication_status が有効な場合 (1000 または 2000) は、Mobile Link ユーザ名が Mobile Link システム・テーブルの ml_user に追加されます。authentication_status が有効でない場合、ml_user は更新されず、エラーが発生します。

b. カスタム・スクリプトが呼び出されず、指定された Mobile Link ユーザ名も ml_user テーブルにない場合は、Mobile Link サーバを -zu+ オプションで起動していれば、指定した Mobile Link ユーザ名が ml_user に追加されます。それ以外の場合は、エラーが発生し、authentication_status が無効に設定されます。

c. カスタム・スクリプトが呼び出され、指定された Mobile Link ユーザ名が ml_user テーブルに存在する場合は、何も実行されません。

d. カスタム・スクリプトが「呼び出されず」、指定された Mobile Link ユーザ名が ml_user テーブルに存在する場合、ml_user テーブルにある値に対してパスワードがチェックされます。パスワードが Mobile Link ユーザ ml_user テーブルにある値と一致する場合、authentication_status は有効に設定されます。一致しない場合、authentication_status は無効に設定されます。

7. authentication_status が有効で、authenticate_user と authenticate_user_hashed のいずれのスクリプトも呼び出されなかった場合に、この Mobile Link ユーザの ml_user テーブルで新しいパスワードを指定すると、パスワードが指定したものに変更されます。

8. authenticate_parameters スクリプトを定義しており、authentication_status が有効である場合 (1000 または 2000)、次のイベントが発生します。

   a. パラメータが authenticate_parameters スクリプトに渡されます。

   b. authenticate_parameters スクリプトが現在の authentication_status より大きい authentication_status 値を返す場合、新しい authentication_status は古い値を上書きします。

9. authentication_status が有効でない場合、同期はアボートされます。

10. modify_user スクリプトを定義している場合はそのスクリプトが呼び出され、指定していた Mobile Link ユーザ名が、このスクリプトによって返された新しい Mobile Link ユーザ名に置き換えられます。

11. Mobile Link サーバは、authentication_status に関係なく、Mobile Link ユーザ認証後に必ずトランザクションをコミットします。authentication_status が有効な場合 (1000 または 2000)、同期は継続されます。authentication_status が有効でない場合、同期はアボートされます。

カスタム・ユーザ認証

組み込みの Mobile Link メカニズム以外のユーザ認証メカニズムを使用するように選択することもできます。カスタム・ユーザ認証メカニズムを使用する理由には、たとえば、次のものがあります。

- 既存のデータベース・ユーザ認証スキームまたは外部認証メカニズムとの統合を含めるため。
- 組み込みの Mobile Link メカニズムにはない、パスワードの最小長や有効期限などのカスタム機能を提供するため。

次の 3 つのカスタム認証ツールがあります。

- mlsrv11 -zu+ オプション
- authenticate_user スクリプトまたは authenticate_user_hashed スクリプト
- authenticate_parameters スクリプト

mlsrv11 -zu+ オプションを使用すると、ユーザの自動追加処理を制御できます。たとえば、-zu+ オプションを指定すると、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+ オプションを必要とするのは、組み込みの Mobile Link 認証だけです。

authenticate_user スクリプト、authenticate_user_hashed スクリプト、authenticate_parameters スクリプトは、デフォルトの Mobile Link ユーザ認証メカニズムを無効にします。正常に認証が行われるユーザは、自動的に ml_user テーブルに追加されます。

authenticate_user スクリプトを使用して、ユーザ ID とパスワードのカスタム認証を作成できます。このスクリプトがあると、組み込みのパスワード比較の代わりにそのメカニズムが実行されます。このスクリプトでは、認証の成功または失敗を示すエラー・コードを返さなければなりません。

Mobile Link は、authenticate_user イベント用に事前に定義されたスクリプトをいくつかインストールします。これらのスクリプトは、LDAP、POP3、IMAP サーバを使用した認証を簡単に行えるようにします。「外部サーバに対する認証」 22 ページを参照してください。

ユーザ ID とパスワード以外の値によるカスタム認証を作成するには、authenticate_parameters を使用します。

参照

- 「-zu オプション」 『Mobile Link - サーバ管理』
- 「authenticate_user 接続イベント」 『Mobile Link - サーバ管理』
- 「authenticate_user_hashed 接続イベント」 『Mobile Link - サーバ管理』
- 「authenticate_parameters 接続イベント」 『Mobile Link - サーバ管理』

Java と .NET のユーザ認証

Java クラスと .NET クラスではアプリケーション・サーバなどのコンピューティング環境で使用されるユーザ名とパスワードの他のソースにアクセスできるため、ユーザ認証は Java と .NET の同期論理で本来使用されているものです。

samples-dir¥MobiLink¥JavaAuthentication ディレクトリに簡単な例があります。*samples-dir¥MobiLink¥JavaAuthentication¥CustEmpScripts.java* 内のサンプル・コードは、単純なユーザ認証システムを実装します。最初の同期時に、Mobile Link ユーザ名が login_added テーブルに追加されます。それ以降の同期時には、login_audit テーブルにローが 1 つ追加されます。このサンプルでは、login_added テーブルにユーザ ID を追加する前のテストは行いません。*samples-dir* の詳細については、「サンプル・ディレクトリ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

ユーザ認証を説明する .NET サンプルについては、「.NET 同期のサンプル」 『Mobile Link - サーバ管理』を参照してください。

外部サーバに対する認証

Mobile Link には、authenticate_user イベントを使用して外部サーバに対する認証を簡単に行えるようにする、事前に定義された Java 同期スクリプトが用意されています。事前に定義されたスクリプトは、次の認証サーバで使用できます。

- JavaMail 1.2 API を使用している POP3 または IMAP サーバ
- Java Naming と Directory Interface (JNDI) を使用している LDAP サーバ

これらのスクリプトをどのように使用するかは、Mobile Link ユーザ名を外部認証システムのユーザ ID に直接マッピングしているかどうかによって決まります。

注意
Sybase Central モデル・モードで [認証] タブを使用して外部サーバへの認証を設定することもできます。「Mobile Link のモデル」 『Mobile Link - クイック・スタート』を参照してください。

Mobile Link ユーザ名をユーザ ID に直接マッピングしている場合

Mobile Link ユーザ名を認証システムの有効なユーザ ID に直接マッピングしている単純なケースでは、authenticate_user 接続イベントに対する応答で、定義済みのスクリプトを直接使用できます。認証コードは、ml_property テーブルに格納されているプロパティに基づいてそのコード自体を初期化します。

♦ 事前に定義されたスクリプトを authenticate_user で直接使用するには、次の手順に従います。

1. 事前に定義された Java 同期スクリプトを Mobile Link の ml_scripts システム・テーブルに追加します。追加するには、ストアド・プロシージャを使用するか、Sybase Central を使用します。

- ml_add_java_connection_script ストアド・プロシージャを使用するには、次のコマンドを実行します。

```
call ml_add_java_connection_script(
  'MyVersion',
  'authenticate_user',
  'ianywhere.ml.authentication.ServerType.authenticate' )
```

  ここで、*MyVersion* はスクリプト・バージョンの名前で、*ServerType* は **LDAP**、**POP3**、または **IMAP** です。

- Sybase Central の**接続スクリプト追加ウィザード**を使用するには、スクリプト・タイプとして **authenticate_user** を選択し、コード・エディタで次のコードを入力します。

```
ianywhere.ml.authentication.ServerType.authenticate
```

  ここで、*ServerType* は **LDAP**、**POP3**、または **IMAP** です。

  「ml_add_java_connection_script システム・プロシージャ」 『Mobile Link - サーバ管理』を参照してください。

2. この認証サーバのプロパティを追加します。

   設定が必要な各プロパティに対し、ml_add_property ストアド・プロシージャを使用します。

```
call ml_add_property(
  'ScriptVersion',
  'MyVersion',
  'property_name',
  'property_value' )
```

   ここで、*MyVersion* はスクリプト・バージョンの名前で、*property_name* は認証サーバによって決まります。また、*property_value* は使用しているアプリケーションに適切な値です。この呼び出しを、設定の必要な各プロパティに対して繰り返し行います。

   「外部認証識別符号プロパティ」 24 ページと「ml_add_property システム・プロシージャ」『Mobile Link - サーバ管理』を参照してください。

Mobile Link ユーザ名をユーザ ID に直接マッピングしていない場合

Mobile Link ユーザ名がユーザ ID と同じでない場合は、コードを間接的に呼び出す必要があり、ユーザ ID を ml_user 値から抽出するか、マッピングしなければなりません。これを行うには、Java クラスを作成します。

「Java による同期スクリプトの作成」 『Mobile Link - サーバ管理』を参照してください。

次に、簡単な例を示します。extractUserID メソッド内のコードは、ml_user 値をユーザ ID にマッピングする方法によって異なるため、この例では省きます。すべての作業は、認証クラスの "authenticate" メソッドで行われます。

```
package com.mycompany.mycode;

import ianywhere.ml.authentication.*;
import ianywhere.ml.script.*;

public class MLEvents
{
   private DBConnectionContext _context;
```

```
    private POP3 _pop3;

    public MLEvents( DBConnectionContext context )
    {
        _context = context;
        _pop3 = new POP3( context );
    }

    public void authenticateUser(
      InOutInteger status,
      String userID,
      String password,
      String newPassword )
    {
        String realUserID = extractUserID( userID );
        _pop3.authenticate( status, realUserID, password, newPassword );
    }

    private String extractUserID( String userID )
    {
        // code here to map ml_user to a "real" POP3 user
    }
}
```

この例では、初期化プロパティを検出できるようにするために、POP3 オブジェクトを DBConnectContext オブジェクトで初期化する必要があります。この方法でオブジェクトを初期化しない場合は、コードにプロパティを設定する必要があります。次に例を示します。

```
POP3 pop3 = new POP3();
pop3.setServerName( "smtp.sybase.com" );
pop3.setServerPort( 25 );
```

これはどのような認証クラスにも適用されますが、プロパティはクラスによって異なります。

外部認証識別符号プロパティ

Mobile Link では、特に LDAP の場合において、可能なかぎり適切なデフォルトを用意しています。設定できるプロパティはさまざまですが、次に基本的なプロパティを説明します。

POP3 認証識別符号

mail.pop3.host	サーバのホスト名
mail.pop3.port	ポート番号 (デフォルトの 110 を使用する場合は省略可能)

http://java.sun.com/products/javamail/javadocs/com/sun/mail/pop3/package-summary.html を参照してください。

IMAP 認証識別符号

mail.imap.host	サーバのホスト名
mail.imap.port	ポート番号 (デフォルトの 143 を使用する場合は省略可能)

http://java.sun.com/products/javamail/javadocs/com/sun/mail/imap/package-summary.html を参照してください。

LDAP 認証識別符号

java.naming.provider.url	ldap://ops-yourLocation/dn=sybase,dn=com などの LDAP サーバの URL

詳細については、JNDI のマニュアルを参照してください。

Mobile Link クライアント・ユーティリティ

目次

Mobile Link クライアント・ユーティリティの概要

Mobile Link には、次の 2 つのクライアント・ユーティリティがあります。

- 「ActiveSync プロバイダ・インストール・ユーティリティ (mlasinst)」 29 ページ
- 「Mobile Link ファイル転送ユーティリティ (mlfiletransfer)」 32 ページ

次の各項も参照してください。

- Ultra Light ユーティリティ：「Ultra Light ユーティリティ」 『Ultra Light データベース管理とリファレンス』
- Mobile Link サーバ・ユーティリティ：「Mobile Link ユーティリティ」 『Mobile Link - サーバ管理』
- SQL Anywhere のその他のユーティリティ：「データベース管理ユーティリティ」 『SQL Anywhere サーバ - データベース管理』

ActiveSync プロバイダ・インストール・ユーティリティ (mlasinst)

ActiveSync (Windows Vista での名称は Windows Mobile Device Center) 用 Mobile Link プロバイダをインストールするか、Windows Mobile デバイス上で Ultra Light アプリケーションを登録してインストールします。

構文

mlasinst [*options*] [[*src*] *dst name class* [*args*]]

オプション	説明
-d	最初にアプリケーションを無効にします。
-k *path*	デスクトップ・プロバイダ *mlasdesk.dll* のロケーションを指定します。 デフォルトでは、このファイルは *install-dir¥bin32* にあります。 エンド・ユーザ (通常は SQL Anywhere 全体がインストールされていないユーザ) は、Mobile Link ActiveSync プロバイダのインストール時に、-k の指定が必要になる場合があります。
-l *filename*	アクティビティ・ログ・ファイルの名前を指定します。
-n	アプリケーションを登録するが、デバイスにはコピーしない このオプションを指定すると、Mobile Link ActiveSync プロバイダのインストールに加えてアプリケーションが登録されますが、アプリケーションはデバイスにコピーはされません。アプリケーションに複数のファイルがある場合 (静的ライブラリではなく Ultra Light ランタイム・ライブラリ DLL を使用するようにコンパイルされている場合など)、または他の方法でアプリケーションをデバイスにコピーする場合は、このオプションを指定します。
-t *n*	デスクトップ・プロバイダがクライアントからの応答を待つ秒数を指定します。この時間が経過すると、タイムアウトします。デフォルトは 30 です。
-u	ActiveSync 用 Mobile Link プロバイダをアンインストールする。 このオプションを指定すると、Mobile Link ActiveSync プロバイダ用に登録されたアプリケーションの登録がすべて解除され、Mobile Link ActiveSync プロバイダがアンインストールされます。この操作によってデスクトップ・コンピュータやデバイスからファイルが削除されることはありません。デバイスがデスクトップに接続されていない場合は、エラーが返されます。

オプション	説明
-v *path*	デバイス・プロバイダ *mlasdev.dll* のロケーションを指定します。デフォルトでは、このファイルは *install-dir¥CE* 内のプラットフォーム固有のディレクトリ内で検索されます。 エンド・ユーザ (通常は SQL Anywhere 全体がインストールされていないユーザ) は、Mobile Link ActiveSync プロバイダのインストール時に、-v の指定が必要になる場合があります。

その他のパラメータ	説明
src	アプリケーションをデバイスにコピーするためのソース・ファイル名とパスを指定する。このパラメータを指定するのは、アプリケーションを登録してデバイスにコピーする場合のみです。-n オプションを使用する場合は、このパラメータを指定しないでください。
dst	デバイス上でアプリケーションに使用するコピー先ファイル名とパスを指定する。
name	ActiveSync がアプリケーションを参照するときに使用する名前を指定します。
class	アプリケーションの登録済み Windows クラス名を指定する。
args	ActiveSync によってアプリケーションが起動されるときにアプリケーションに渡すコマンド・ライン引数を指定する。

備考

このユーティリティによって、ActiveSync 用の Mobile Link プロバイダがインストールされます。プロバイダには、デスクトップ上で実行されるコンポーネント (*mlasdesk.dll*) と、Windows Mobile デバイスに展開されるコンポーネント (*mlasdev.dll*) の両方が含まれています。mlasinst ユーティリティは、デスクトップ・プロバイダの現在のロケーションを示すレジストリ・エントリを作成し、デバイス・プロバイダをデバイスにコピーします。

また、*mlasinst* ユーティリティに追加の引数を指定すると、Ultra Light アプリケーションを Windows Mobile デバイスに登録してインストールできます。さらに、ActiveSync ソフトウェアを使用して Ultra Light アプリケーションの登録とインストールを行う方法もあります。

ライセンス要件に応じて、このアプリケーションをデスクトップ・コンポーネントやデバイス・コンポーネントとともにエンド・ユーザに提供できる場合があります。この場合、エンド・ユーザは、ActiveSync で使用できるようにアプリケーションのコピーを作成できます。

ActiveSync プロバイダをインストールするには、リモート・デバイスに接続してください。

ActiveSync プロバイダ・インストール・ユーティリティの使用方法の詳細は、次の項を参照してください。

- SQL Anywhere：「ActiveSync 用 Mobile Link プロバイダのインストール」 125 ページ
- Ultra Light：「Windows Mobile の ActiveSync」 『Ultra Light データベース管理とリファレンス』

例

次のコマンドは、デフォルト引数を使用して、ActiveSync 用の Mobile Link プロバイダをインストールします。アプリケーションの登録は行いません。正常にインストールするには、デバイスをデスクトップ・マシンに接続してください。

```
mlasinst
```

次のコマンドは、ActiveSync 用の Mobile Link プロバイダをアンインストールします。正常にアンインストールするには、デバイスをデスクトップ・マシンに接続してください。

```
mlasinst -u
```

次のコマンドは、ActiveSync 用の Mobile Link プロバイダがまだインストールされていない場合はインストールし、アプリケーション *myapp.exe* を登録します。また、*c:¥My Files¥myapp.exe* ファイルをデバイス上の *¥Program Files¥myapp.exe* にコピーします。-p -x 引数は、ActiveSync によって起動されるときの *myapp.exe* に対するコマンド・ライン・オプションです。このコマンドは、1 行に入力してください。

```
mlasinst "C:¥My Files¥myapp.exe" "¥Program Files¥myapp.exe"
  "My Application" MYAPP -p -x
```

参照

- 「ActiveSync 同期の使用」 124 ページ
- 「Ultra Light 同期パラメータとネットワーク・プロトコル・オプション」 『Ultra Light データベース管理とリファレンス』

Mobile Link ファイル転送ユーティリティ (mlfiletransfer)

Mobile Link を介してファイルをダウンロードします。

構文

mlfiletransfer [*options*] *file*

オプション	説明
-ap *param1*, ...	Mobile Link 認証パラメータ。「認証パラメータ」『Mobile Link - サーバ管理』を参照してください。
-dp *path*	ダウンロードしたファイルを保存するローカル・パス。デフォルトでは、ダウンロードしたファイルは Windows Mobile ではルート・ディレクトリに、その他のプラットフォームでは現在のディレクトリに保存されます。
-df *filename*	ダウンロードしたファイルのローカル名。サーバで使用していたファイル名とは異なるファイル名をクライアントで使用する場合はこのオプションを使用します。デフォルトでは、サーバでのファイル名が使用されます。
-f	強制的にダウンロードし、ローカル・ファイルも最新の状態にします。前の部分的なダウンロードは破棄されます。
-g	ダウンロードの進行状況を表示します。
-p *password*	Mobile Link ユーザ名のパスワード。
-q	クワイエット・モード。メッセージを表示しません。
-r	ダウンロードの再開を有効にします。このオプションを設定すると、ユーティリティは、通信エラーまたはユーザのキャンセルによって中断した前の部分的なダウンロードを再開します。サーバのファイルが部分的なファイルよりも新しい場合、部分的なファイルは破棄されます。このオプションよりも、-f オプションが優先されます。
-u *username*	Mobile Link ユーザ名。このオプションは必須です。
-v *version*	スクリプト・バージョン。このオプションは必須です。

オプション	説明
-x *protocol*(*options*)	*protocol* には、**tcpip**、**tls**、**http**、または **https** のいずれかを指定します。このオプションは必須です。 使用できる protocol-options は、プロトコルによって異なります。各プロトコルのオプションのリストについては、「Mobile Link クライアント・ネットワーク・プロトコル・オプションの一覧」 37 ページを参照してください。
file	転送するファイルのサーバでのファイル名。ファイルのロケーションは mlsrv11 -ftr オプション (Mobile Link サーバを起動するときに使用) によって決まるので、パスを含めないでください。Mobile Link は、-ftr ディレクトリの username サブディレクトリでファイルを検索します。このサブディレクトリにファイルが見つからない場合は、-ftr ディレクトリを検索します。ファイルがどちらのディレクトリでも見つからない場合は、エラーが生成されます。 「-ftr オプション」 『Mobile Link - サーバ管理』を参照してください。

備考

このユーティリティは、初めてリモート・データベースを作成する場合、リモート・デバイスでソフトウェアをアップグレードする必要がある場合などに、ファイルをダウンロードするときに役立ちます。

このユーティリティを使用するには、-ftr オプションを使用して Mobile Link サーバを起動する必要があります。-ftr オプションは、転送するファイルのルート・ディレクトリを作成し、登録されている Mobile Link ユーザごとにサブディレクトリを作成します。

Ultra Light ユーザは、Ultra Light ランタイムで MLFileTransfer メソッドも使用できます。「Mobile Link ファイル転送の使用」 『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「-ftr オプション」 『Mobile Link - サーバ管理』
- 「authenticate_file_transfer 接続イベント」 『Mobile Link - サーバ管理』

例

次のコマンドは、Mobile Link サーバを CustDB サンプル・データベースに接続します。-ftr %SystemRoot%¥system32 オプションは、要求されたファイルがないかどうか、*Windows¥system32* ディレクトリをモニタし始めるように Mobile Link サーバに指示します。この例では、Mobile Link サーバはまず *C:¥Windows¥system32¥mobilink-username* ディレクトリでファイルを探します。ファイルがない場合、*C:¥Windows¥system32* ディレクトリで探します。通常は、ファイルがないかどうか、*Windows¥system32* フォルダを Mobile Link サーバにモニタさせる必要はあり

ません。この例では、同じ場所にあるメモ帳ユーティリティを転送できるように、*Windows ¥system32* ディレクトリを使用しています。

```
mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -zu+ -ftr %SystemRoot%¥system32
```

次のコマンドは、mlfiletransfer ユーティリティを実行します。このコマンドによって、Mobile Link サーバはローカル・ディレクトリに *notepad.exe* をダウンロードします。

```
MLFileTransfer -u 1 -v "custdb 10.0" -x tcpip notepad.exe
```


Mobile Link クライアント・ネットワーク・プロトコル・オプション

目次

Mobile Link クライアント・ネットワーク・プロトコル・オプションの一覧

この項では、Mobile Link クライアントを Mobile Link サーバに接続するときに使用できるネットワーク・プロトコル・オプションについて説明します。次のように、複数の Mobile Link クライアント・ユーティリティで Mobile Link クライアント・ネットワーク・プロトコル・オプションが使用されます。

クライアント・ネットワーク・プロトコル・オプションを使用するユーティリティ	参照先
dbmlsync	「CommunicationAddress (adr) 拡張オプション」 197 ページ
Ultra Light	「Stream Parameters 同期パラメータ」 『Ultra Light データベース管理とリファレンス』 または「Ultra Light 同期ユーティリティ (ulsync)」 『Ultra Light データベース管理とリファレンス』の-x オプション
Ultra Light J	「Ultra Light J 同期ストリームのネットワーク・プロトコルのオプション」 『Ultra Light J』
リダイレクタ	「リダイレクタのプロパティの設定 (サーバ・グループをサポートするリダイレクタの場合)」 『Mobile Link - サーバ管理』または「リダイレクタのプロパティの設定 (サーバ・グループをサポートしないリダイレクタの場合)」 『Mobile Link - サーバ管理』の Mobile Link ディレクティブ
Mobile Link モニタ	「Mobile Link モニタの起動」 『Mobile Link - サーバ管理』
Mobile Link ファイル転送	「Mobile Link ファイル転送ユーティリティ (mlfiletransfer)」 32 ページ
Mobile Link Listener	「Windows デバイス用の Listener ユーティリティ」 『Mobile Link - サーバ起動同期』の -x
QAnywhere Agent	「-x オプション」 『QAnywhere』

クライアントが使用する同期プロトコルと一致するネットワーク・プロトコルを選択してください。Mobile Link サーバの接続オプションを設定する方法については、「-x オプション」 『Mobile Link - サーバ管理』を参照してください。

プロトコル・オプション

- **TCP/IP プロトコル・オプション**　**tcpip** オプションを指定する場合は、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。

TCP/IP プロトコル・オプション	詳細の参照先
buffer_size=*bytes*	「buffer_size」 43 ページ
client_port=*nnnnn*[-*mmmmm*]	「client_port」 50 ページ
e2ee_type={**rsa**|**ecc**}	「e2ee_type」 53 ページ
e2ee_public_key=*file*	「e2ee_public_key」 54 ページ
compression={**zlib**|**none**}	「compression」 51 ページ
host=*hostname*	「host」 57 ページ
network_leave_open={**off**|**on**}	「network_leave_open」 63 ページ
network_name=*name*	「network_name」 64 ページ
port=*portnumber*	「port」 67 ページ
timeout=*seconds*	「timeout」 71 ページ
zlib_download_window_size=*window-bits*	「zlib_download_window_size」 80 ページ
zlib_upload_window_size=*window-bits*	「zlib_upload_window_size」 81 ページ

- **セキュリティ付きの TCP/IP プロトコル**　**tls** オプション (TLS セキュリティを使用する TCP/IP) を指定すると、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。

TLS プロトコル・オプション	詳細の参照先
buffer_size=*bytes*	「buffer_size」 43 ページ
certificate_company=*company_name*	「certificate_company」 45 ページ
certificate_name=*name*	「certificate_name」 47 ページ
certificate_unit=*company_unit*	「certificate_unit」 49 ページ
client_port=*nnnnn*[-*mmmmm*]	「client_port」 50 ページ
compression={**zlib**|**none**}	「compression」 51 ページ
e2ee_type={**rsa**|**ecc**}	「e2ee_type」 53 ページ
e2ee_public_key=*file*	「e2ee_public_key」 54 ページ
fips={**y**|**n**}	「fips」 55 ページ

TLS プロトコル・オプション	詳細の参照先
host=*hostname*	「host」 57 ページ
network_leave_open={**off**|**on**}	「network_leave_open」 63 ページ
network_name=*name*	「network_name」 64 ページ
port=*portnumber*	「port」 67 ページ
timeout=*seconds*	「timeout」 71 ページ
tls_type={**rsa**|**ecc**}	「tls_type」 73 ページ
trusted_certificates=*filename*	「trusted_certificates」 75 ページ
zlib_download_window_size=*window-bits*	「zlib_download_window_size」 80 ページ
zlib_upload_window_size=*window-bits*	「zlib_upload_window_size」 81 ページ

- **HTTP プロトコル**　**http** オプションを指定する場合は、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。

HTTP プロトコル・オプション	詳細の参照先
buffer_size=*number*	「buffer_size」 43 ページ
client_port=*nnnnn*[-*mmmmm*]	「client_port」 50 ページ
compression={**zlib**|**none**}	「compression」 51 ページ
custom_header=*header*	「custom_header」 52 ページ
e2ee_type={**rsa**|**ecc**}	「e2ee_type」 53 ページ
e2ee_public_key=*file*	「e2ee_public_key」 54 ページ
http_password=*password*	「http_password」 58 ページ
http_proxy_password=*password*	「http_proxy_password」 59 ページ
http_proxy_userid=*userid*	「http_proxy_userid」 60 ページ
http_userid=*userid*	「http_userid」 61 ページ
host=*hostname*	「host」 57 ページ
network_leave_open={**off**|**on**}	「network_leave_open」 63 ページ

HTTP プロトコル・オプション	詳細の参照先
network_name=*name*	「network_name」 64 ページ
persistent={**off**|**on**}	「persistent」 66 ページ
port=*portnumber*	「port」 67 ページ
proxy_host=*proxy-hostname-or-ip*	「proxy_host」 68 ページ
proxy_port=*proxy-portnumber*	「proxy_port」 69 ページ
set_cookie=*cookie-name*=*cookie-value*	「set_cookie」 70 ページ
timeout=*seconds*	「timeout」 71 ページ
url_suffix=*suffix*	「url_suffix」 77 ページ
version=*HTTP-version-number*	「version」 79 ページ
zlib_download_window_size=*window-bits*	「zlib_download_window_size」 80 ページ
zlib_upload_window_size=*window-bits*	「zlib_upload_window_size」 81 ページ

- **HTTPS プロトコル**　**https** オプション (RSA 暗号化を使用する HTTP) 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。

HTTPS プロトコル・オプション	詳細の参照先
buffer_size=*number*	「buffer_size」 43 ページ
certificate_company=*company_name*	「certificate_company」 45 ページ
certificate_name=*name*	「certificate_name」 47 ページ
certificate_unit=*company_unit*	「certificate_unit」 49 ページ
client_port=*nnnnn*[**-***mmmmm*]	「client_port」 50 ページ
compression={**zlib**|**none**}	「compression」 51 ページ
custom_header=*header*	「custom_header」 52 ページ
e2ee_type={**rsa**|**ecc**}	「e2ee_type」 53 ページ
e2ee_public_key=*file*	「e2ee_public_key」 54 ページ
fips={**y**|**n**}	「fips」 55 ページ

HTTPS プロトコル・オプション	詳細の参照先
host=*hostname*	「host」 57 ページ
http_password=*password*	「http_password」 58 ページ
http_proxy_password=*password*	「http_proxy_password」 59 ページ
http_proxy_userid=*userid*	「http_proxy_userid」 60 ページ
http_userid=*userid*	「http_userid」 61 ページ
network_leave_open={**off**|**on**}	「network_leave_open」 63 ページ
network_name=*name*	「network_name」 64 ページ
persistent={**off**|**on**}	「persistent」 66 ページ
port=*portnumber*	「port」 67 ページ
proxy_host=*proxy-hostname-or-ip*	「proxy_host」 68 ページ
proxy_port=*proxy-portnumber*	「proxy_port」 69 ページ
set_cookie=*cookie-name*=*cookie-value*	「set_cookie」 70 ページ
timeout=*seconds*	「timeout」 71 ページ
tls_type={**rsa**|**ecc**}	「tls_type」 73 ページ
trusted_certificates=*filename*	「trusted_certificates」 75 ページ
url_suffix=*suffix*	「url_suffix」 77 ページ
version=*HTTP-version-number*	「version」 79 ページ
zlib_download_window_size=*window-size*	「zlib_download_window_size」 80 ページ
zlib_upload_window_size=*window-bits*	「zlib_upload_window_size」 81 ページ

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

buffer_size

ネットワークに書き込む前にバッファする最大バイト数を指定します。HTTP と HTTPS では、このバイト数が、HTTP 要求の本文の最大サイズに変換されます。

構文

buffer_size=*bytes*

プロトコル

- TCPIP、TLS、HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

- Palm - 4K
- CE - 16K
- その他のプラットフォーム - 64K

備考

一般に、HTTP と HTTPS のバッファ サイズが大きくなるほど、HTTP 要求応答のサイクルの数は減りますが、必要なメモリ量は増えます。

TCPIP と TLS でも、バッファ サイズが大きくなるほどパフォーマンスは向上しますが、必要なメモリ量が増えます。ただし、パフォーマンスの差は、HTTP ほど大きくありません。

単位はバイトです。キロバイトには K、メガバイトには M、ギガバイトには G を指定してください。

最大値は 1 G です。

このオプションは、クライアントからの要求のサイズを制御します。Mobile Link からの応答のサイズは制限しません。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の例は、最大バイト数を 32 K に設定します。

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e "adr=buffer_size=32K"
```

Embedded SQL または C++ で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
synch_info.stream_parms = TEXT("buffer_size=32K");
```


certificate_company

このオプションを指定した場合、証明書に記されている組織フィールドがこの値と一致する場合にだけ、アプリケーションはサーバ証明書を受け入れます。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」『SQL Anywhere 11 - 紹介』を参照してください。

構文

certificate_company=*organization*

プロトコル

- TLS、HTTPS

デフォルト

なし

備考

Mobile Link クライアントは認証局が署名した証明書をすべて信頼するため、同じ認証局が他の会社用に発行した証明書も信頼してしまうことがあります。識別方法がないままだと、クライアントは競争相手の Mobile Link サーバを自分の会社のものだと勘違いし、誤って機密性の高い情報を送信してしまう可能性があります。このオプションによって追加の検証が指定され、証明書の識別情報部分にある組織フィールドが、指定した特定の値と照合されます。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「Mobile Link クライアント／サーバ通信の暗号化」『SQL Anywhere サーバ - データベース管理』
- 「証明書フィールドの確認」『SQL Anywhere サーバ - データベース管理』
- 「-x オプション」『Mobile Link - サーバ管理』
- 「trusted_certificates」 75 ページ
- 「certificate_name」 47 ページ
- 「certificate_unit」 49 ページ

例

3 つの識別情報フィールドすべてを検査し、指定した値だけを受け入れるように SQL Anywhere クライアントに指示する例を示します。この例は 3 つのフィールドすべてを確認します。フィールドを 1 つまたは 2 つだけ確認するように選択することもできます。

たとえば、SQL Anywhere クライアントが存在する場合、証明書の検証をサブスクリプションで次のように設定できます。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
FOR user01
TO test_pub
ADDRESS 'port=3333;
  trusted_certificates=certicom.crt;
  certificate_company=Sybase, Inc.;
  certificate_unit=iAnywhere;certificate_name=sample'
```

C または C++ の Embedded SQL で作成された Ultra Light アプリケーションでは、次のように証明書の検証を設定できます。ここでは、データベースの作成時に、信頼できる証明書がデータベースにインストールされていることを前提としています。

```
ul_synch_info info;
info.stream = "tls";
info.stream_parms = UL_TEXT("port=9999;")
   UL_TEXT ( "certificate_company=Sybase, Inc.;" )
   UL_TEXT ( "certificate_unit=iAnywhere;" )
   UL_TEXT ( "certificate_name=sample;" );
...
ULSynchronize( &info );
```


certificate_name

このオプションを指定した場合、証明書に記されている通称フィールドがこの値と一致する場合にだけ、アプリケーションはサーバ証明書を受け入れます。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

構文

certificate_name=*common-name*

プロトコル

- TLS、HTTPS

デフォルト

なし

備考

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」 『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「Mobile Link クライアント／サーバ通信の暗号化」 『SQL Anywhere サーバ - データベース管理』
- 「証明書フィールドの確認」 『SQL Anywhere サーバ - データベース管理』
- 「-x オプション」 『Mobile Link - サーバ管理』
- 「trusted_certificates」 75 ページ
- 「certificate_company」 45 ページ
- 「certificate_unit」 49 ページ

例

HTTPS プロトコルの RSA 暗号化を設定する例を示します。これは、サーバとクライアント両方での設定が必要です。各コマンドは、1 行に入力する必要があります。

サーバでは、実装は次のようになります。

```
mlsrv11
  -c "dsn=SQL Anywhere 11 Demo;uid=DBA;pwd=sql"
  -x https(
    port=9999;
```



```
    identity=c:¥sa10¥bin32¥rsaserver.id;
    identity_password=test)
```

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync
  -c "dsn=mydb;uid=DBA;pwd=sql"
  -e "ctp=https;
      adr='port=9999;
        trusted_certificates=c:¥sa10¥bin32¥rsaroot.crt;
        certificate_name=RSA Server'"
```

Ultra Light クライアントでは、実装は次のようになります。

```
    info.stream = "https";
    info.stream_parms = TEXT(
      "port=9999;
      trusted_certificates=¥sa10¥bin32¥rsaroot.crt;
      certificate_name=RSA Server");
```


certificate_unit

このオプションを指定した場合、証明書に記されている組織単位フィールドがこの値と一致する場合にだけ、アプリケーションはサーバ証明書を受け入れます。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

構文

certificate_unit=*organization-unit*

プロトコル

- TLS、HTTPS

デフォルト

なし

備考

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」 『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「Mobile Link クライアント／サーバ通信の暗号化」 『SQL Anywhere サーバ - データベース管理』
- 「証明書フィールドの確認」 『SQL Anywhere サーバ - データベース管理』
- 「-x オプション」 『Mobile Link - サーバ管理』
- 「trusted_certificates」 75 ページ
- 「certificate_company」 45 ページ
- 「certificate_name」 47 ページ

例

セキュリティの例については、「certificate_name」 47 ページと「trusted_certificates」 75 ページを参照してください。

client_port

通信に使用するクライアント・ポートの範囲を指定します。

構文

client_port=*nnnnn*[**-***mmmmm*]

プロトコル

- TCPIP、TLS、HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

なし

備考

可能なポート番号の範囲を示す開始値と終了値を指定します。クライアントを特定のポート番号に制限するには、*nnnnn* と *mmmmm* に同じ番号を指定します。値を 1 つだけ指定すると、範囲の上限値は初期値より 100 大きくなり、ポート数の合計は 101 になります。

このオプションは、ファイアウォール内のクライアントがファイアウォール外の Mobile Link サーバと通信する場合に役立ちます。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の例は、許容できるクライアント・ポートとして 10000 台のポート範囲を設定します。

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e "adr=client_port=10000-19999"
```

Embedded SQL または C++ で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
synch_info.stream_parms = TEXT("client_port=10000-19999");
```


compression

Mobile Link サーバと Mobile Link クライアント間の同期ストリームの圧縮のオンとオフを切り替えます。

構文

compression= { zlib | none }

プロトコル

- TCPIP、TLS、HTTP、HTTPS

サポートに関する注意

- Palm OS または Symbian ではサポートされません。
- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

Ultra Light では、compression はデフォルトでオフになっています。

dbmlsync では、デフォルトで zlib compression がオンになっています。

> SQL Anywhere クライアントでは、圧縮をオフにすると、データはまったく難読化されなくなります。セキュリティが問題になる場合は、ストリームを暗号化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」『SQL Anywhere サーバ - データベース管理』を参照してください。

備考

zlib compression を使用する場合は、zlib_download_window_size オプションと zlib_upload_window_size オプションを使用して、アップロードとダウンロードの圧縮を設定できます。この 2 つのオプションを使用して、アップロードまたはダウンロードの圧縮をオフにすることもできます。

Ultra Light で zlib compression を使用するには、*mlczlib10.dll* を配備し、また C++ の場合はアプリケーションで ULEnableZlibSyncCompression(sqlca) を呼び出す必要があります。

参照

- 「zlib_download_window_size」 80 ページ
- 「zlib_upload_window_size」 81 ページ

例

次のオプションでは、アップロードの圧縮のみを設定できます。アップロードのウィンドウ・サイズは 9 に設定します。

```
"compression=zlib;zlib_download_window_size=0;zlib_upload_window_size=9"
```


custom_header

カスタム HTTP ヘッダを指定します。

構文

custom_header=*header*

HTTP ヘッダの形式は、*header-name*: *header-value* です。

プロトコル

- HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

なし

備考

カスタム HTTP ヘッダを指定すると、クライアントは HTTP 要求を送信するごとにそのヘッダを含めます。複数のカスタム・ヘッダを指定するには、セミコロン (;) を区切り文字として使用しながら、custom_header を複数回使用してください。たとえば、**custom_header=***header1*:*value1*; **customer_header=***header2*:*value2* のようになります。

カスタム・ヘッダは、カスタム・ヘッダが必要なサード・パーティ・ツールとの対話を同期クライアントが行う場合に便利です。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

例

一部の HTTP プロキシは、すべての要求に対して特殊なヘッダを含めること要求します。次の例では、Embedded SQL または C++ の Ultra Light アプリケーション内の値 ProxyUser に MyProxyHdr というカスタム HTTP ヘッダを設定します。

```
info.stream = "http";
info.stream_parms = TEXT(
  "host=www.myhost.com;proxy_host=www.myproxy.com;
  custom_header=MyProxyHdr:ProxyUser");
```


e2ee_type

エンドツーエンド暗号化のキー交換に使用する非対称アルゴリズムを指定します。

構文

e2ee_type= { **rsa** | **ecc** }

プロトコル

TCPIP、TLS、HTTP、HTTPS

デフォルト

RSA

備考

rsa と **ecc** のどちらかにし、サーバで指定した値と一致させてください。

参照

- 「e2ee_public_key」 54 ページ
- 「-x オプション」 『Mobile Link - サーバ管理』
- 「キー・ペア・ジェネレータ・ユーティリティ (createkey)」 『SQL Anywhere サーバ - データベース管理』

例

次の例は、Ultra Light クライアント用のエンドツーエンド暗号化を示します。

```
info.stream = "https";
info.stream_parms =
"tls_type=rsa;trusted_certificates=rsaroot.crt;e2ee_type=rsa;e2ee_public_key=rsapublic.pem"
```


e2ee_public_key

エンドツーエンド暗号化に使用する、サーバの PEM でコード化されたパブリック・キーを指定します。

構文

e2ee_public_key=*file*

プロトコル

TCPIP、TLS、HTTP、HTTPS

デフォルト

なし

備考

キー・タイプは、e2ee_type パラメータで指定したタイプと一致させてください。

エンドツーエンド暗号化を有効にするためには、このオプションが必須です。

エンドツーエンド暗号化は TLS/HTTPS プロトコル・オプション fips と組み合わせて使用することもできます。このオプションは、ECC の使用時にはサポートされません。「fips」 55 ページを参照してください。

参照

- 「e2ee_type」 53 ページ
- 「-x オプション」 『Mobile Link - サーバ管理』
- 「キー・ペア・ジェネレータ・ユーティリティ (createkey)」 『SQL Anywhere サーバ - データベース管理』

例

次の例は、Ultra Light クライアント用のエンドツーエンド暗号化を示します。

```
info.stream = "https";
info.stream_parms =
"tls_type=rsa;trusted_certificates=rsaroot.crt;e2ee_type=rsa;e2ee_public_key=rsapublic.pem"
```


fips

TLS 暗号化とエンドツーエンド暗号化に FIPS 認定の暗号化の実装を使用します。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

構文

fips={ y | n }

プロトコル

HTTPS、TLS

デフォルト

N

備考

FIPS は、RSA 暗号化でのみでサポートされています。

非 FIPS クライアントは、FIPS サーバに接続することはできません。逆についても同様です。

このオプションは、エンドツーエンド暗号化と組み合わせて使用できます。fips が **y** に設定されている場合、Mobile Link クライアントは、FIPS 140-2 基準を満たした RSA 実装および AES 実装を使用します。このオプションは、ECC の使用時にはサポートされません。「e2ee_type」 53 ページと「e2ee_public_key」 54 ページを参照してください。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

参照

- 「tls_type」 73 ページ
- 「e2ee_type」 53 ページ

例

TCP/IP プロトコルの FIPS 認定の RSA 暗号化を設定する例を示します。これは、サーバとクライアント両方での設定が必要です。各コマンドは、1 行に入力する必要があります。

サーバでは、実装は次のようになります。

```
mlsrv11
  -c "dsn=SQL Anywhere 11 Demo;uid=DBA;pwd=sql"
  -x tls(
    port=9999;
    tls_type=rsa;
    fips=y;
    identity=c:¥sa11¥bin32¥rsaserver.id;
    identity_password=test)
```

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e
  "CommunicationType=tls;
   CommunicationAddress=
     'tls_type=rsa;
      fips=y;
      trusted_certificates=¥rsaroot.crt;
      certificate_name=RSA Server'"
```

C または C++ の Embedded SQL で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
info.stream = "tls";
info.stream_parms = TEXT(
  "tls_type=rsa;
   fips=y;
   trusted_certificates=¥rsaroot.crt;
   certificate_name=RSA Server");
```


host

Mobile Link サーバを実行中のコンピュータ、または、Web サーバを介して同期する場合は Web サーバを実行中のコンピュータのホスト名または IP アドレスを指定します。

構文

host=*hostname-or-ip*

プロトコル

- TCPIP、TLS、HTTP、HTTPS

デフォルト

- Windows Mobile - デフォルト値は、デバイスに ActiveSync パートナーシップが設定されているデスクトップ・コンピュータの IP アドレスです。
- 他のすべてのデバイス - デフォルトは **localhost** です。

備考

Windows Mobile では、localhost を使用しないでください。これはリモート・デバイス自体を示します。デフォルト値を使用すると、Windows Mobile デバイスに ActiveSync パートナーシップが設定されているデスクトップ・コンピュータ上の Mobile Link サーバに、Windows Mobile デバイスを接続できます。

Palm Computing Platform の場合は、localhost のデフォルト値がデバイスを指します。デスクトップ・コンピュータに接続するには、明示的なホスト名または IP アドレスを指定してください。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」 『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の例では、クライアントはコンピュータ myhost のポート 1234 に接続します。

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e "adr='host=myhost;port=1234'"
```

Embedded SQL または C++ で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
synch_info.stream_parms = TEXT("host=myhost;port=1234");
```


http_password

RFC 2617 の基本認証またはダイジェスト認証を使用してサード・パーティの HTTP サーバとゲートウェイに対する認証を行います。

構文

http_password=*password*

プロトコル

- HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

なし

備考

この機能は、RFC 2617 に記述されている基本認証とダイジェスト認証をサポートします。

基本認証ではパスワードはクリア・テキストで HTTP ヘッダに含められますが、HTTPS を使用すると、ヘッダを暗号化してパスワードを保護できます。ダイジェスト認証では、ヘッダはクリア・テキストでは送信されず、ハッシュされます。

このオプションは、http_userid と併用する必要があります。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「http_userid」 61 ページ
- 「http_proxy_password」 59 ページ
- 「http_proxy_userid」 60 ページ

例

次に示す Embedded SQL または C++ の Ultra Light アプリケーションの例は、Web サーバに対する基本認証のユーザ ID とパスワードを提供します。

```
synch_info.stream = "https";
synch_info.stream_parms = TEXT("http_userid=user;http_password=pwd");
```


http_proxy_password

RFC 2617 の基本認証またはダイジェスト認証を使用してサード・パーティの HTTP プロキシに対する認証を行います。

構文

http_proxy_password=*password*

プロトコル

- HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

なし

備考

この機能は、RFC 2617 に記述されている基本認証とダイジェスト認証をサポートします。

基本認証では、パスワードはクリア・テキストで HTTP ヘッダに含められ、HTTPS を使用できます。ただし、プロキシに対する最初の接続は HTTP を通して行われるため、このパスワードはクリア・テキストです。ダイジェスト認証では、ヘッダはクリア・テキストでは送信されず、ハッシュされます。

このオプションは、http_proxy_userid と併用する必要があります。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「http_password」 58 ページ
- 「http_userid」 61 ページ
- 「http_proxy_userid」 60 ページ

例

次に示す Embedded SQL または C++ の Ultra Light アプリケーションの例は、Web プロキシに対する基本認証のユーザ ID とパスワードを提供します。

```
synch_info.stream = "https";
synch_info.stream_parms = TEXT("http_proxy_userid=user;http_proxy_password=pwd");
```

http_proxy_userid

RFC 2617 の基本認証またはダイジェスト認証を使用してサード・パーティの HTTP プロキシに対する認証を行います。

構文

http_proxy_userid=*userid*

プロトコル

- HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

なし

備考

この機能は、RFC 2617 に記述されている基本認証とダイジェスト認証をサポートします。

基本認証では、パスワードはクリア・テキストで HTTP ヘッダに含められ、HTTPS を使用できます。ただし、プロキシに対する最初の接続は HTTP を通して行われるため、このパスワードはクリア・テキストです。ダイジェスト認証では、ヘッダはクリア・テキストでは送信されず、ハッシュされます。

このオプションは、http_proxy_password と併用する必要があります。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「http_password」 58 ページ
- 「http_userid」 61 ページ
- 「http_proxy_password」 59 ページ

例

次に示す Embedded SQL または C++ の Ultra Light アプリケーションの例は、Web プロキシに対する基本認証のユーザ ID とパスワードを提供します。

```
synch_info.stream = "https";
synch_info.stream_parms = TEXT("http_proxy_userid=user;http_proxy_password=pwd");
```


http_userid

RFC 2617 の基本認証またはダイジェスト認証を使用してサード・パーティの HTTP サーバとゲートウェイに対する認証を行います。

構文

http_userid=*userid*

プロトコル

- HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

なし

備考

この機能は、RFC 2617 に記述されている基本認証とダイジェスト認証をサポートします。

基本認証ではパスワードはクリア・テキストで HTTP ヘッダに含められますが、HTTPS を使用すると、ヘッダを暗号化してパスワードを保護できます。ダイジェスト認証では、ヘッダはクリア・テキストでは送信されず、ハッシュされます。

このオプションは、http_password と併用する必要があります。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「http_password」 58 ページ
- 「http_proxy_password」 59 ページ
- 「http_proxy_userid」 60 ページ

例

次に示す Embedded SQL または C++ の Ultra Light アプリケーションの例は、Web サーバに対する基本認証のユーザ ID とパスワードを提供します。

```
synch_info.stream = "https";
synch_info.stream_parms = TEXT("http_userid=user;http_password=pwd");
```


identity_name

この機能は、Common Access Card (CAC) のクライアント ID (証明書とプライベート・キー) を使用した認証をサポートします。この機能は、Windows Mobile だけでサポートされます。

このパラメータを使用して、パブリック証明書の通称を指定します。

別途ライセンスが必要な必須コンポーネント
この機能は CAC 認証アドオンの一部であり、別途ライセンスが必要です。「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

構文

identity_name=*name*

プロトコル

- TLS、HTTPS

デフォルト

なし

備考

このパラメータは、FIPS 認定 RSA 暗号化のみと組み合わせて使用できます。

パブリック証明書がデバイスの証明書ストアにインストールされている必要があります。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

network_leave_open

network_name を指定すると、オプションとして、同期が終了した後にネットワーク接続を開いたままにするように指定できます。

構文

network_leave_open={ off | on }

プロトコル

- TCPIP、TLS、HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

Palm 以外のデバイスでは、デフォルトは **off** です。

Palm では、デフォルトは **on** です。

備考

このオプションを使用するには、network_name を指定する必要があります。

このオプションを on に設定すると、同期が終了した後もネットワーク接続は開いたままになります。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「network_name」 64 ページ

例

次の例では、クライアントは、ネットワーク名 MyNetwork を使用し、同期の完了後も接続を開いたままにしておくように指定します。

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e "adr='network_name=MyNetwork;network_leave_open=on'"
```

Embedded SQL または C++ で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
synch_info.stream_parms = TEXT("network_name=MyNetwork;network_leave_open=on");
```


network_name

ネットワークに接続しようとして失敗した場合に開始するネットワーク名を指定します。

構文

network_name=*name*

プロトコル

- TCPIP、TLS、HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

なし

備考

ネットワーク名を指定して、Mobile Link の自動ダイヤル機能を使用できるようにします。これによって、手動でダイヤルすることなく Windows Mobile デバイスまたは Windows デスクトップ・コンピュータから接続できます。自動ダイヤルとは、Mobile Link サーバへの接続の 2 回目の試行のことです。クライアントがダイヤルせずに接続しようとして失敗し、network_name を指定すると、自動ダイヤルがアクティブになります。スケジュールと組み合わせて使用すると、リモート・データベースを無人で同期できます。スケジュールと組み合わせなくても、手動でダイヤルして接続せずに dbmlsync を実行できます。

Windows Mobile では、*name* は、[設定] - [接続] - [接続] のドロップダウン・リスト内のネットワーク・プロファイルである必要があります。インターネット・ネットワークまたは勤務先ネットワークのデフォルト設定を使用するには、名前をそれぞれキーワード **default_internet** または **default_work** に設定します。

Windows デスクトップ・プラットフォームでは、*name* は [ネットワークとダイアルアップ接続] 内のネットワーク・プロファイルである必要があります。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「同期のスケジュール」 128 ページ
- 「network_leave_open」 63 ページ

例

次の例では、クライアントは、ネットワーク名 MyNetwork を使用し、同期の完了後も接続を開いたままにしておくように指定します。

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e "adr='network_name=MyNetwork;network_leave_open=on'"
```

Embedded SQL または C++ で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
synch_info.stream_parms = TEXT("network_name=MyNetwork;network_leave_open=on");
```

persistent

同期のすべての HTTP 要求に単一の TCP/IP 接続を使用します。

構文

persistent={ off | on }

プロトコル

- HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

Off

備考

On という値を指定すると、クライアントは同期ですべての HTTP 要求に同じ TCP/IP 接続を使用しようとします。設定を off にすると、通常、中間エージェントとの互換性が高まります。

Palm デバイスを除き、Mobile Link に直接接続している場合は、persistent を on に設定してください。プロキシやリダイレクタなどの中間エージェントを通じて接続している場合は、永続的な接続によって問題が発生することがあります。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

port

Mobile Link サーバのソケット・ポート番号を指定します。

構文

port=*port-number*

プロトコル

- TCPIP、TLS、HTTP、HTTPS

デフォルト

TCP/IP のデフォルト値は **2439** です。これは、Mobile Link サーバの IANA 登録ポート番号です。

HTTP のデフォルト値は **80** です。

HTTPS のデフォルト値は **443** です。

備考

ポート番号は 10 進数で、Mobile Link サーバが受信するように設定されているポートと一致させます。

Web サーバを介して同期する場合は、HTTP 要求または HTTPS 要求を受け付ける Web サーバ・ポートを指定してください。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の例では、クライアントはコンピュータ myhost のポート 1234 に接続します。

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e "adr='host=myhost;port=1234'"
```

Embedded SQL または C++ で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
synch_info.stream_parms = TEXT("host=myhost;port=1234");
```

proxy_host

プロキシ・サーバのホスト名または IP アドレスを指定します。

構文

proxy_host=*proxy-hostname-or-ip*

プロトコル

- HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

なし

備考

HTTP プロキシを通す場合だけ使用します。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の例では、クライアントはコンピュータ myproxyhost 上で実行されているプロキシ・サーバのポート 1234 に接続します。

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e "adr='proxy_host=myproxyhost;proxy_port=1234'"
```

Embedded SQL または C++ で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
synch_info.stream_parms = TEXT("proxy_host=myproxyhost;proxy_port=1234");
```


proxy_port

プロキシ・サーバのポート番号を指定します。

構文

proxy_port=*proxy-port-number*

プロトコル

- HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

なし

備考

HTTP プロキシを通す場合だけ使用します。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の例では、クライアントはコンピュータ myproxyhost 上で実行されているプロキシ・サーバのポート 1234 に接続します。

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e "adr='proxy_host=myproxyhost;proxy_port=1234'"
```

Embedded SQL または C++ で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
synch_info.stream_parms = TEXT("proxy_host=myproxyhost;proxy_port=1234");
```


set_cookie

同期中に使用される HTTP 要求内に設定するカスタム HTTP cookie を指定します。

構文

set_cookie=*cookie-name***=***cookie-value*,...

プロトコル

- HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

なし

備考

カスタム HTTP cookie は、同期クライアントがサード・パーティ・ツール (セッションを識別するために cookie を使用する認証ツールなど) と対話をする場合に便利です。たとえば、ユーザ・エージェントが Web サーバ、プロキシ、またはゲートウェイに接続し、それ自体の認証を行うシステムが存在するとします。正常に動作する場合、エージェントはサーバから 1 つ以上の cookie を受け取ります。続いてエージェントは同期を開始し、set_cookie オプションを通してそのセッション cookie を渡します。

複数の名前と値の組み合わせがある場合は、カンマで区切ります。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」 『Ultra Light データベース管理とリファレンス』を参照してください。

例

Embedded SQL または C++ の Ultra Light アプリケーションでカスタム HTTP cookie を設定する例を示します。

```
info.stream = "http";
info.stream_parms = TEXT(
  "host=www.myhost.com;
  set_cookie=MySessionID=12345, enabled=yes;");
```


timeout

クライアントがネットワーク操作が失敗したと判断するまで待機する時間 (秒単位) を指定します。

構文

timeout=*seconds*

プロトコル

- TCPIP、TLS、HTTP、HTTPS

デフォルト

240 秒

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

備考

指定した時間内で接続、読み取り、書き込みの試行が完了しなかった場合、クライアントは同期に失敗します。

同期全体を通して、クライアントは指定された間隔で活性更新を送信して、クライアントが接続されていることを Mobile Link サーバに通知します。また、Mobile Link は活性更新を送り返して、サーバが接続されていることをクライアントに通知します。低速ネットワークでタイムアウトが指定時間よりも遅れることを防ぐために、Mobile Link クライアントは、タイムアウト値の半分が経過するたびに Mobile Link サーバにキープアライブ・バイトを送信します。この値を 240 秒に設定すると、キープアライブ・メッセージは 120 秒ごとに送信されます。

設定するタイムアウトの値が低くなりすぎないように注意します。活性チェックを行うと、接続がアクティブであることを確認するために、Mobile Link サーバとクライアントが各タイムアウト時間内に通信する必要があるため、ネットワーク・トラフィックが増加します。ネットワークまたはサーバの負荷が非常に大きく、タイムアウト時間が非常に短い場合は、Mobile Link サーバと dbmlsync が、接続がアクティブであることを確認できないため、活性接続が中止されることがあります。活性タイムアウトは、通常 30 秒 以上に設定します。

タイムアウトの最大値は 10 分です。600 秒を超える数を指定することはできますが、600 秒と解釈されます。

値 0 は、タイムアウトが 10 分であることを意味します。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の例は、タイムアウトを 300 秒に設定します。

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e "adr=timeout=300"
```

Embedded SQL または C++ で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
synch_info.stream_parms = TEXT("timeout=300");
```

tls_type

同期に使用する暗号を解く鍵を指定します。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

構文

tls_type=*cipher*

プロトコル

- TLS、HTTPS

デフォルト

RSA

備考

この同期のための通信はすべて、指定された暗号を使用して暗号化されます。暗号は次のいずれかを指定してください。

- **ecc** 楕円曲線暗号化です。
- **rsa** RSA 暗号化です。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

参照

- 「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」 『SQL Anywhere サーバ - データベース管理』
- 「fips」 55 ページ
- 「Mobile Link クライアント／サーバ通信の暗号化」 『SQL Anywhere サーバ - データベース管理』
- 「-x オプション」 『Mobile Link - サーバ管理』
- 「certificate_company」 45 ページ
- 「certificate_name」 47 ページ
- 「certificate_unit」 49 ページ
- 「trusted_certificates」 75 ページ

例

TCP/IP プロトコルの RSA 暗号化を設定する例を示します。これは、サーバとクライアント両方での設定が必要です。各コマンドは、1 行に入力する必要があります。

サーバでは、実装は次のようになります。

```
mlsrv11
  -c "dsn=SQL Anywhere 11 Demo;uid=DBA;pwd=sql"
  -x tls(
    port=9999;
    tls_type=rsa;
    identity=c:¥sa10¥bin32¥rsaserver.id;
    identity_password=test )
```

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e
   "CommunicationType=tls;
    CommunicationAddress=
      'tls_type=rsa;
       trusted_certificates=¥rsaroot.crt;
       certificate_name=RSA Server'"
```

C または C++ の Embedded SQL で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
      info.stream = "tls";
      info.stream_parms = TEXT(
       "tls_type=rsa;
        trusted_certificates=¥rsaroot.crt;
        certificate_name=RSA Server");
```

trusted_certificates

安全な同期に使用される信頼できるルート証明書のリストを含むファイルを指定します。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

構文

trusted_certificates=*filename*

構文 2 (Palm OS)

trusted_certificates=vfs:[*volume-label*:| *volume-ordinal*:]*filename*

プロトコル

- TLS、HTTPS

デフォルト

なし

備考

Certicom TLS 同期ストリームを介して同期が発生すると、Mobile Link サーバは、その証明書、その証明書に署名したエンティティの証明書など、自己署名ルート証明書に至るまでの証明書をクライアントに送信します。

クライアントは、連鎖が有効で連鎖内のルート証明書が信頼できることを確認します。この機能を使用すると、信頼できるルート証明書を指定できます。

Ultra Light クライアントでは、データベースを作成するときに、信頼できるルートを ulinit、ulcreate、ulload に指定できます。trusted_certificates パラメータを指定すると、ファイル内にある信頼できる証明書が、データベース内に保存されている信頼できる証明書に置き換わります。

32 ビットの Windows と Windows Mobile では、信頼できる証明書が指定されないと、クライアントは、オペレーティング・システムの信頼できる証明書ストアから証明書をロードします。この証明書ストアは、Web サーバを安全に管理するために HTTPS を介して接続するときに、Web ブラウザで使用されます。

信頼できる証明書は、Palm OS ファイル・システムでサポートされています (レコードベースのデータ・ストアではサポートされていません)。Palm OS では、*volume_label* には、**INTERNAL** (組み込みのドライブ)、**CARD** (拡張カード)、またはボリュームのラベル名を指定できます。また、*volume-ordinal* を使用して、ボリュームを識別することもできます (デフォルトは 0 で、プラットフォームで列挙されている最初のボリュームです)。Palm プラットフォームのファイル名とパスの命名規則に従って、*filename* には、ファイルへのフル・パスを指定してください。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「Ultra Light 接続パラメータでのファイル・パスの指定」『Ultra Light データベース管理とリファレンス』
- 「Mobile Link クライアント／サーバ通信の暗号化」『SQL Anywhere サーバ - データベース管理』
- 「-x オプション」『Mobile Link - サーバ管理』
- 「tls_type」 73 ページ
- 「certificate_company」 45 ページ
- 「certificate_name」 47 ページ
- 「certificate_unit」 49 ページ

例

HTTPS プロトコルの RSA 暗号化を設定する例を示します。これは、サーバとクライアント両方での設定が必要です。各コマンドは、1 行に入力する必要があります。

サーバでは、実装は次のようになります。

```
mlsrv11
  -c "dsn=SQL Anywhere 11 Demo;uid=DBA;pwd=sql"
  -x https(
    port=9999;
    identity=c:¥sa10¥bin32¥rsaserver.id;
    identity_password=test)
```

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync
  -c "dsn=mydb;uid=DBA;pwd=sql"
  -e "ctp=https;
      adr='port=9999;
        trusted_certificates=c:¥sa10¥bin32¥rsaroot.crt;
        certificate_name=RSA Server'"
```

Ultra Light クライアントでは、実装は次のようになります。

```
    info.stream = "https";
    info.stream_parms = TEXT(
      "port=9999;
       trusted_certificates=¥rsaroot.crt;
       certificate_name=RSA Server");
```

Palm OS が動作している Ultra Light クライアントの場合、stream と stream_parms の設定は次のようになります。

```
info.stream = "https";
info.stream_parms = "trusted_certificates=vfs:/rsaroot.crt;port=9999";
```


url_suffix

同期中に送信される各 HTTP 要求の 1 行目の URL に追加するサフィックスを指定します。

構文

url_suffix=*suffix*

suffix の構文は、使用するリダイレクタのタイプによって異なります。

リダイレクタ	*suffix* の構文
ISAPI	exe_dir/iaredirect.dll/ml/[server-group/] *exe-dir* は *iaredirect.dll* のロケーションです。
NSAPI	mlredirect/ml/[server-group/] *mlredirect* は *obj.conf* ファイル内でマッピングされている名前です。
Apache	*httpd.conf* ファイルでリダイレクタの <location> タグに指定した内容
Servlet	iaredirect/ml/
M-Business Anywhere	*sync.conf* ファイルでリダイレクタの <location> タグに指定した内容

プロトコル

- HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できませんが、クライアントでリダイレクタ用に設定されます。

デフォルト

デフォルトは **MobiLink** です。

備考

プロキシ・サーバまたは Web サーバを介して同期する場合、Mobile Link サーバを検索するために url_suffix が必要な場合があります。

サーバ・グループは、一部のリダイレクタでのみサポートされます。詳細については、http://www.ianywhere.jp/developers/technotes/redirector_web_servers_1101.html を参照してください。

リダイレクタを使用する場合にこのオプションを設定する方法については、「Mobile Link クライアントとサーバのリダイレクタ設定」 『Mobile Link - サーバ管理』を参照してください。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「Mobile Link クライアントとサーバのリダイレクタ設定」『Mobile Link - サーバ管理』
- 「Mobile Link サーバ・グループ」『Mobile Link - サーバ管理』

例

次の SQL 文は、HTTPS での同期に使用される、sales5322 という同期ユーザを作成します。企業のファイアウォールの外側で Mobile Link サーバを実行し、リダイレクタ (NSAPI Web サーバのリバース・プロキシ) を使用して同期要求をリダイレクトすることが前提です。Mobile Link ユーザは、*https://www.mycompany.com:80/mlredirect/ml/* という URL に同期します。

```
CREATE SYNCHRONIZATION USER sales5322
TYPE https
ADDRESS 'host=www.mycompany.com;port=80;url_suffix=mlredirect/ml/'
```

例

次の例は、Ultra Light クライアント用の、ISAPI リダイレクタを通じた HTTP を設定します。コマンド・ラインは、すべて 1 行で入力してください。

```
mlsrv11 -c "dsn=SQL Anywhere 11 CustDB"
  -dl -fr -ot mlserver.mls -zu+ -v+
  -x http(port=8081)
```

ulsync を使用して同期するには、次のようなコマンドを実行します。

```
ulsync -c "dbf=custdb.udb"
  -v "MobiLinkUid=50;ScriptVersion=custdb 11.0;
   Stream=http(port=80;url_suffix=Scripts/iaredirect.dll/ml/)"
```


version

同期に使用する HTTP のバージョンを指定します。

構文

version=*HTTP-version-number*

プロトコル

- HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。

デフォルト

デフォルト値は **1.1** です。

備考

このオプションは、HTTP インフラが特定の HTTP バージョンを必要とする場合に便利です。値は、**1.0** または **1.1** を指定します。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

例

次の例は、HTTP バージョンを 1.0 に設定します。

SQL Anywhere クライアントでは、実装は次のようになります。

```
dbmlsync -e "adr=version=1.0"
```

Embedded SQL または C++ で作成された Ultra Light アプリケーションでは、実装は次のようになります。

```
synch_info.stream_parms = TEXT("version=1.0");
```

zlib_download_window_size

compression オプションを zlib に設定した場合は、このオプションを使用して、ダウンロードの圧縮ウィンドウ・サイズを指定します。

構文

zlib_download_window_size=*window-bits*

プロトコル

- TCPIP、TLS、HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。
- Palm OS または Symbian ではサポートされません。

デフォルト

Windows Mobile では 12、それ以外では 15 です。

備考

ダウンロード圧縮をオフにするには、*window-bits* を 0 に設定します。それ以外の場合は、ウィンドウ・サイズは 9 ～ 15 になります。通常、ウィンドウ・サイズを大きくすると圧縮率を高くすることができますが、必要なメモリが大きくなります。

window-bits は、ウィンドウ・サイズを底が 2 の対数 (履歴バッファのサイズ) で指定します。次の式は、各 *window-bits* に対して、クライアントで使用されるメモリ量の算出に使用します。

upload (compress): memory = $2^{(window\text{-}bits + 3)}$

download (decompress): memory = $2^{(window\text{-}bits)}$

Ultra Light で zlib compression をサポートするには、アプリケーションで ULEnableZlibSyncCompression(sqlca) を呼び出し、*mlczlib10.dll* を配備する必要があります。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「compression」 51 ページ
- 「zlib_upload_window_size」 81 ページ

例

次のオプションでは、アップロードの圧縮のみを設定できます。

```
"compression=zlib;zlib_download_window_size=0"
```


zlib_upload_window_size

compression オプションを zlib に設定した場合は、このオプションを使用して、アップロードの圧縮ウィンドウ・サイズを指定します。

構文

zlib_upload_window_size=*window-bits*

プロトコル

- TCPIP、TLS、HTTP、HTTPS

サポートに関する注意

- リダイレクタの ML ディレクティブでは使用できません。
- Palm OS または Symbian ではサポートされません。

デフォルト

Windows Mobile では 12、それ以外では 15 です。

備考

アップロード圧縮をオフにするには、ウィンドウ・サイズを 0 に設定します。それ以外の場合は、ウィンドウ・サイズは 9 ～ 15 になります。通常、ウィンドウ・サイズを大きくすると圧縮率を高くすることができますが、必要なメモリが大きくなります。

window-bits は、ウィンドウ・サイズを底が 2 の対数 (履歴バッファのサイズ) で指定します。次の式は、各 *window-bits* に対して、クライアントで使用されるメモリ量の算出に使用します。

upload (compress): memory = $2^{(window\text{-}size + 3)}$

download (decompress): memory = $2^{(window\text{-}size)}$

Ultra Light で zlib compression をサポートするには、アプリケーションで ULEnableZlibSyncCompression(sqlca) を呼び出し、*mlczlib10.dll* を配備する必要があります。

dbmlsync を使用してネットワーク・プロトコル・オプションを設定する方法については、「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

Ultra Light を使用してネットワーク・プロトコル・オプションを設定する方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「compression」 51 ページ
- 「zlib_download_window_size」 80 ページ

例

次のオプションでは、ダウンロードの圧縮のみを設定できます。

```
"compression=zlib;zlib_upload_window_size=0"
```

リモート・クライアントでのスキーマの変更

目次

Mobile Link クライアントでのスキーマの変更の概要

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。

警告
スキーマを変更する直前に、正常な同期を実行します。スキーマをアップグレードするときに、開いているトランザクションがないことを確認してください。

SQL Anywhere リモート・データベースのスキーマのアップグレード

SQL 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。

> リモート・データベースに他の接続がないことが確実である場合は、ALTER PUBLICATION 文を手動で使用して、新規または変更したテーブルをパブリケーションに追加できます。それ以外の場合は、sp_hook_dbmlsync_schema_upgrade フックを使用して、スキーマをアップグレードしてください。
>
> 「sp_hook_dbmlsync_schema_upgrade」 302 ページを参照してください。

♦ **SQL 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。同期処理が正常に実行されたことを確認してから、次の処理を続行してください。
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;
   ```

   この文は、sp_hook_dbmlsync_schema_upgrade フックの内部で使用できます。
   「sp_hook_dbmlsync_schema_upgrade」 302 ページを参照してください。

   詳細については、「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』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 SQL 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」 『Mobile Link - サーバ管理』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。同期処理が正常に実行されたことを確認してから、次の処理を続行してください。
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、「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   この文は、sp_hook_dbmlsync_schema_upgrade フックの内部で使用できます。「sp_hook_dbmlsync_schema_upgrade」 302 ページ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、「ALTER TABLE 文」 『SQL Anywhere サーバ - SQL リファレンス』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、「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   この文は、sp_hook_dbmlsync_schema_upgrade フックの内部で使用できます。「sp_hook_dbmlsync_schema_upgrade」 302 ページ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。

Ultra Light リモート・データベースのスキーマのアップグレード

既存のアプリケーションで DDL を実行することで、Ultra Light リモート・データベースのスキーマを変更できます。

- 新しいデータベースで新しいアプリケーションを配備する場合は、Mobile Link 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。
- データベースをアップグレードする DDL を含む新しいアプリケーションを配備する場合、データは保持されます。
- 既存のアプリケーションが、一般的な方法で DDL 文を受信できる場合は、DDL をデータベースに適用することができ、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、現実的ではありません。したがって、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 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 DDL の詳細については、「Ultra Light SQL 文」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「Ultra Light スキーマのアップグレードの配備」『Ultra Light データベース管理とリファレンス』

SQL パススルー

目次

SQL パススルーの概要

SQL パススルー機能によって、統合データベースから SQL Anywhere または Ultra Light のクライアントに SQL 文のスクリプトをダウンロードし、クライアント上で適切な時間に SQL 文を実行させることができます。スクリプトは番号が付けられ、クライアントで順序どおりに実行されることが保証されます。

スクリプトの実行後、またはスクリプトの実行が試みられた後、次回の同期を使用して統合データベースにステータスが送り返されます。これにより、実行の成功と失敗の両方を一元的にモニタできます。クライアントでのスクリプト実行中にエラーが発生した場合、統合データベースにステータスがアップロードされ、さらに、統合データベースが処理の続行方法に関する命令をクライアントにダウンロードするまで、そのクライアントではそれよりも後のスクリプトは実行されません。これらの命令は、既存のスクリプトをリトライすること、既存のスクリプトをスキップすること、エラーを解決するために新しいスクリプトをダウンロードすることのいずれかを意味する場合があります。

SQL パススルーを使用するのに必要な手順の概要を次に示します。

- スクリプトを作成し、統合データベースに格納します。「スクリプトの作成」 90 ページと「ml_add_passthrough_script システム・プロシージャ」『Mobile Link - サーバ管理』を参照してください。
- スクリプトを実行するクライアントを特定する 1 つ以上のパススルー・エントリを、スクリプトに対して作成します。「パススルー・エントリの作成」 91 ページと「ml_add_passthrough システム・プロシージャ」『Mobile Link - サーバ管理』を参照してください。
- スクリプトをダウンロードします。「スクリプトのダウンロード」 91 ページと「ml_add_passthrough システム・プロシージャ」『Mobile Link - サーバ管理』を参照してください。
- スクリプトを実行します。「スクリプトの実行」 91 ページを参照してください。
- 結果を取得します。「スクリプトの結果の取得」 95 ページを参照してください。
- 結果を確認します。「スクリプトの結果の確認」 95 ページを参照してください。
- 必要に応じて、エラーを処理します。「スクリプトのエラーの処理」 95 ページを参照してください。

スクリプトの作成

ml_add_passthrough_script ストアド・プロシージャを使用して、スクリプトを作成し、統合データベースに格納します。「ml_add_passthrough_script システム・プロシージャ」『Mobile Link - サーバ管理』を参照してください。

パススルー・エントリの作成

ml_add_passthrough システム・プロシージャを使用して、統合データベースに作成されたパススルー・スクリプトを受け取る Mobile Link クライアントを指定します。「ml_add_passthrough システム・プロシージャ」 『Mobile Link - サーバ管理』を参照してください。

スクリプトのダウンロード

スクリプトは、同期中に統合データベースから Mobile Link クライアントに自動的にダウンロードされます。次のような状況では、スクリプトはダウンロード「されません」。

- ファイルベースのダウンロードが実行されたとき。
- ping が実行されたとき。
- ダウンロードが再起動されたとき。

SQL Anywhere クライアントでの SQL パススルー・スクリプトの格納

統合データベースから SQL Anywhere クライアントにダウンロードされた SQL パススルー・スクリプトは、dbo.sync_passthrough_script テーブルに格納されます。

Ultra Light クライアントでの SQL パススルー・スクリプトの格納

統合データベースから Ultra Light クライアントにダウンロードされた SQL パススルー・スクリプトは、syssql テーブルに格納されます。

スクリプトの実行

スクリプトは、Mobile Link クライアント・データベースに格納されると、自動的に実行することも、手動で実行することもできます。スクリプトは、実行方法にかかわらず、run_order に従って順序どおりに実行されます。

SQL Anywhere クライアントでは、スクリプトは常に DBO アカウントの下で DBA 権限を使用して実行されます。

SQL Anywhere クライアントでスクリプトを手動で実行

手動で任意のスクリプトを実行できます。SQL Anywhere クライアントでスクリプトを手動で実行するには、sync_get_next_passthrough_script 関数と sync_execute_next_passthrough_script 関数を使用します。

sync_get_next_passthrough_script 関数はパラメータを取らず、次に実行するスクリプトの run_order を返します。そのスクリプトに関する情報を取得するには、dbo.sync_passthrough_script テーブルに問い合わせてください。

dbo.sync_passthrough_script テーブル

dbo.sync_passthrough_script テーブルは、次のように定義されています。

カラム名	説明
run_order	INTEGER。run_order パラメータによって、スクリプトをリモート・データベースに適用する順序が決まります。スクリプトは常に、run_order に従って順序どおりに適用されます。 この値は負でない整数にします。
script_id	INTEGER。この値はスクリプトをユニークに識別します。
script_name	VARCHAR(128)。スクリプトの名前。このカラムは、統合データベースで ml_add_passthrough_script を呼び出したときにスクリプトに対して指定した script_name の値と対応します。
flags	BIGINT。flags カラムは、ml_add_passthrough_script ストアド・プロシージャに渡される flags パラメータで指定された情報を格納します。指定されたフラグは、整数にコード化されます。コード化は、指定された各フラグを次に示す値に変換し、その値を OR 演算子で結合することによって行われます。 ● **manual**　スクリプトが手動実行モードでのみ実行できることを示します。デフォルトでは、すべてのスクリプトは自動実行モードと手動実行モードのどちらでも実行できます。 ● **排他**　スクリプトがすべての同期対象テーブルに対する排他ロックが取得された、同期の最後に自動的に実行できることを示します。affected_pubs 値にパブリケーションが 1 つもリストされていない場合、このオプションは無視されます。このオプションは、SQL Anywhere リモートでのみ有効です。 ● **schema_diff**　スクリプトがスキーマ diff モードでのみ実行する必要があることを示します。このモードでは、スクリプト内に記述されたスキーマと一致するようにデータベース・スキーマが変更されます。たとえば、既存のテーブルに対する create 文は alter 文として扱われます。このフラグは、Ultra Light リモートで実行されるスクリプトにのみ適用されます。
affected_pubs	LONG VARCHAR。スクリプトの実行前に同期する必要があるパブリケーションのリスト。このカラムは、ml_add_passthrough_script を呼び出したときにスクリプトに対して指定した affected_pubs の値と対応します。
script	LONG VARCHAR。パススルー・スクリプトの内容。このカラムは、ml_add_passthrough_script を呼び出したときにスクリプトに対して指定した script の値と対応します。

カラム名	説明
description	VARCHAR(2000)。スクリプトのコメントまたは説明。このカラムは、ml_add_passthrough_script を呼び出したときにスクリプトに対して指定した description の値と対応します。

実行するスクリプトがもうない場合や、最後に実行したスクリプトでエラーが発生しても処理の続行方法に関する命令をサーバからまだ受け取っていない場合、sync_get_next_passthrough_script 関数は NULL を返します。

sync_execute_next_passthrough_script 関数はパラメータを取りません。この関数は、次のスクリプトを実行し、スクリプトの結果を後で統合データベースにアップロードできるように、データベース内の進行状況情報とステータス情報を更新します。最後のスクリプトがエラーを返し、エラーの処理方法に関する命令が Mobile Link サーバからまだ届いていない場合、スクリプトは実行されません。スクリプトが実行された場合、そのスクリプトの実行順序が返されます。スクリプトがまったく実行されなかった場合、NULL が返されます。「スクリプトの結果の取得」 95 ページを参照してください。

Ultra Light クライアントでスクリプトを手動で実行

Ultra Light クライアントでは、スクリプトを手動で適用するためにいくつかの ESQL API メソッドを使用できます。メソッドは次のとおりです。

- 「GetSQLPassthroughScriptCount メソッド」 『Ultra Light - .NET プログラミング』
- 「ExecuteSQLPassthroughScripts メソッド」 『Ultra Light - .NET プログラミング』
- 「ExecuteNextSQLPassthroughScript メソッド」 『Ultra Light - .NET プログラミング』

C++、Ultra Light.NET、M-Business Anywhere の各インタフェースに同等のメソッドがあります。

SQL Anywhere クライアントでスクリプトを自動的に実行

SQL Anywhere クライアントでは、待機しているスクリプトがある場合、それぞれの同期の最後にそのようなスクリプトの実行が試みられます。実行可能なスクリプトは run_order の順序に従って、次のいずれかの状況が発生するまで 1 つずつ実行されます。

- すべてのスクリプトが実行された。
- スクリプトが失敗した。
- 自動的に実行できないスクリプトの番になった。

次のいずれかに当てはまる場合、スクリプトを自動的に実行することはできません。

- スクリプトの作成時に **manual** フラグが指定された。

- スクリプトの **affected publications** 値が空でなく、さらに次のいずれかの条件が当てはまる。
  - アップロードが実行されなかった。
  - アップロードが失敗した。
  - **affected publications** 値にリストされているパブリケーションのうち 1 つ以上が同期されなかった。
  - スクリプトの作成時に **exclusive** フラグが指定され、同期の開始時にすべての同期対象テーブルに対する排他ロックは取得されなかった。

注意
ダウンロード専用のパブリケーションは、影響を受けるパブリケーションとしてリストしないでください。

dbmlsync では、同期の最初に、LockTables 拡張オプションを使用して、要求されているロックよりも制限の厳しいロックを同期対象テーブルに対して取得する可能性があります。これは、同期の最後にスクリプトを確実に実行できるようにするためです。たとえば、LockTables が SHARE に設定されているにもかかわらず、実行可能な次のスクリプトが排他ロックを要する場合、排他ロックが取得されることがあります。

Ultra Light クライアントでスクリプトを自動的に実行

Ultra Light クライアントでは、**manual** とマーク付けされていないスクリプトは、データベースの次回起動時に自動的に実行されます。ただし、接続パラメータ **dont_run_scripts** が設定されている場合は実行されません。

進行状況 observer コールバックが指定されている場合、手動実行と自動実行のどちらのときも、スクリプト実行の進行状況を提供できます。進行状況 observer コールバックは、次のように定義されます。

```
typedef void(UL_CALLBACK_FN *ul_sql_passthrough_observer_fn)
    ( ul_sql_passthrough_status * status );
```

ul_sql_passthrough_status 構造体は、次のように定義されます。

```
typedef struct {
    ul_sql_passthrough_state    state; // current state
    ul_u_long  script_count; // total number of scripts to execute
    ul_u_long  cur_script; // current script being executed (1-based)
    ul_bool  stop;  // set to true to stop script execution
        // can only be set in the starting state
    ul_void *  user_data; // user data provided in register call
    SQLCA *  sqlca;
} ul_sql_passthrough_status;
```

進行状況 observer コールバックは、次のメソッドによって登録されます。

```
UL_FN_SPEC ul_ret_void UL_FN_MOD ULRegisterSQLPassthroughCallback(
SQLCA *                         sqlca,
ul_sql_passthrough_observer_fn      callback,
ul_void *                       user_data );
```

スクリプトの結果の取得

スクリプトを手動で実行したか、自動的に実行したかに関係なく、Mobile Link クライアントにスクリプト実行の結果が保存されます。

SQL Anywhere クライアントでは、結果は dbo.sync_passthrough_status テーブルに格納されます。結果は、リモート・データベースでのスクリプト実行時刻と、スクリプトが正常に実行されたかエラーがレポートされたかを示す情報で構成されます。エラーがレポートされた場合、これらに加えて SQL コードと、エラー・メッセージのテキストも格納されます。

Ultra Light クライアントでは、結果は syssql テーブルに格納されます。結果は、リモート・データベースでのスクリプト実行時刻と、スクリプトが正常に実行されたかエラーがレポートされたかを示す情報で構成されます。エラーがレポートされた場合、これらに加えて SQL コード、失敗した文のスクリプト内での行番号、エラー・パラメータのリストも格納されます。

スクリプトの結果の確認

Ultra Light と SQL Anywhere のどちらのクライアントも、それぞれの同期の一部として実行したスクリプトの結果をアップロードします。Mobile Link サーバは、アップロードされた結果を統合データベースの ml_passthrough_status テーブルに格納します。このテーブルを調べることによって、クライアントに配布されたパススルー・スクリプトが成功したかどうかを確認できます。「ml_passthrough_status」 『Mobile Link - サーバ管理』を参照してください。

スクリプトのエラーの処理

警告
SQL パススルー機能は非常に強力なので、使用時には「注意」が必要です。特に重要なのは、スクリプトを十分にテストすることです。SQL パススルー・スクリプトでエラーが発生すると、すべてのリモートが使用できなくなったり、損傷したりする可能性があるためです。入念なテストを行って、エラーを回避してください。

クライアント上の SQL パススルー・スクリプトでエラーが生成された場合、統合データベースでエラーを解決する必要があります。解決しないと、クライアントはそれよりも後の SQL パススルー・スクリプトを実行できません。

mlsrv11 用の -vo サーバ・オプションを使用して、Mobile Link サーバ上で SQL パススルー・アクティビティを取得すると、エラーの解決に役立つことがあります。「-v オプション」 『Mobile Link - サーバ管理』を参照してください。

クライアントでのエラー解決の基本的メカニズムは、ml_add_passthrough_repair システム・プロシージャを使用して ml_passthrough_repair テーブルにローを追加することです。ml_passthrough_repair テーブル内のローには、特定のスクリプトが特定のエラー・コードを生成したときにクライアントが実行する必要のあるアクションが記述されます。

参照

- 「ml_passthrough_repair」 『Mobile Link - サーバ管理』
- 「ml_add_passthrough_repair システム・プロシージャ」 『Mobile Link - サーバ管理』
- 「ml_delete_passthrough_repair システム・プロシージャ」 『Mobile Link - サーバ管理』

Mobile Link 用 SQL Anywhere クライアント

この項では、Mobile Link 同期のために SQL Anywhere クライアントを設定し実行する方法について説明します。

SQL Anywhere クライアント

目次

リモート・データベースの作成

SQL Anywhere データベースは、Mobile Link システムでリモート・データベースとして使用できます。必要な作業は、パブリケーションを作成し、Mobile Link ユーザを作成して統合データベースに登録し、Mobile Link ユーザをパブリケーションにサブスクライブすることだけです。

同期モデル作成ウィザードを使用して Mobile Link クライアント・アプリケーションを作成した場合、これらのオブジェクトはモデルの配備時に自動的に作成されます。この場合も、概念を理解している必要があります。

♦ SQL Anywhere データベースをリモート・データベースとして使用するには、次の手順に従います。

1. 既存の SQL Anywhere データベースを指定して起動するか、新しいデータベースを作成してテーブルを追加します。
2. リモート・データベース内で 1 つまたは複数のパブリケーションを作成します。

   「データのパブリッシュ」 105 ページを参照してください。
3. リモート・データベースに Mobile Link ユーザを作成します。

   「Mobile Link ユーザの作成」 113 ページを参照してください。
4. ユーザを統合データベースに登録します。

   「統合データベースへの Mobile Link ユーザ名の追加」 11 ページを参照してください。
5. 1 つまたは複数のパブリケーションに対する Mobile Link ユーザのサブスクリプションを作成します。

   「同期サブスクリプションの作成」 116 ページを参照してください。

リモート・データベースの配備

SQL Anywhere リモート・データベースを配備するには、データベースを作成し、適切なパブリケーションとサブスクリプションを追加する必要があります。これを行うには、プロトタイプのリモート・データベースをカスタマイズします。

スターター・データベースを複数のロケーションに配備する場合は、リモート ID に NULL が設定されているデータベースを配備するのが最も安全です。事前に移植するようにデータベースを同期した場合は、配備前にリモート ID を NULL に設定し直すことができます。この方法によって、リモート・データベースが初めて同期したときにユニークなリモート ID が割り当てられるため、リモート ID をユニークにすることができます。また、リモート ID はリモート・セットアップ手順として設定できますが、ユニークでなければなりません。

「リモート ID の設定」 102 ページを参照してください。

♦ **プロトタイプをカスタマイズして Mobile Link リモート・データベースを配備するには、次の手順に従います。**

1. プロトタイプとなるリモート・データベースを作成します。

   プロトタイプ・データベースには、必要なテーブルとパブリケーションをすべて入れますが、各データベースに固有の情報を入れる必要はありません。通常、この情報は次のとおりです。

   - Mobile Link ユーザ名
   - 同期サブスクリプション
   - グローバル・オートインクリメント・キー値の始点を提供する global_database_id オプション

2. リモート・データベースごとに、次の操作を実行します。

   - リモート・データベースを保持するディレクトリを作成します。
   - そのディレクトリにプロトタイプのリモート・データベースをコピーします。

     トランザクション・ログがリモート・データベースと同じディレクトリに保持されている場合、ログ・ファイル名を変更する必要はありません。

   - 個々の情報をデータベースに追加する SQL スクリプトを実行します。

     この SQL スクリプトは、パラメータ化されたスクリプトにすることができます。パラメータ化されたスクリプトの詳細については、「PARAMETERS 文 [Interactive SQL]」『SQL Anywhere サーバ - SQL リファレンス』と「SQL コマンド・ファイルの使用」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

同期モデル作成ウィザードを使用して Mobile Link クライアント・アプリケーションを作成する場合は、ウィザードを使用してデータベースを配備できます。「モデルの配備」『Mobile Link - クイック・スタート』を参照してください。

参照

- 「SQL Anywhere Mobile Link クライアントの配備」『Mobile Link - サーバ管理』
- 「最初の同期は常に行われる」 103 ページ

例

次の SQL スクリプトは、Contact の例から抜粋したものです。このスクリプトは *samples-dir¥MobiLink¥Contact¥customize.sql* にあります。*samples-dir* の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

```
PARAMETERS ml_userid, db_id;
go
SET OPTION PUBLIC.global_database_id = {db_id}
go

CREATE SYNCHRONIZATION USER {ml_userid}
      TYPE 'TCPIP'
      ADDRESS 'host=localhost;port=2439'
      OPTION MEM=''
go
CREATE SYNCHRONIZATION SUBSCRIPTION TO "DBA"."Product"
```

```
        FOR {ml_userid}
go
CREATE SYNCHRONIZATION SUBSCRIPTION TO "DBA"."Contact"
        FOR {ml_userid}
go
commit work
go
```

次のコマンドは、データ・ソース dsn_remote_1 を指定し、リモート・データベースに対してスクリプトを実行します。

```
dbisql -c "dsn=dsn_remote_1" read customize.sql [SSinger] [2]
```

リモート ID の設定

リモート ID により、Mobile Link 同期システムのリモート・データベースがユニークに識別されます。SQL Anywhere データベースを作成すると、リモート ID は NULL に設定されます。データベースを Mobile Link と同期する場合、Mobile Link は NULL のリモート ID をチェックし、該当するリモート ID が見つかると、GUID をリモート ID として割り当てます。リモート ID を一度設定すると、手動で変更しないかぎり、データベースでは同じリモート ID を保持します。

Mobile Link のイベント・スクリプトや別のものからリモート ID を参照する場合は、リモート ID にわかりやすい名前を付けることができます。リモート ID を変更するには、リモート・データベースの ml_remote_id データベース・オプションを設定します。ml_remote_id オプションはユーザ定義のオプションで、SYSOPTION システム・テーブルに格納されます。このオプションを変更するには、SET OPTION 文や Sybase Central の SQL Anywhere プラグインを使用します。

リモート ID は、同期システム内でユニークでなければなりません。

データベース・オプションの変更の詳細については、次の項を参照してください。

- 「SET OPTION 文」 『SQL Anywhere サーバ - SQL リファレンス』
- 「データベース・オプションの設定」 『SQL Anywhere サーバ - データベース管理』
- 「SYSOPTION システム・ビュー」 『SQL Anywhere サーバ - SQL リファレンス』

警告
リモート ID は最初の同期を実行する前に変更すると安全です。リモート ID を後で変更する場合は、変更する直前に同期処理の実行を完了しておくようにしてください。同期処理を行わないでリモート ID を変更すると、一部のデータが失われ、データベースの整合性が保たれなくなる可能性があります。

参照

- 「リモート ID」 15 ページ

例

次の SQL 文では、リモート ID を HR001 に設定します。

```
SET OPTION PUBLIC.ml_remote_id = 'HR001'
```


リモート・データベースのアップグレード

新しい SQL Anywhere リモート・データベースをインストールして古いバージョンを上書きすると、統合データベース内の同期進行状況情報が正しくなくなります。この問題を解決するには、このユーザの ml_user テーブルの progress カラムを 0 (ゼロ) に設定します。

Mobile Link システム・テーブル ml_user の詳細については、「ml_user」『Mobile Link - サーバ管理』を参照してください。

アップグレードの詳細については、「SQL Anywhere Mobile Link クライアントのアップグレード」『SQL Anywhere 11 - 変更点とアップグレード』を参照してください。

進行オフセット

進行オフセットは、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時点を示す整数値です。dbmlsync ユーティリティは、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.ISYSSYNC 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、オフセットは ml_subscription テーブルの progress カラムに格納されます。

リモートごとに、リモート・データベースと統合データベースが各サブスクリプションに対するオフセットを管理します。Mobile Link ユーザが同期を行うと、その Mobile Link ユーザに関連するすべてのサブスクリプションに対してオフセットが確認されます。これは、その時点でサブスクリプションの同期が取られていない場合でも同様です。このように処理されるのは、複数のパブリケーションに同じデータを含めることができるためです。唯一の例外として、dbmlsync は、アップロードを試みるまでサブスクリプションの進行オフセットをチェックしません。

リモート・データベースのオフセットと統合データベースのオフセットが一致しない場合、デフォルトの動作はリモート・データベースのオフセットを統合データベースの値で更新し、そのオフセットに基づいて新しいアップロードを送信します。ほとんどの場合、このデフォルト動作が適切です。たとえば、統合データベースがバックアップからリストアされ、リモート・トランザクション・ログが変更されていないとき、またはアップロードは成功したが通信エラーによりアップロードの確認が送信されないときに、広く有効です。

進行オフセットが一致しない場合の大部分は、統合進行値を使用することで自動的に解決されます。進行オフセットの問題の修正が必要になることがまれにありますが、この場合は dbmlsync -r オプションを使用できます。

詳細については、「-r オプション」 179 ページを参照してください。

最初の同期は常に行われる

新規に作成したサブスクリプションを最初に同期しようとすると、統合データベースの進行オフセットに対するサブスクリプションの進行オフセットはチェックされません。この機能を使用すると、統合データベースで保持されるステータス情報を削除せずに、リモート・データベースを再作成したり同期したりできます。

リモート・データベース・システム・テーブル SYS.ISYSSYNC 内の **progress** カラムの値が **created** カラムの値と同じであり、**log_sent** カラムの値が NULL の場合、dbmlsync ユーティリティは最初の同期を検出します。

ただし、同じアップロードで複数のサブスクリプションを同期し、そのサブスクリプションのいずれかが初めての同期でない場合、初めて同期されるサブスクリプションも含めて、同期対象のすべてのサブスクリプションに関して進行オフセットがチェックされます。たとえば、2 つのパブリケーション (-n pub1,pub2) に関して dbmlsync -n オプションを指定する場合で、pub1 は以前に同期され、pub2 は同期されていないとします。この場合、両方のサブスクリプションの進行オフセットが統合データベースの値に対してチェックされます。

詳細については、次の項を参照してください。

- 「ISYSSYNC システム・テーブル」『SQL Anywhere サーバ - SQL リファレンス』
- 「ml_subscription」『Mobile Link - サーバ管理』
- 「トランザクション・ログ・ファイル」 120 ページ

データのパブリッシュ

パブリケーションとは、同期されるデータを識別するデータベース・オブジェクトです。パブリケーションは、アップロードされるデータを定義し、ダウンロード先になるテーブルを制限します (ダウンロードについては download_cursor スクリプトで定義されます)。

パブリケーションは、1 つまたは複数のアーティクルから成ります。各アーティクルでは、同期するテーブルのサブセットを指定します。サブセットには、テーブル全体またはテーブルのローやカラムのサブセットを指定できます。パブリケーション内の各アーティクルは、異なるテーブルを参照するようにしてください。

パブリケーションをユーザにリンクするためのサブスクリプションを作成します。

Sybase Central または CREATE PUBLICATION 文を使用して、パブリケーションを作成します。

Sybase Central では、**[パブリケーション]** フォルダにすべてのパブリケーションとアーティクルがあります。

パブリケーションについての注意

- パブリケーションの作成と削除には DBA 権限が必要です。
- 同じテーブルの異なるカラム・サブセットが含まれた 2 つのパブリケーションを作成することはできません。
- パブリケーションはどのカラムが選択されているかは確認しますが、それらが送信される順序は確認しません。カラムは、CREATE TABLE 文で定義された順に常に送信されます。
- 各アーティクルは、それぞれが参照するテーブルのプライマリ・キー内のカラムをすべて含んでいる必要があります。
- アーティクルでは、同期するテーブルのカラムを制限できます。WHERE 句を使用して、ローを制限することもできます。
- パブリケーションにビューとストアド・プロシージャを入れることはできません。
- パブリケーションとサブスクリプションは、Sybase のメッセージベースのレプリケーション・テクノロジである SQL Remote でも使用されます。SQL Remote の場合は、統合データベースとリモート・データベースの両方にパブリケーションとサブスクリプションが必要です。これに対して、Mobile Link では、パブリケーションは SQL Anywhere リモート・データベースにのみ存在します。Mobile Link 統合データベースは、同期スクリプトを使用して設定されます。

参照

- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』

テーブル全体のパブリッシュ

作成できる最も簡単なパブリケーションは、アーティクルのセットで構成されます。各アーティクルには 1 つのテーブルのすべてのローとカラムを含めます。これらのテーブルをあらかじめ用意してください。

♦ 1 つ以上のテーブル全体をパブリッシュするには、次の手順に従います (Sybase Central 管理モードの場合)。

1. SQL Anywhere プラグインを使用して、DBA 権限のあるユーザとしてリモート・データベースに接続します。
2. **[パブリケーション]** フォルダを開きます。
3. **[ファイル]** - **[新規]** - **[パブリケーション]** を選択します。
4. **[新しいパブリケーションの名前を指定してください。]** フィールドに、新しいパブリケーションの名前を入力します。**[次へ]** をクリックします。
5. **[次へ]** をクリックします。
6. **[使用可能なテーブル]** リストでテーブルを選択します。**[追加]** をクリックします。
7. **[完了]** をクリックします。

♦ 1 つまたは複数のテーブル全体をパブリッシュするには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてリモート・データベースに接続します。
2. CREATE PUBLICATION 文を実行して、新しく作成するパブリケーション名とパブリッシュするテーブルを指定します。

   「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

例

次の文は、customer テーブル全体をパブリッシュするパブリケーションを作成します。

```
CREATE PUBLICATION pub_customer (
 TABLE customer
)
```

次の文は、SQL Anywhere のサンプル・データベースから、テーブル・セットの各テーブルのすべてのカラムとローを含むパブリケーションを作成します。

```
CREATE PUBLICATION sales (
 TABLE Customers,
 TABLE Salesorders,
 TABLE SalesOrderItems,
 TABLE Products
)
```


テーブル内の一部のカラムだけをパブリッシュする

Sybase Central では、テーブルのすべてのローと、一部のカラムだけを含むパブリケーションを作成できます。また、CREATE PUBLICATION 文でカラムのリストを指定しても、同様に作成できます。

注意

- 異なるカラムのサブセットを持つ同じテーブルが含まれたパブリケーションを 2 つ作成すると、両方のパブリケーションをサブスクライブするユーザは同期することができません。
- アーティクルには、テーブル内のすべてのプライマリ・キーを含めてください。

♦ テーブル内の一部のカラムだけをパブリッシュするには、次の手順に従います (Sybase Central 管理モードの場合)。

1. SQL Anywhere プラグインを使用して、DBA 権限のあるユーザとしてリモート・データベースに接続します。
2. **[パブリケーション]** フォルダを開きます。
3. **[ファイル]** - **[新規]** - **[パブリケーション]** を選択します。
4. **[新しいパブリケーションの名前を指定してください。]** フィールドに、新しいパブリケーションの名前を入力します。**[次へ]** をクリックします。
5. **[次へ]** をクリックします。
6. **[使用可能なテーブル]** リストでテーブルを選択します。**[追加]** をクリックします。
7. **[次へ]** をクリックします。
8. **[使用可能なカラム]** リストで、使用可能なカラムのリストを展開します。カラムを選択し、**[追加]** をクリックします。
9. **[完了]** をクリックします。

♦ テーブル内の一部のカラムだけをパブリッシュするには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてリモート・データベースに接続します。
2. CREATE PUBLICATION 文を実行して、パブリケーション名とテーブル名を指定します。テーブル名の後ろにあるカッコの中に、パブリッシュするカラムをリストします。

   「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

例

次の文は、customer テーブルのカラムである id、company_name、city のすべてのローをパブリッシュするパブリケーションを作成します。

```
CREATE PUBLICATION pub_customer (
 TABLE customer (id, company_name,
```

```
    city )
)
```

テーブル内の一部のローだけをパブリッシュする

パブリケーション定義で WHERE 句の指定がないと、パブリケーション内で変更されたすべてのローがアップロードされます。パブリケーション内のアーティクルに WHERE 句を追加することで、変更されたローのうち WHERE 句の探索条件に一致するローのみをアップロードするよう制限できます。

WHERE 句内の探索条件では、アーティクルに含まれるカラムだけを参照できます。また、WHERE 句では次のいずれも使用できません。

- サブクエリ
- 変数
- 非決定的関数

これらの条件は強制ではありませんが、従わなかった場合、予期しない結果が発生します。WHERE 句に関連するエラーは、パブリケーションの定義時ではなく、その WHERE 句で参照されたテーブルに対して DML が実行されたときに発生します。

♦ WHERE 句を使用してパブリケーションを作成するには、次の手順に従います (Sybase Central 管理モードの場合)。

1. SQL Anywhere プラグインを使用して、DBA 権限のあるユーザとしてリモート・データベースに接続します。
2. **[パブリケーション]** フォルダを開きます。
3. **[ファイル]** - **[新規]** - **[パブリケーション]** を選択します。
4. **[新しいパブリケーションの名前を指定してください。]** フィールドに、新しいパブリケーションの名前を入力します。**[次へ]** をクリックします。
5. **[次へ]** をクリックします。
6. **[使用可能なテーブル]** リストでテーブルを選択します。**[追加]** をクリックします。
7. **[次へ]** をクリックします。
8. **[次へ]** をクリックします。
9. **[アーティクル]** リストでテーブルを選択し、**[選択したアーティクルには次の WHERE 句があります]** ウィンドウ枠で探索条件を入力します。
10. **[完了]** をクリックします。

♦ WHERE 句を使用してパブリケーションを作成するには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてリモート・データベースに接続します。

2. パブリケーション対象のテーブルと WHERE 条件を含む CREATE PUBLICATION 文を実行します。

   「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

例

次の例は、Employees テーブル全体を含み、SalesOrders テーブルの中でアーカイブ済みとしてマーク付けされていないすべてのローを含むパブリケーションを作成します。

```
CREATE PUBLICATION main_publication (
TABLE Employees,
TABLE SalesOrders
WHERE archived = 'N'
);
```

テーブル内の archived カラムを N 以外の値から N に変更すると、次回の同期中に Mobile Link サーバに delete が送信されます。逆に、archived カラムを N から N 以外の値に変更すると、insert が送信されます。archived カラムに対する更新は、Mobile Link サーバに送信**されません**。

ダウンロード専用のパブリケーション

リモート・データベースへのデータのダウンロードのみを行い、データのアップロードは行わないパブリケーションを作成できます。ダウンロード専用のパブリケーションでは、クライアントのトランザクション・ログを使用しません。

異なるダウンロード専用方法における相違点

(アップロードを行わず) ダウンロードのみを行うよう指定するには、2 つの方法があります。

- **ダウンロード専用の同期**　dbmlsync オプションの -e DownloadOnly または -ds を使用します。
- **ダウンロード専用のパブリケーション**　FOR DOWNLOAD ONLY キーワードを使用してパブリケーションを作成します。

これらの 2 つの方法には大きな違いがあります。

ダウンロード専用の同期	ダウンロード専用のパブリケーション
リモート・データベースで変更されているがアップロードはされていないローをダウンロード処理で変更しようとした場合、ダウンロードは失敗します。	リモート・データベースで変更されているがアップロードはされていないローをダウンロード処理で上書きできます。
アップロードやダウンロードが可能な通常のパブリケーションを使用します。ダウンロード専用の同期処理は、dbmlsync のコマンド・ライン・オプションまたは拡張オプションを使用して指定します。	ダウンロード専用パブリケーションを使用します。パブリケーションに対するすべての同期処理はダウンロード専用です。通常のパブリケーションをダウンロード専用に変更することはできません。

ダウンロード専用の同期	ダウンロード専用のパブリケーション
ログ・ファイルが必要です。	ログ・ファイルは必要ありません。
サブスクリプションが長期間アップロードされない場合、ログ・ファイルはトランケートされず、大量のディスク領域が使用される場合があります。	ログ・ファイルが存在すると、同期処理によって同期トランケーション・ポイントは影響されません。このため、パブリケーションが長期間にわたって同期されなくても、ログ・ファイルはトランケートされます。ダウンロード専用のパブリケーションは、ログ・ファイルのトランケーションに影響しません。
ダウンロード専用同期によってスキャンされるログの量を減らすために、時々アップロードを行う必要があります。そうしないと、ダウンロード専用同期が完了するのに次第に時間がかかるようになります。	アップロードを行う必要はありません。

参照

- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「アップロード専用の同期とダウンロード専用の同期」『Mobile Link - サーバ管理』

既存のパブリケーションの変更

パブリケーションを作成してから、アーティクルの追加、修正、削除などの変更を加えたり、パブリケーションの名前を変更したりできます。アーティクルを修正する場合は、そのアーティクル全体の仕様を入力してください。

Sybase Central を使用するか、ALTER PUBLICATION 文によって、これらのタスクを実行できます。

注意

- DBA 権限を持つユーザか、パブリケーションの所有者だけがパブリケーションを変更できます。
- 変更には十分注意してください。Mobile Link 設定の実行中にパブリケーションを変更すると、エラーが発生し、データが失われることがあります。変更するパブリケーションにサブスクリプションが含まれている場合は、スキーマのアップグレードとして変更処理を行ってください。「リモート・クライアントでのスキーマの変更」 83 ページを参照してください。

♦ **既存のパブリケーションまたはアーティクルのプロパティを修正するには、次の手順に従います (Sybase Central の管理モードの場合)。**

1. パブリケーションを所有するユーザ、または DBA 権限を持つユーザとしてリモート・データベースに接続します。
2. 左ウィンドウ枠で、パブリケーションまたはアーティクルをクリックします。プロパティが右ウィンドウ枠に表示されます。
3. プロパティを設定します。

♦ **アーティクルを追加するには、次の手順に従います (Sybase Central の管理モードの場合)。**

1. SQL Anywhere プラグインを使用して、パブリケーションを所有するユーザ、または DBA 権限を持つユーザとしてリモート・データベースに接続します。
2. **[パブリケーション]** フォルダを展開します。
3. パブリケーションを選択します。
4. **[ファイル]** - **[新規]** - **[アーティクル]** を選択します。
5. **アーティクル作成ウィザード**で、次の作業を実行します。
   - **[このアーティクルに使用するテーブルを指定してください。]** リストでテーブルを選択します。**[次へ]** をクリックします。
   - **[選択したカラム]** をクリックし、カラムを選択します。**[次へ]** をクリックします。
   - **[このアーティクルに WHERE 句を指定できます。]** ウィンドウ枠で、オプションの WHERE 句を入力します。**[完了]** をクリックします。

♦ **アーティクルを削除するには、次の手順に従います (Sybase Central の管理モードの場合)。**

1. SQL Anywhere プラグインを使用して、パブリケーションを所有するユーザ、または DBA 権限を持つユーザとしてデータベースに接続します。
2. **[パブリケーション]** フォルダを展開します。
3. パブリケーションを右クリックして、**[削除]** を選択します。
4. **[はい]** をクリックします。

♦ **既存のパブリケーションを修正するには、次の手順に従います (SQL の場合)。**

1. パブリケーションを所有するユーザ、または DBA 権限を持つユーザとしてリモート・データベースに接続します。
2. ALTER PUBLICATION 文を実行します。

   「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

例

- 次の文は、customer テーブルを pub_contact パブリケーションに追加します。

```
ALTER PUBLICATION pub_contact
 ADD TABLE customer
```

「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』も参照してください。

パブリケーションの削除

Sybase Central または DROP PUBLICATION 文のいずれかを使用して、パブリケーションを削除できます。パブリケーションに接続されたサブスクリプションをすべて削除してから、パブリケーションを削除してください。

パブリケーションを削除するには、DBA 権限が必要です。

♦ パブリケーションを削除するには、次の手順に従います (Sybase Central の管理モードの場合)。

1. SQL Anywhere プラグインを使用して、DBA 権限のあるユーザとしてリモート・データベースに接続します。
2. **[パブリケーション]** フォルダを開きます。
3. パブリケーションを右クリックして、**[削除]** を選択します。

♦ パブリケーションを削除するには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてリモート・データベースに接続します。
2. DROP PUBLICATION 文を実行します。

   「DROP PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

例

次の文は、パブリケーション pub_orders を削除します。

```
DROP PUBLICATION pub_orders
```


Mobile Link ユーザの作成

Mobile Link のユーザ名は、Mobile Link サーバに接続するときの認証に使用されます。Mobile Link ユーザをリモート・データベースに作成し、統合データベースに登録してください。

Mobile Link ユーザは、データベース・ユーザとは異なります。データベース・ユーザ名と一致する Mobile Link ユーザ名を作成することはできますが、Mobile Link も SQL Anywhere も、名前の一致の影響は受けません。

♦ Mobile Link ユーザをリモート・データベースに追加するには、次の手順に従います (Sybase Central 管理モードの場合)。

1. SQL Anywhere プラグインから、DBA 権限のあるユーザとしてデータベースに接続します。
2. **[Mobile Link ユーザ]** フォルダをクリックします。
3. **[ファイル] - [新規] - [ユーザ]** を選択します。
4. **[新しいユーザの名前を指定してください。]** フィールドに、Mobile Link ユーザの名前を入力します。
5. **[完了]** をクリックします。

♦ Mobile Link ユーザをリモート・データベースに追加するには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. CREATE SYNCHRONIZATION USER 文を実行します。Mobile Link ユーザ名はリモート・データベースをユニークに識別します。このため、Mobile Link ユーザ名は同期システム内でユニークである必要があります。

   次は、Mobile Link ユーザ SSinger を追加する例です。

   ```
   CREATE SYNCHRONIZATION USER SSinger
   ```

   Mobile Link ユーザのプロパティは、CREATE SYNCHRONIZATION USER 文の一部として指定したり、別個に ALTER SYNCHRONIZATION USER 文で指定したりします。

   詳細については、「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   パスワードなど、Mobile Link ユーザのプロパティを設定する方法については、「Mobile Link ユーザの拡張オプションの格納」 114 ページを参照してください。

Mobile Link ユーザの登録については、「統合データベースへの Mobile Link ユーザ名の追加」 11 ページを参照してください。

Mobile Link ユーザの拡張オプションの格納

リモート・データベースで各 Mobile Link ユーザのオプションを指定するには、拡張オプションを使用します。拡張オプションは、コマンド・ラインで指定、データベースに格納、sp_hook_dbmlsync_set_extended_options イベント・フックで指定できます。

拡張オプションのリストについては、「Mobile Link SQL Anywhere クライアントの拡張オプション」 193 ページを参照してください。

♦ データベースに Mobile Link 拡張オプションを保存するには、次の手順に従います (Sybase Central 管理モードの場合)。

1. SQL Anywhere プラグインから、DBA 権限のあるユーザとしてデータベースに接続します。
2. **[Mobile Link ユーザ]** フォルダを開きます。
3. Mobile Link ユーザ名を右クリックし、**[プロパティ]** を選択します。
4. 必要に応じてプロパティを変更します。

♦ データベースに Mobile Link 拡張オプションを保存するには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. ALTER SYNCHRONIZATION SUBSCRIPTION 文を実行します。

   次は、Mobile Link ユーザ SSinger の拡張オプションをデフォルト値に変更する例です。

   ```
   ALTER SYNCHRONIZATION USER SSinger
   DELETE ALL OPTION
   ```

   詳細については、「ALTER SYNCHRONIZATION USER 文 [Mobile Link]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

   Mobile Link ユーザ名を作成するときに、プロパティを指定することも可能です。

   詳細については、「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

♦ クライアント・イベント・フックを使用して Mobile Link ユーザ・プロパティを指定するには、次の手順に従います。

- 次の同期の動作はプログラムを使用してカスタマイズできます。

  詳細については、「sp_hook_dbmlsync_set_extended_options」 304 ページを参照してください。

参照

- 「dbmlsync 拡張オプションの使用」 119 ページ

Mobile Link ユーザの削除

Mobile Link ユーザは、そのユーザ用のサブスクリプションをすべて削除した後で、リモート・データベースから削除してください。

♦ Mobile Link ユーザをリモート・データベースから削除するには、次の手順に従います (Sybase Central 管理モードの場合)。

1. SQL Anywhere プラグインから、DBA 権限のあるユーザとしてデータベースに接続します。
2. **[Mobile Link ユーザ]** フォルダで、対象の Mobile Link ユーザを探します。
3. Mobile Link ユーザを右クリックし、**[削除]** を選択します。

♦ Mobile Link ユーザをリモート・データベースから削除するには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. DROP SYNCHRONIZATION SUBSCRIPTION 文を実行します。

   次は、データベースから Mobile Link ユーザ SSinger を削除する例です。

   ```
   DROP SYNCHRONIZATION USER SSinger
   ```

   詳細については、「DROP SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

同期サブスクリプションの作成

Mobile Link ユーザとパブリケーションを作成後、1 つまたは複数の既存のパブリケーションに対して、少なくとも 1 人の Mobile Link ユーザのサブスクリプションを作成してください。これを行うには、同期サブスクリプションを作成します。

パブリケーションの作成については、「データのパブリッシュ」 105 ページを参照してください。Mobile Link ユーザの作成については、「Mobile Link ユーザの作成」 113 ページを参照してください。

注意
Mobile Link ユーザのすべてのサブスクリプションが、1 つの統合データベースに対してのみ同期されていることを確認する必要があります。複数の統合データベースに同期されていると、データの損失や予期しない動作が発生する場合があります。

同期サブスクリプションは、特定の Mobile Link ユーザをパブリケーションとリンクします。また、同期に必要なその他の情報を含めることもできます。たとえば、Mobile Link サーバのアドレスや、同期サブスクリプションに使用する他のオプションを指定できます。特定の同期サブスクリプションの値によって、Mobile Link ユーザに設定された値が上書きされます。

同期サブスクリプションは、Mobile Link SQL Anywhere リモート・データベース内でのみ必要です。サーバ論理は、統合データベース内の Mobile Link システム・テーブルに格納されている同期スクリプトによって実装されます。

単一の SQL Anywhere データベースは複数の Mobile Link サーバと同期できます。複数のサーバとの同期を行うには、サーバごとに異なる Mobile Link ユーザを作成します。

「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

例

SQL Anywhere サンプル・データベース内の Customers テーブルと SalesOrders テーブルの同期を行うには、次の文を使用します。

1. 最初に、Customers テーブルと SalesOrders テーブルをパブリッシュします。パブリケーション名として testpub を指定します。

```
CREATE PUBLICATION testpub
 (TABLE Customers, TABLE SalesOrders)
```

2. 次に Mobile Link ユーザを作成します。この場合、Mobile Link ユーザは demo_ml_user です。

```
CREATE SYNCHRONIZATION USER demo_ml_user
```

3. 処理を完了するために、ユーザとパブリケーションにリンクするサブスクリプションを作成します。

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


Mobile Link サブスクリプションの変更

Sybase Central を使用するか、ALTER SYNCHRONIZATION SUBSCRIPTION 文を使用すると、同期サブスクリプションを変更できます。構文は CREATE SYNCHRONIZATION SUBSCRIPTION 文に似ていますが、より簡単に追加、修正、削除できるように拡張オプションが用意されています。

♦ 同期サブスクリプションを変更するには、次の手順に従います (Sybase Central 管理モードの場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. **[Mobile Link ユーザ]** フォルダを開きます。
3. ユーザをクリックします。プロパティが右ウィンドウ枠に表示されます。
4. 右ウィンドウ枠で、**[サブスクリプション]** タブをクリックします。変更するサブスクリプションを右クリックし、**[プロパティ]** を選択します。
5. 必要に応じてプロパティを変更します。

♦ 同期サブスクリプションを変更するには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. ALTER SYNCHRONIZATION SUBSCRIPTION 文を実行します。

「ALTER SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

Mobile Link サブスクリプションの削除

Sybase Central または DROP SYNCHRONIZATION SUBSCRIPTION 文のいずれかを使用して、同期サブスクリプションを削除できます。

同期サブスクリプションを削除するには、DBA 権限が必要です。

♦ 同期サブスクリプションを削除するには、次の手順に従います (Sybase Central 管理モードの場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. **[Mobile Link ユーザ]** フォルダを開きます。
3. Mobile Link ユーザを選択します。
4. サブスクリプションを右クリックして、**[削除]** を選択します。

♦ 同期サブスクリプションを削除するには、次の手順に従います (SQL の場合)。

1. DBA 権限のあるユーザとしてデータベースに接続します。

2. DROP SYNCHRONIZATION SUBSCRIPTION 文を実行します。

例

次の文は、パブリケーション pub_orders に対する Mobile Link ユーザ jsmith の同期サブスクリプションを削除します。

```
DROP SYNCHRONIZATION SUBSCRIPTION
FOR jsmith TO pub_orders
```

「DROP SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

同期の開始

Mobile Link 同期を開始するのは、常にクライアントです。SQL Anywhere クライアントの場合は、dbmlsync ユーティリティを実行することで同期処理が開始されます。このユーティリティは SQL Anywhere リモート・データベースへの接続と同期を行います。

「Mobile Link SQL Anywhere クライアント・ユーティリティ (dbmlsync)」 137 ページを参照してください。

dbmlsync コマンド・ラインで -c オプションを使用して接続パラメータを指定できます。これらのパラメータは、リモート・データベース用です。接続パラメータを指定しないと、[接続] ウィンドウが表示され、必要な接続パラメータと起動オプションを指定するように要求されます。

「-c オプション」 151 ページを参照してください。

クライアントのネットワーク・プロトコル・オプションは、リモート・データベースの同期サブスクリプション、パブリケーション、ユーザに保存するか、dbmlsync コマンド・ラインで指定できます。これらのオプションは、適切な Mobile Link サーバの検索に使用されます。

「CommunicationAddress (adr) 拡張オプション」 197 ページを参照してください。

dbmlsync のパーミッション

dbmlsync でデータベースに接続するときは、処理中のすべての変更を適用するパーミッションが必要です。dbmlsync コマンド・ラインには、この接続用のパスワードが含まれます。このため、セキュリティ上の問題が発生する可能性があります。

セキュリティの問題を回避するには、ユーザ (DBA 以外) に REMOTE DBA 権限を付与し、このユーザ ID を dbmlsync 接続文字列に使用します。REMOTE DBA 権限を付与されたユーザ ID が DBA 権限を持つのは、dbmlsync ユーティリティから接続が確立された場合のみです。同じユーザ ID を使用する他の接続には、特別な権限は付与されません。

「GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

同期のカスタマイズ

「dbmlsync の同期のカスタマイズ」 130 ページを参照してください。

dbmlsync 拡張オプションの使用

Mobile Link には、同期処理をカスタマイズするための拡張オプションがいくつか用意されています。拡張オプションは、パブリケーション、ユーザ、またはサブスクリプションに設定できます。また、拡張オプションの値は、dbmlsync コマンド・ラインでオプションを使用して上書きできます。

拡張オプションの完全なリストは、「Mobile Link SQL Anywhere クライアントの拡張オプション」 193 ページを参照してください。

♦ **dbmlsync コマンド・ラインで拡張オプションを上書きするには、次の手順に従います。**

- -e または -eu dbmlsync オプションで、dbmlsync の拡張オプションの値を *option-name=value* の形式で指定します。次に例を示します。

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

♦ **サブスクリプション、パブリケーション、またはユーザの拡張オプションを設定するには、次の手順に従います。**

- SQL Anywhere リモート・データベース内で、CREATE SYNCHRONIZATION SUBSCRIPTION 文または CREATE SYNCHRONIZATION USER 文にオプションを追加します。

  パブリケーションに拡張オプションを追加する場合は、少し異なります。パブリケーションに拡張オプションを追加するには、ALTER/CREATE SYNCHRONIZATION SUBSCRIPTION 文で FOR 句を省略します。

例

次の文は、拡張オプションを使用する同期サブスクリプションを作成します。拡張オプションによって、アップロード・ストリームを準備するキャッシュ・サイズを 3 MB に設定し、アップロードのインクリメント・サイズを 3 KB に設定します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION TO my_pub
FOR ml_user
ADDRESS 'host=test.internal;port=2439;'
OPTION memory='3m',increment='3k'
```

オプション値は一重引用符で囲むことができますが、オプション名は引用符で囲まないでください。

dbmlsync ネットワーク・プロトコル・オプション

dbmlsync 接続情報には、サーバとの通信に使用するプロトコル、Mobile Link サーバのアドレスなどの接続パラメータが含まれます。

詳細については、次の項を参照してください。

- 「CommunicationType (ctp) 拡張オプション」 199 ページ
- 「CommunicationAddress (adr) 拡張オプション」 197 ページ

トランザクション・ログ・ファイル

多くの場合、何をアップロードするかは、dbmlsync によって SQL Anywhere トランザクション・ログを使用して決定されます。オフセットは、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。

SQL Anywhere データベースは、デフォルトでトランザクション・ログを管理します。データベースを作成するときに、トランザクション・ログの格納場所と、トランザクション・ログを保持するかどうかを指定できます。

スクリプト化されたアップロードを実装したり、ダウンロード専用のパブリケーションのみを使用する場合は、トランザクション・ログが必要ない場合があります。

アップロードの準備において、dbmlsync ユーティリティは、同期している Mobile Link ユーザのすべてのサブスクリプションが最後に正常に同期された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。ただし、通常、SQL Anywhere ログ・ファイルは、定期的なデータベース管理作業の中でトランケートされ、名前が変更されます。その場合は、記述されている変更内容がすべて正常に同期されるまで、古いログ・ファイルの名前を変更し、別のディレクトリに保存してください。

dbmlsync コマンド・ラインで、名前が変更されたログ・ファイルが格納されているディレクトリを指定できます。前回の同期の後で作業ログ・ファイルのトランケートと名前の変更が行われていない場合、または名前が変更されたログ・ファイルがあるディレクトリから dbmlsync を実行する場合は、このパラメータを省略できます。

参照

- 「バックアップとデータ・リカバリ」 『SQL Anywhere サーバ - データベース管理』
- 「進行オフセット」 103 ページ
- 「トランザクション・ログ」 『SQL Anywhere サーバ - データベース管理』
- 「初期化ユーティリティ (dbinit)」 『SQL Anywhere サーバ - データベース管理』
- 「スクリプト化されたアップロード」 385 ページ

例

古いログ・ファイルがディレクトリ *c:¥oldlogs* に格納されているとします。次のコマンドを使用してリモート・データベースを同期することができます。

```
dbmlsync -c "dbn=remote;uid=syncuser" c:¥oldlogs
```

古いログ・ディレクトリへのパスは、コマンド・ラインに最後の引数として指定してください。

同期中の同時実行性

同期の整合性を確保するために、dbmlsync ではアップロードが構築されてからダウンロードが適用されるまでの間に、ダウンロードのローが修正されないようにする必要があります。

Windows Mobile 以外のプラットフォームでは、デフォルトの動作として dbmlsync は同期中のパブリケーションで指定されているすべてのテーブルに対する共有ロックを取得します。
Windows Mobile では、デフォルトの動作として dbmlsync は排他ロックを取得します。dbmlsync は、アップロードの構築を開始する前にロックを取得し、ダウンロードが適用されるまでそのロックを保持します。

ロックの詳細については、「ロー・ロック」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

このロック動作は、次のオプションを使用してカスタマイズできます。

- -d オプション
- LockTables オプション

-d オプション

このロック・メカニズムを使用しているときに、データベースに別の接続が存在し、その接続に同期テーブルに対するロックがある場合は、同期が失敗します。別のロックが存在しても、同期がすぐに行われるようにする場合は、dbmlsync で -d オプションを使用します。このオプションを指定すると、同期に影響するロックのある接続はデータベースによって削除されるため、同期を進行できます。削除された接続のコミットされていない変更は、ロールバックされます。

詳細については、「-d オプション」 152 ページを参照してください。

LockTables オプション

データの整合性を保持する別の方法は、LockTables 拡張オプションを OFF に設定することです。OFF にすると、アーティクルのテーブルがロックされるのを防ぎます。これにより、dbmlsync はアップロードの構築後に修正されたローをすべて追跡します。ダウンロードを受信しても、そのローが修正されている場合はダウンロードは適用されません。この場合、dbmlsync は同期のリトライを行います。リトライは、新しいダウンロードの競合が検出されないかぎり正常に実行されます。

詳細については、「LockTables (lt) 拡張オプション」 214 ページを参照してください。

競合が検出された場合は、ダウンロード・フェーズがキャンセルされ、新しい変更が上書きされないようにダウンロード操作がロールバックされます。次に、dbmlsync ユーティリティはアップロード手順を含む同期を再試行します。今度はローが同期処理の最初に処理されており、アップロードにこのローが含まれているため、このローを失うことはありません。

デフォルトでは、dbmlsync は正常に実行されるまで同期のリトライを行います。リトライの回数を制限するには、拡張オプション ConflictRetries を使用します。ConflictRetries を -1 に設定すると、正常に実行されるまで dbmlsync によってリトライが実行されます。これを正の整数に設定すると、dbmlsync は指定した回数以内でリトライを実行します。

詳細については、「ConflictRetries (cr) 拡張オプション」 200 ページを参照してください。

アプリケーションからの同期の開始

リモート・ユーザに別個の実行ファイルを提供するのではなく、アプリケーションに dbmlsync の機能を組み込むことができます。

このようにするには、次の 2 つの方法があります。

- dbmlsync 統合コンポーネントを使用します。

  詳細については、「dbmlsync 統合コンポーネント (旧式)」 349 ページを参照してください。

- DLL を呼び出すことのできる言語で開発している場合は、DBTools インタフェースを使用して dbmlsync にアクセスできます。C や C++ でプログラムを作成している場合は、SQL Anywhere 11 ディレクトリの *SDK¥Include* サブディレクトリにある *dbtools.h* ヘッダ・ファイ

ルを含めることができます。このファイルには、a_sync_db 構造体と DBSynchronizeLog 関数の記述があり、dbmlsync 機能をアプリケーションに追加するときに使用します。この解決方法は、Windows や UNIX など、サポート対象となっているすべてのプラットフォームに使用できます。

詳細については、次の項を参照してください。

○「dbmlsync の DBTools インタフェース」 377 ページ
○「DBSynchronizeLog 関数」『SQL Anywhere サーバ - プログラミング』
○「a_sync_db 構造体」『SQL Anywhere サーバ - プログラミング』

ActiveSync 同期の使用

ActiveSync は、Microsoft Windows Mobile ハンドヘルド・デバイス用の同期ソフトウェアです。ActiveSync は、Windows Mobile デバイスとデスクトップ・コンピュータ間の同期を制御します。ActiveSync 用の Mobile Link プロバイダは、Mobile Link サーバとの同期を制御します。

SQL Anywhere クライアント用の ActiveSync 同期を設定するには、次の手順に従います。

- SQL Anywhere リモート・データベースを ActiveSync 同期用に設定する。

  「ActiveSync 用の SQL Anywhere リモート・データベースの設定」 124 ページを参照してください。

- ActiveSync 用 Mobile Link プロバイダをインストールする。

  「ActiveSync 用 Mobile Link プロバイダのインストール」 125 ページを参照してください。

- SQL Anywhere クライアントを、ActiveSync で使用できるように登録する。

  「ActiveSync 用 SQL Anywhere クライアントの登録」 126 ページを参照してください。

ActiveSync 同期を使用する場合は、ActiveSync ソフトウェアから同期を開始します。ActiveSync 用の Mobile Link プロバイダは、dbmlsync を起動するか、スケジュール文字列でのスケジュールに従ってスリープ中の dbmlsync をウェイクアップできます。

リモート・データベース内で遅延フックを使用して dbmlsync をスリープ・モードに設定することもできますが、ActiveSync 用の Mobile Link プロバイダは、このステータスからは同期を開始できません。

同期スケジュールの詳細については、「同期のスケジュール」 128 ページを参照してください。

ActiveSync 用の SQL Anywhere リモート・データベースの設定

♦ SQL Anywhere リモート・データベースを ActiveSync 用に設定するには、次の手順に従います。

1. 同期タイプ (TCP/IP、TLS、HTTP、または HTTPS) を選択します。

   同期タイプは、同期パブリケーション、同期ユーザ、または同期サブスクリプション用に設定できます。それぞれの設定方法は似ています。ここでは、典型的な CREATE SYNCHRONIZATION USER 文の一部を示します。

   ```
   CREATE SYNCHRONIZATION USER SSinger
   TYPE tcpip
   ...
   ```

2. ADDRESS 句を使用して、ActiveSync 用 Mobile Link プロバイダと Mobile Link サーバ間の通信を指定します。

HTTP または TCP/IP 同期の場合は、CREATE SYNCHRONIZATION USER または CREATE SYNCHRONIZATION SUBSCRIPTION 文の ADDRESS 句によって、Mobile Link クライアントとサーバ間の通信を指定します。ActiveSync の場合、通信は 2 段階で発生します。つまり、デバイス上の dbmlsync ユーティリティからデスクトップ・コンピュータ上の ActiveSync 用 Mobile Link プロバイダへの通信が発生してから、デスクトップ・コンピュータから Mobile Link サーバへの通信が発生します。ADDRESS 句では、ActiveSync 用 Mobile Link プロバイダと Mobile Link サーバ間の通信を指定します。

次の文は、コンピュータ kangaroo 上の Mobile Link サーバへの TCP/IP 通信を指定します。

```
CREATE SYNCHRONIZATION USER SSinger
TYPE tcpip
ADDRESS 'host=kangaroo;port=2439'
```

詳細については、「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

ActiveSync 用 Mobile Link プロバイダのインストール

インストール・ユーティリティ (*mlasinst.exe*) を使用して、ActiveSync 用 Mobile Link プロバイダをインストールしてから、SQL Anywhere Mobile Link クライアントを ActiveSync 用に登録します。

SQL Anywhere for Windows Mobile のインストーラによって、ActiveSync 用 Mobile Link プロバイダがインストールされます。SQL Anywhere for Windows Mobile をインストールする場合、この項で説明する手順を実行する必要はありません。

ActiveSync 用 Mobile Link プロバイダをインストールしたら、各アプリケーションを個別に登録してください。手順については、「ActiveSync 用 SQL Anywhere クライアントの登録」 126 ページを参照してください。

♦ ActiveSync 用 Mobile Link プロバイダをインストールするには、次の手順に従います。

1. 使用中のコンピュータに ActiveSync ソフトウェアがインストールしてあり、Windows Mobile デバイスが接続されていることを確認します。
2. 次のコマンドを実行して、Mobile Link プロバイダをインストールします。

   ```
   mlasinst -k desk-path -v dev-path
   ```

   desk-path はプロバイダのデスクトップ・コンポーネント (*mlasdesk.dll*) のロケーション、*dev-path* はデバイス・コンポーネント (*mlasdev.dll*) のロケーションです。

   コンピュータに SQL Anywhere がインストールされている場合、*mlasdesk.dll* は *install-dir¥bin32* にあり、*mlasdev.dll* は *install-dir¥CE* にあります。-v または -k を省略すると、デフォルトでこれらのディレクトリが検索されます。

   リモート・プロバイダが開けないというメッセージが表示された場合は、デバイスのソフト・リセットを実行し、コマンドを繰り返します。

詳細については、「ActiveSync プロバイダ・インストール・ユーティリティ (mlasinst)」 29 ページを参照してください。

3. コンピュータを再起動します。

   コンピュータを再起動すると、ActiveSync で新しいプロバイダが認識されます。

4. Mobile Link プロバイダを有効にします。

   Vista よりも前のバージョンの Windows の場合：

   - [ActiveSync] ウィンドウで **[オプション]** をクリックします。
   - リストにある [Mobile Link] 項目を有効にして **[OK]** をクリックし、Mobile Link プロバイダをアクティブにします。
   - 登録されたアプリケーションのリストを表示するには、もう一度 **[オプション]** をクリックし、Mobile Link プロバイダを選択して **[設定]** をクリックします。

     アプリケーションの登録についての詳細は、「ActiveSync 用 SQL Anywhere クライアントの登録」 126 ページを参照してください。

   Windows Vista の場合：

   - [Windows Mobile デバイス センター] ウィンドウで、**[モバイル デバイスの設定]** - **[コンテンツの設定の変更]** をクリックします。
   - **[Mobile Link クライアント]** を選択し、**[保存]** をクリックして Mobile Link プロバイダをアクティブにします。
   - 登録されたアプリケーションのリストを表示するには、**[コンテンツの設定の変更]** - **[Mobile Link クライアント]** - **[同期の設定]** をクリックします。

ActiveSync 用 SQL Anywhere クライアントの登録

ActiveSync で使用するアプリケーションを登録するには、ActiveSync プロバイダのインストール・ユーティリティを使用する方法と、ActiveSync ソフトウェア自体を使用する方法があります。この項では、ActiveSync ソフトウェアを使用する方法について説明します。

もう 1 つの方法については、「ActiveSync プロバイダ・インストール・ユーティリティ (mlasinst)」 29 ページを参照してください。

♦ SQL Anywhere クライアントを ActiveSync 用に登録するには、次の手順に従います。

1. ActiveSync 用 Mobile Link プロバイダがインストールされていることを確認します。

   詳細については、「ActiveSync 用 Mobile Link プロバイダのインストール」 125 ページを参照してください。

2. デスクトップ・コンピュータで、ActiveSync ソフトウェアを起動します。
3. Vista よりも前の Windows の場合：
   - **[ActiveSync]** ウィンドウで **[オプション]** を選択します。
   - 情報タイプのリストから、**[Mobile Link]** を選択し、**[設定]** をクリックします。

- **[Mobile Link 同期]** ウィンドウで **[新規]** をクリックします。

Windows Vista の場合：

- **[Windows Mobile デバイス センター]** ウィンドウで、**[モバイル デバイスの設定]** - **[コンテンツの設定の変更]** をクリックします。
- **[コンテンツの設定の変更]** をクリックします。
- **[Mobile Link クライアント]** をクリックします。
- **[同期の設定]** をクリックします。

4. アプリケーションについて次の情報を入力します。
   - **[アプリケーション名]** ActiveSync ユーザ・インタフェースに表示されるアプリケーションを識別する名前。
   - **[クラス名]** -wc オプションを使用して設定した、dbmlsync クライアントのクラス名。

     詳細については、「-wc オプション」 190 ページを参照してください。
   - **[パス]** デバイス上の dbmlsync アプリケーションのロケーション。
   - **[引数]** ActiveSync が dbmlsync の起動時に使用するコマンド・ライン引数。

     dbmlsync は、2 つのモードのうちの 1 つを使用して開始します。
     - スケジューリング・オプションを指定すると、dbmlsync は停止モードに入ります。この場合、クラス名の設定と一致する値を指定した dbmlsync -wc オプションを使用します。

       詳細については、「-wc オプション」 190 ページと「同期のスケジュール」 128 ページを参照してください。
     - このように指定しないと、dbmlsync は停止モードに入りません。この場合、-k を使用して dbmlsync を停止します。

       詳細については、「-k オプション (旧式)」 164 ページを参照してください。
5. **[OK]** をクリックしてアプリケーションを登録します。

同期のスケジュール

定義した規則に基づいて定期的に同期を実行するよう dbmlsync を設定できます。dbmlsync を設定する方法は 2 つあります。

- 特定の時刻や曜日、または定期的に同期を開始するには、dbmlsync 拡張オプションの SCHEDULE を使用します。この場合、ユーザが停止するまで dbmlsync は実行を続けます。

  「dbmlsync オプションを使用したスケジュールの設定」 128 ページを参照してください。

- 定義した論理に基づいて同期を開始するには、dbmlsync イベント・フックを使用します。この方法は、不定期またはイベントの応答として同期を起動する場合に適しています。この場合、指定したフック・コードによって dbmlsync を自動的に停止できます。

  「イベント・フックを使用した同期の開始」 129 ページを参照してください。

停止

スケジュール・オプションやフックを指定すると、dbmlsync は停止モードになります。停止とは、ログのスキャンに使用される時間を少なくする機能です。停止を実行してパフォーマンスを向上させるには、dbmlsync 拡張オプション HoverRescanThreshold を設定するか、dbmlsync ストアド・プロシージャ sp_hook_dbmlsync_log_rescan を使用します。

詳細については、次の項を参照してください。

- 「HoverRescanThreshold (hrt) 拡張オプション」 210 ページ
- 「sp_hook_dbmlsync_log_rescan」 288 ページ

dbmlsync オプションを使用したスケジュールの設定

dbmlsync をバッチ形式で実行して、同期終了後に停止するように設定する代わりに、dbmlsync を継続的に実行してあらかじめ決められた時間に同期を行うように、SQL Anywhere クライアントを設定することもできます。

同期スケジュールは、拡張オプションとして指定します。同期スケジュールを dbmlsync コマンド・ライン上で指定するか、同期ユーザ、同期サブスクリプション、または同期パブリケーション用にデータベースに同期スケジュールを格納できます。

スケジュール構文については、「Schedule (sch) 拡張オプション」 223 ページを参照してください。

拡張オプションの詳細については、次の項を参照してください。

- 「Mobile Link SQL Anywhere クライアントの拡張オプション」 193 ページ
- 「-eu オプション」 162 ページ

♦ 同期サブスクリプションにスケジュールを追加するには、次の手順に従います。

- 同期サブスクリプション内で Schedule 拡張オプションを設定します。次に例を示します。


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

dbmlsync -is オプションを使用すると、スケジュールの設定を無効にし、直ちに同期を行うことができます。-is オプションは、スケジュール拡張オプションで指定したスケジュールを無視するよう dbmlsync に指示します。詳細については、「-is オプション」 163 ページを参照してください。

♦ **dbmlsync コマンド・ラインでスケジュールを追加するには、次の手順に従います。**

● スケジュール拡張オプションを設定します。拡張オプションは -e または -eu で設定します。次に例を示します。

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

同期スケジュールがこのどちらかで指定された場合、dbmlsync は同期終了後も停止せず、継続して実行します。

イベント・フックを使用した同期の開始

同期処理のタイミングを制御するために実装できる dbmlsync イベント・フックがあります。

sp_hook_dbmlsync_end フックを使用すると、#hook_dict テーブルの Restart ローを使用して、同期処理が終了するたびに dbmlsync が同期を繰り返すかどうかを判断できます。

詳細については、「sp_hook_dbmlsync_end」 285 ページを参照してください。

sp_hook_dbmlsync_delay フックを使用すると、同期処理の開始時に遅延を作成して、同期を続行するタイミングを選択できます。このフックでは、一定の時間遅延させたり、定期的にポーリングを行うことで、ある条件が満たされるまで待機できます。

詳細については、「sp_hook_dbmlsync_delay」 265 ページを参照してください。

dbmlsync の同期のカスタマイズ

dbmlsync クライアント・イベント・フック

イベント・フックでは、SQL ストアド・プロシージャを使用して、dbmlsync のクライアント側の同期処理を管理できます。クライアント・イベント・フックは、dbmlsync コマンド・ライン・ユーティリティや dbmlsync プログラミング・インタフェースで使用できます。

イベント・フックを使用して同期イベントのログを取り、処理することができます。たとえば、論理イベントに基づいた同期のスジュール、接続障害のリトライ、または特定のエラーや参照整合性違反の処理などが可能です。

クライアント・イベント・フックの詳細については、「SQL Anywhere クライアントのイベント・フック」 247 ページを参照してください。

dbmlsync プログラミング・インタフェース

次のプログラミング・インタフェースを使用して、Mobile Link クライアントをアプリケーションに統合し、同期を開始できます。これらのインタフェースは、dbmlsync コマンド・ライン・ユーティリティの代わりに使用できます。

- **dbmlsync API** dbmlsync API は、C++ または .NET で記述された Mobile Link クライアントが同期を起動し、要求した同期の進行状況に関するフィードバックを受け取れるようにするプログラミング・インタフェースです。この新しいプログラミング・インタフェースを使用することで、同期の結果に関してアクセスできる情報が大幅に増え、同期のキューイングも可能になるため、同期の管理が容易になります。

  「dbmlsync API の概要」 320 ページを参照してください。

- **dbmlsync の DBTools インタフェース** dbmlsync 用の DBTools インタフェースを使用することで、SQL Anywhere 同期クライアント・アプリケーションに同期機能を統合できます。SQL Anywhere データベース管理ユーティリティはすべて、DBTools によって構築されます。

  「dbmlsync の DBTools インタフェース」 377 ページを参照してください。

スクリプト化されたアップロード

また、クライアント・トランザクション・ログの使用を上書きして、独自のアップロード・ストリームを定義することもできます。「スクリプト化されたアップロード」 385 ページを参照してください。

SQLAnywhere クライアントのロギング

SQL Anywhere リモート・データベースを使用する Mobile Link アプリケーションを作成する場合、注意が必要なクライアント・ログ・ファイルは 2 種類あります。

- dbmlsync のメッセージ・ログ
- SQL Anywhere トランザクション・ログ

dbmlsync のメッセージ・ログ

デフォルトでは、dbmlsync のメッセージは dbmlsync のメッセージ・ウィンドウに送信されます。また、-o または -ot オプションを使用して結果をメッセージ・ログ・ファイルにも送信できます。次のコマンド・ラインの一部では、結果を *dbmlsync.log* という名前のログ・ファイルに送ります。

```
dbmlsync -o dbmlsync.log ...
```

dbmlsync アクティビティをロギングすると、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。

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。

- ログ・ファイルを指定して、結果をログ・ファイルに追加する場合は、-o オプションを使用します。
- ログ・ファイルを指定して、結果をログ・ファイルに追加する前にファイルの内容を削除する場合は、-ot オプションを使用します。
- -o または -ot に加えて -os オプションを使用してサイズを指定すると、そのサイズに達したときに、ログ・ファイルの名前が変更され、元の名前を持つ新しいファイルが使用されます。

詳細については、次の項を参照してください。

- 「-o オプション」 169 ページ
- 「-ot オプション」 171 ページ
- 「-os オプション」 170 ページ

-v オプションを使用すると、メッセージ・ログ・ファイルに記録され、dbmlsync のメッセージ・ウィンドウに表示される情報を制御できます。

詳細については、「-v オプション」 189 ページを参照してください。

delete_old_logs オプションを使用すると、ログ・ファイルを管理できます。

詳細については、「delete_old_logs オプション [Mobile Link クライアント] [SQL Remote] [Replication Agent]」 『SQL Anywhere サーバ - データベース管理』を参照してください。

メッセージ・ログ・ファイルを指定しなかった場合、すべての出力が dbmlsync のメッセージ・ウィンドウに表示されます。メッセージ・ログ・ファイルを指定した場合、dbmlsync のメッセージ・ウィンドウに送信される出力は指定しなかった場合よりも少なくなります。

SQL Anywhere トランザクション・ログ

「トランザクション・ログ・ファイル」 120 ページを参照してください。

Mobile Link の Mac OS X での実行

Mobile Link サーバと SQL Anywhere Mobile Link クライアントは、Mac OS X で実行できます。Ultra Light は、Mac OS X では実行できません。

Mac OS X 上の Mobile Link 統合データベースを同期するには、ドライバ・マネージャとして SQL Anywhere ODBC ドライバを使用します。「Mac OS X での ODBC データ・ソースの作成」『SQL Anywhere サーバ - データベース管理』を参照してください。

♦ Mobile Link サーバを Mac OS X で開始するには、次の手順に従います。

1. SyncConsole を起動します。

   [Finder] で、SyncConsole をダブルクリックします。SyncConsole アプリケーションは */Applications/SQLAnywhere11* にあります。

2. **[ファイル] - [新規] - [Mobile Link サーバ]** を選択します。
3. Mobile Link サーバを設定します。
   a. **[接続パラメータ]** フィールドに次の文字列を入力します。

      ```
      dsn=dsn-name
      ```

      dsn-name は SQL Anywhere ODBC データ・ソース名です。ODBC データ・ソースの作成については、「UNIX と Mac OS X での環境変数の設定」『SQL Anywhere サーバ - データベース管理』を参照してください。

      dsn-name にスペースが含まれている場合は、文字列を二重引用符で囲みます。次に例を示します。

      ```
      dsn="SQL Anywhere 11 Demo"
      ```

   b. 必要に応じて、**[オプション]** フィールドでオプションを設定します。

      [オプション] フィールドを使用すると、Mobile Link サーバの動作をさまざまな面から制御できます。オプションの完全なリストについては、「mlsrv11 の構文」『Mobile Link - サーバ管理』を参照してください。

4. **[起動]** をクリックして、Mobile Link サーバを起動します。

   データベース・サーバ・メッセージ・ウィンドウが開き、サーバが同期要求を受け付ける準備ができていることを示すメッセージが表示されます。

♦ dbmlsync を Mac OS X で開始するには、次の手順に従います。

1. SyncConsole を起動します。

   [Finder] で、**SyncConsole** をダブルクリックします。SyncConsole アプリケーションは */Applications/SQLAnywhere11* にあります。

2. **[ファイル] - [新規] - [Mobile Link クライアント]** を選択します。

クライアント・オプションのウィンドウが表示されます。このウィンドウには設定オプションが多数用意されていますが、それぞれが dbmlsync コマンド・ライン・オプションに対応しています。完全なリストについては、「dbmlsync 構文」 139 ページを参照してください。

[ログイン]、**[データベース]**、**[ネットワーク]**、**[詳細]** の各タブのオプションはすべて、Mobile Link クライアントから SQL Anywhere リモート・データベースへの接続を定義しています。ほとんどの場合、**[ログイン]** タブの ODBC データ・ソースを指定するだけで接続できます。

[DBMLSync] タブのオプションは、Mobile Link サーバへの接続について定義します。この機能がリモート・データベースのパブリケーションおよびサブスクリプションで定義されている場合は、このタブのオプションは空のままにできます。

♦ **サンプル・データベースを Mac OS X で実行するには、次の手順に従います。**

1. *sa_config* 設定スクリプトを準備します。

   詳細については、「UNIX と Mac OS X での環境変数の設定」 『SQL Anywhere サーバ - データベース管理』を参照してください。

2. ODBC データ・ソースを設定します。次に例を示します。

   ```
   dbdsn -w  "SQL Anywhere 11 Demo"
   -c "uid=DBA;pwd=sql;dbf=/Applications/SQLAnywhere11/System/demo.db"
   ```

3. Mobile Link サーバを実行します。次に例を示します。

   ```
   mlsrv11 -c "dsn=SQL Anywhere 11 Demo"
   ```


バージョンに関する考慮事項

dbmlsync が正しく機能するためには、dbmlsync.exe のメジャー・バージョンとマイナー・バージョンの両方が、データベース・サーバと一致している必要があります。さらに、データベース・ファイルのメジャー・バージョンが dbmlsync.exe のメジャー・バージョンと一致していて、データベース・ファイルのマイナー・バージョンが *dbmlsync.exe* のマイナー・バージョン以下である必要があります。データベース・ファイルのバージョンとは、アップグレードされている最新のバージョンのことです。

たとえば、9.02 バージョンの dbmlsync は、9.02 バージョンのデータベース・サーバ (*dbeng9.exe*) に対してのみ使用し、9.00、9.01、9.02 のデータベース・ファイルを使用できます。

Mobile Link SQL Anywhere クライアント・ユーティリティ (dbmlsync)

目次

dbmlsync 構文

dbmlsync ユーティリティを使用して、SQL Anywhere リモート・データベースと統合データベースの同期を行います。

構文

dbmlsync [*options*] [*transaction-logs-directory*]

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みます。「@data オプション」 144 ページを参照してください。
-a	エラー時に再入力のプロンプトを表示しません。「-a オプション」 145 ページを参照してください。
-ap	認証パラメータを指定します。「-ap オプション」 146 ページを参照してください。
-ba *filename*	ダウンロード・ファイルを適用します。「-ba オプション」 147 ページを参照してください。
-bc *filename*	ダウンロード・ファイルを作成します。「-bc オプション」 148 ページを参照してください。
-be *string*	ダウンロード・ファイルを作成するときに文字列を追加します。「-be オプション」 149 ページを参照してください。
-bg	ダウンロード・ファイルを作成するときに、そのファイルを新しいリモートに適合するようにします。「-bg オプション」 150 ページを参照してください。
-c *connection-string*	*parm1=value1;parm2=value2,...* の形式で、データベース接続パラメータを指定します。このオプションを指定しなかった場合はウィンドウが表示され、そこで接続情報を指定します。「-c オプション」 151 ページを参照してください。
-d	ロックされているデータベースへの接続のうち、同期されたアーティクルと競合しているものをすべて削除します。「-d オプション」 152 ページを参照してください。
-dc	以前に失敗したダウンロードを続行します。「-dc オプション」 153 ページを参照してください。
-dl	dbmlsync のメッセージ・ウィンドウにログ・メッセージを表示します。「-dl オプション」 154 ページを参照してください。

オプション	説明
-do	オフライン・トランザクション・ログのスキャンを無効にします。「-do オプション」 155 ページを参照してください。
-drs *bytes*	再起動可能なダウンロードについて、通信障害の後に再送する必要があるデータの最大値を指定します。「-drs オプション」 156 ページを参照してください。
-ds	ダウンロード専用同期を実行します。「-ds オプション」 157 ページを参照してください。
-e "*option=value*"...	拡張オプションを指定します。「Mobile Link SQL Anywhere クライアントの拡張オプション」 193 ページを参照してください。
-eh	フック関数で発生したエラーを無視します。
-ek *key*	暗号化キーを指定します。「-ek オプション」 160 ページを参照してください。
-ep	暗号化キーを入力するよう要求します。「-ep オプション」 161 ページを参照してください。
-eu	最新の -n オプションで定義されたアップロードに対して拡張オプションを指定します。「-eu オプション」 162 ページを参照してください。
-is	スケジュールを無視します。「-is オプション」 163 ページを参照してください。
-k	完了後、ウィンドウを閉じます。「-k オプション (旧式)」 164 ページを参照してください。
-l	使用可能な拡張オプションをリストします。「-l オプション」 165 ページを参照してください。
-mn *password*	新しい Mobile Link パスワードを指定します。「-mn オプション」 166 ページを参照してください。
-mp *password*	Mobile Link パスワードを指定します。「-mp オプション」 167 ページを参照してください。
-n *name*	同期パブリケーション名を指定します。「-n オプション」 168 ページを参照してください。
-o *logfile*	このファイルに出力メッセージのログを取ります。「-o オプション」 169 ページを参照してください。

オプション	説明
-os *size*	メッセージ・ログ・ファイルの最大サイズを指定します (このサイズに達するとログの名前が変更されます)。「-os オプション」 170 ページを参照してください。
-ot *logfile*	メッセージ・ログ・ファイルの内容を削除してから、このファイルに出力メッセージのログを取ります。「-ot オプション」 171 ページを参照してください。
-p	ログスキャンのポーリングを無効にします。「-p オプション」 172 ページを参照してください。
-pc+	同期と同期の間で、Mobile Link サーバへのオープン接続を維持します。「-pc オプション」 173 ページを参照してください。
-pd *dllname*;...	Windows Mobile 用の指定された DLL をプリロードします。「-pd オプション」 174 ページを参照してください。
-pi	Mobile Link に接続できるかどうかをテストします。「-pi オプション」 175 ページを参照してください。
-pp *number*	ログスキャンのポーリング周期を設定します。「-pp オプション」 176 ページを参照してください。
-q	最小化ウィンドウで実行します。「-q オプション」 177 ページを参照してください。
-qc	同期が終了したときに dbmlsync を停止します。「-qc オプション」 178 ページを参照してください。
-r[**a** | **b**]	アップロードのリトライにクライアントの進行状況値を使用します。「-r オプション」 179 ページを参照してください。
-sc	各同期の前にスキーマ情報を再ロードします。「-sc オプション」 180 ページを参照してください。
-sp *sync profile*	コマンド・ラインで指定される同期オプションに、同期プロファイルからのオプションを追加します。「-sp オプション」 181 ページを参照してください。
-tu	トランザクション単位のアップロードを実行します。「-tu オプション」 182 ページを参照してください。
-u *ml_username*	同期する Mobile Link ユーザを指定します。「-u オプション」 184 ページを参照してください。

オプション	説明
-ui	X-Window がサポートされている Linux で、使用可能な表示がない場合にシェル・モードで dbmlsync を起動します。「-ui オプション」 185 ページを参照してください。
-uo	アップロード専用同期を実行します。「-uo オプション」 186 ページを参照してください。
-urc *row-estimate*	アップロードされるロー数の推定値を指定します。「-urc オプション」 187 ページを参照してください。
-ux	Solaris と Linux で、dbmlsync のメッセージ・ウィンドウを開きます。「-ux オプション」 188 ページを参照してください。
-v[*levels*]	冗長オペレーション。「-v オプション」 189 ページを参照してください。
-wc *classname*	ウィンドウ・クラス名を指定します。「-wc オプション」 190 ページを参照してください。
-x	トランザクション・ログの名前を変更して再起動します。「-x オプション」 191 ページを参照してください。
transaction-logs-directory	トランザクション・ログのロケーションを指定します。下のトランザクション・ログ・ファイルを参照してください。

備考

dbmlsync を実行して、SQL Anywhere リモート・データベースと統合データベースの同期を行います。

Mobile Link サーバを検出して接続するために、dbmlsync はパブリケーション、同期ユーザ、同期サブスクリプション、または dbmlsync コマンド・ラインの情報を使用します。

トランザクション・ログ・ファイル　*transaction-logs-directory* には、SQL Anywhere リモート・データベースのトランザクション・ログが格納されているディレクトリを指定します。アクティブなトランザクション・ログ・ファイルとトランザクション・ログ・アーカイブ・ファイルがあります。dbmlsync がアップロードするデータを判別するには、この両方が必要です。次のすべての条件を満たす場合、このパラメータを指定してください。

- 前回の同期の後で、作業ログ・ファイルの内容が削除され、ファイルの名前が変更されている場合
- 名前が変更されたログ・ファイルが格納されているディレクトリ以外のディレクトリから、dbmlsync ユーティリティを実行する場合

詳細については、「トランザクション・ログ・ファイル」 120 ページを参照してください。

dbmlsync イベント・フック　同期処理のカスタマイズに役立つ dbmlsync クライアント・ストアド・プロシージャもあります。詳細については、「dbmlsync のフックの概要」 249 ページと「SQL Anywhere クライアントのイベント・フック」 247 ページを参照してください。

dbmlsync の使用　dbmlsync の使用についての詳細は、「同期の開始」 119 ページを参照してください。

参照

- 「同期の開始」 119 ページ
- 「SQL Anywhere クライアントのイベント・フック」 247 ページ
- 「dbmlsync API」 319 ページ
- 「dbmlsync の DBTools インタフェース」 377 ページ

@data オプション

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

dbmlsync @*data* ...

備考

このオプションを指定すると、環境変数または設定ファイル内にコマンド・ライン・オプションを記述できます。指定された名前に環境変数と設定ファイルの両方が存在する場合、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、「設定ファイルの使用」『SQL Anywhere サーバ - データベース管理』を参照してください。

設定ファイル内のパスワードなどの情報を保護する場合は、ファイル難読化ユーティリティを使用して、設定ファイルの内容を難読化できます。

「ファイル難読化ユーティリティ (dbfhide)」『SQL Anywhere サーバ - データベース管理』を参照してください。

-a オプション

dbmlsync がエラー時に再入力のウィンドウ・プロンプトを表示しないように指定します。

構文

dbmlsync -a ...

-ap オプション

authenticate_parameters スクリプトと認証パラメータにパラメータを入力します。

構文

dbmlsync -ap "*parameters,...***"** ...

備考

authenticate_parameters 接続スクリプトや認証パラメータを使用するときに使用します。次に例を示します。

```
dbmlsync -ap "parm1,parm2,parm3"
```

パラメータは Mobile Link サーバに送信され、authenticate_parameters スクリプトや統合データベース上のその他のイベントに渡されます。

参照

- 「認証パラメータ」 『Mobile Link - サーバ管理』
- 「authenticate_parameters 接続イベント」 『Mobile Link - サーバ管理』

-ba オプション

ダウンロード・ファイルを適用します。

構文

dbmlsync -ba "*filename***"** ...

備考

リモート・データベースに適用する既存のダウンロード・ファイルの名前を指定します。オプションでパスを指定できます。パスを指定しない場合、デフォルト・ロケーションは dbmlsync が起動されたディレクトリです。

参照

- 「Mobile Link ファイルベースのダウンロード」 『Mobile Link - サーバ管理』
- 「-bc オプション」 148 ページ
- 「-be オプション」 149 ページ
- 「-bg オプション」 150 ページ

-bc オプション

ダウンロード・ファイルを作成します。

構文

dbmlsync -bc "*filename***"** ...

備考

指定された名前でダウンロード・ファイルを作成します。ダウンロード・ファイルにはファイル拡張子 .df を使用してください。

オプションでパスを指定できます。パスを指定しない場合、デフォルト・ロケーションは dbmlsync の現在の作業ディレクトリ (dbmlsync が起動されたディレクトリ) です。

オプションで、ダウンロード・ファイルを作成したときと同じ dbmlsync コマンド・ラインで、-be オプションを使用してリモート・データベースで検証できる文字列を指定したり、-bg オプションを使用して新しいリモート・データベースのダウンロード・ファイルを作成したりできます。

参照

- 「Mobile Link ファイルベースのダウンロード」 『Mobile Link - サーバ管理』
- 「-ba オプション」 147 ページ
- 「-be オプション」 149 ページ
- 「-bg オプション」 150 ページ

-be オプション

ダウンロード・ファイルを作成するとき、このオプションはファイルに含まれる追加の文字列を指定します。

構文

dbmlsync -bc "*filename***" -be "***string***"** ...

備考

文字列は、認証や他の目的に使用できます。文字列は、ダウンロード・ファイルが適用されるときに、リモート・データベース上の sp_hook_dbmlsync_validate_download_file ストアド・プロシージャに渡されます。

参照

- 「sp_hook_dbmlsync_validate_download_file」 316 ページ
- 「Mobile Link ファイルベースのダウンロード」 『Mobile Link - サーバ管理』
- 「-bc オプション」 148 ページ
- 「-ba オプション」 147 ページ

-bg オプション

ダウンロード・ファイルを作成するとき、このオプションはまだ同期していないリモート・データベースで使用できるファイルを作成します。

構文

dbmlsync -bc "*filename***" -bg** ...

備考

-bg オプションを使用すると、ダウンロード・ファイルによってリモート・データベースの世代番号が更新されます。

このオプションを使用すると、同期していないリモート・データベースに適用できるダウンロード・ファイルを構築できます。このオプションを使用しない場合は、同期を行ってからダウンロード・ファイルを適用する必要があります。

-bg オプションで構築したダウンロード・ファイルは、スナップショット・ダウンロードです。新しいリモートの最終ダウンロード・タイムスタンプは、デフォルトでは 1900 年 1 月 1 日になっており、これはダウンロード・ファイル内の最終ダウンロード・タイムスタンプより前となるため、タイムスタンプ・ベースのダウンロードは同期していないリモート・データベースと連携しません。タイムスタンプ・ベースでファイル・ベースのダウンロードが動作するには、ダウンロード・ファイル内の最終ダウンロード・タイムスタンプがリモートと同じか、それより前である必要があります。

このオプションは、世代番号による機能を回避するため、そのような機能に依存するシステムの場合は、すでに同期されたリモート・データベースに -bg ダウンロード・ファイルを適用しないでください。

参照

- 「Mobile Link ファイルベースのダウンロード」 『Mobile Link - サーバ管理』
- 「-ba オプション」 147 ページ
- 「-bc オプション」 148 ページ
- 「Mobile Link の世代番号」 『Mobile Link - サーバ管理』
- 「新しいリモートの同期」 『Mobile Link - サーバ管理』

-c オプション

リモート・データベースの接続パラメータを指定します。

構文

dbmlsync -c "*connection-string***"** ...

備考

接続文字列では、DBA や REMOTE DBA 権限を使用して SQL Anywhere リモート・データベースに接続するための dbmlsync パーミッションを指定します。この場合は、REMOTE DBA 権限を持つユーザ ID を使用することをおすすめします。

keyword=value の形式で、複数のパラメータをセミコロンで区切って接続文字列を指定します。いずれかのパラメータ名にスペースが含まれる場合は、接続文字列を二重引用符で囲んでください。

-c を指定しないと、[DBMLSync 設定] ウィンドウが表示されます。この接続ウィンドウのフィールドで、残りのコマンド・ライン・オプションを指定できます。

SQL Anywhere データベースに接続するための接続パラメータの完全なリストについては、「接続パラメータ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

-d オプション

リモート・データベースに対する競合ロックを削除します。

構文

dbmlsync -d ...

備考

同期中は、LockTables 拡張オプションを OFF に設定しないかぎり、同期するパブリケーションに関するすべてのテーブルがロックされて他のプロセスによる変更が加えられません。別の接続でこれらのいずれかのテーブルにロックがあると、同期は失敗したり遅延したりする可能性があります。このオプションを指定すると、SQL Anywhere は競合ロックを保持するリモート・データベースへの他の接続をすべて強制的に削除するため、同期はただちに実行されます。

参照

- 「同期中の同時実行性」 121 ページ

-dc オプション

以前に失敗したダウンロードを再起動します。

構文

dbmlsync -dc ...

備考

デフォルトでは、ダウンロード中に Mobile Link で障害が発生すると、ダウンロード・データはリモート・データベースに適用されません。ただし、受信したダウンロードの一部がリモート・デバイスのテンポラリ・ファイルに保存されているため、次に dbmlsync を起動するときに -dc オプションを指定すると、短時間でダウンロードを完了できます。-dc オプションを指定すると、dbmlsync はダウンロードを再起動し、前のダウンロードで受信しなかった部分をダウンロードします。残りのデータをダウンロードできる場合は、完全なダウンロードがリモート・データベースに適用されます。

-dc オプションを使用した場合、アップロードされる新しいデータがあると、再起動可能なダウンロードは失敗します。

また、ContinueDownload 拡張オプションや sp_hook_dbmlsync_end フックを使用して、失敗したダウンロードを再起動することもできます。

参照

- 「失敗したダウンロードの再開」『Mobile Link - サーバ管理』
- 「ContinueDownload (cd) 拡張オプション」 201 ページ
- 「sp_hook_dbmlsync_end」 285 ページ
- 「DownloadReadSize (drs) 拡張オプション」 205 ページ

-dl オプション

メッセージを dbmlsync のメッセージ・ウィンドウまたはコマンド・プロンプトに表示し、メッセージ・ログ・ファイルに書き込みます。

構文

dbmlsync -dl ...

備考

通常、ファイルに出力のログを取るときは、dbmlsync ウィンドウに書き込まれるよりも多くのメッセージがログ・ファイルに書き込まれます。このオプションを使用すると、dbmlsync は通常はファイルにのみ書き込まれる情報をウィンドウにも出力します。このオプションを使用すると、同期の速度に影響する場合があります。

-do オプション

オフライン・トランザクション・ログのスキャンを無効にします。

構文

dbmlsync -do ...

備考

1 つのディレクトリに複数のデータベースのトランザクション・ログ・ファイルが格納されている場合、これらのいずれかのデータベースのオフライン・トランザクション・ログ・ファイルがなくても、dbmlsync はどのデータベースからの同期もできない可能性があります。dbmlsync は、-do オプションが指定された場合、オフライン・トランザクション・ログをスキャンしません。したがって、あるデータベースが他のすべてのデータベースとともに 1 つのディレクトリに格納されており、そのデータベースにオフライン・トランザクション・ログ・ファイルがない場合、-do が指定された dbmlsync は、そのデータベースからの同期を実行できる必要があります。

このオプションを使用しても、オフライン・トランザクション・ログが必要な場合、dbmlsync は同期できません。

-x オプションとともに使用することはできません。

-drs オプション

再起動可能なダウンロードについて、通信障害の後に再送する必要があるデータの最大値を指定します。

構文

dbmlsync -drs *bytes* ...

備考

-drs オプションは、再起動可能なダウンロードを実行するときにだけ有用なダウンロードの読み込みサイズを指定します。

dbmlsync は、チャンク単位でダウンロードを読み込みます。ダウンロードの読み込みサイズは、このチャンクのサイズを定義します。通信エラーが発生すると、処理中のチャンク全体が失われます。エラーが発生した時点によって、失われるバイト数は 0 ～ ダウンロードの読み込みサイズ -1 のいずれかになります。たとえば、DownloadReadSize が 100 バイトで、497 バイトを読み込んだ後でエラーが発生した場合は、読み込んだ最後の 97 バイトが失われます。このようにして失われたバイト数は、ダウンロードが再起動されたときに再送信されます。

通常は、ダウンロード読み込みサイズの値を大きくすると、正常な同期でのパフォーマンスが向上しますが、エラーが発生したときに再送信されるデータが多くなります。

このオプションの一般的な用途は、通信の信頼性が低いときにデフォルトのサイズを減らすことです。

デフォルトは **32767** です。このオプションを 32767 より大きな値に設定すると、32767 が使用されます。

また、DownloadReadSize 拡張オプションを使用して、ダウンロードの読み込みサイズを指定することもできます。

参照

- 「DownloadReadSize (drs) 拡張オプション」 205 ページ
- 「失敗したダウンロードの再開」 『Mobile Link - サーバ管理』
- 「ContinueDownload (cd) 拡張オプション」 201 ページ
- 「sp_hook_dbmlsync_end」 285 ページ
- 「-dc オプション」 153 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -drs 100
```


-ds オプション

ダウンロード専用同期を実行します。

構文

dbmlsync -ds ...

備考

ダウンロード専用同期が発生する場合、dbmlsync はローの操作またはデータをアップロードしません。しかし、スキーマと進行オフセットについての情報はアップロードします。

さらに、dbmlsync は、ダウンロード専用同期中に、リモートでの変更が上書きされないようにします。これを実行するには、ログをスキャンし、アップロードされるのを待機している操作に関連するローを検出します。これらのローのいずれかがダウンロードによって修正されると、ダウンロードはロールバックされ、同期は失敗します。この理由で同期が失敗した場合は、問題を訂正するために完全な同期を行う必要があります。

ダウンロード専用同期で同期されたリモートがある場合、ダウンロード専用同期がスキャンするログの量を減らすために、定期的に完全な双方向同期を行ってください。そうしないと、ダウンロード専用同期が完了するのに次第に時間がかかるようになります。

-ds を使用すると、ConflictRetries 拡張オプションが無視され、dbmlsync はダウンロード専用同期をリトライしません。ダウンロード専用同期は、失敗した場合、通常の同期が行われるまで失敗し続けます。

ダウンロード専用同期に定義する必要があるスクリプトのリストについては、「必要なスクリプト」 『Mobile Link - サーバ管理』を参照してください。

参照

- 「アップロード専用の同期とダウンロード専用の同期」 『Mobile Link - サーバ管理』
- 「DownloadOnly (ds) 拡張オプション」 204 ページ
- 「ダウンロード専用のパブリケーション」 109 ページ

-e オプション

拡張オプションを指定します。

構文

dbmlsync -e *extended-option*=*value*; ...

extended-option:
adr cd cr ctp dbs dir drs ds eh el ft hrt inc isc lt mem mn mp p pp sa sc sch scn st sv toc tor uo v vn vm vo vr vs vu

パラメータ

拡張オプションは、長形式または省略形で指定できます。

「Mobile Link SQL Anywhere クライアントの拡張オプション」 193 ページを参照してください。

備考

コマンド・ラインで -e オプションを使用して指定したオプションは、同じコマンド・ラインで要求したすべての同期に適用されます。たとえば、次のコマンド・ラインでは、拡張オプション sv=test は pub1 と pub2 の両方の同期に適用されます。

```
dbmlsync -e "sv=test" -n pub1 -n pub2
```

拡張オプションは、dbmlsync メッセージ・ログと SYSSYNC システム・ビューで確認できます。

単一のアップロードに拡張オプションを指定するには、-eu オプションを使用します。

参照

- 「-eu オプション」 162 ページ
- 「SYSSYNC システム・ビュー」 『SQL Anywhere サーバ - SQL リファレンス』
- 「sp_hook_dbmlsync_set_extended_options」 304 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を起動するときの拡張オプションの設定方法を示します。

```
dbmlsync -e "adr=host=localhost;dir=c:¥db¥logs"...
```


-eh オプション

フック関数で発生したエラーを無視します。

構文

dbmlsync -eh ...

-ek オプション

強力に暗号化されたデータベースの暗号化キーをコマンド・ラインで直接指定できます。

構文

dbmlsync -ek *key* ...

備考

強力に暗号化されたデータベースを扱う場合には、データベースやトランザクション・ログ (オフライン・トランザクションなど) を使用するのに、常に暗号化キーを使用する必要があります。強力な暗号化が適用されたデータベースの場合、-ek または -ep のどちらかを指定します。両方同時には指定できません。強力に暗号化されたデータベースでは、キーを指定しないとコマンドが失敗します。

-ep オプション

暗号化キーを入力するよう要求します。

構文

dbmlsync -ep ...

備考

このオプションを指定すると、暗号化キーを入力するためのウィンドウが表示されます。クリア・テキストでは暗号化キーを見ることができないようにすることで、高いセキュリティが得られます。強力な暗号化が適用されたデータベースの場合、-ek または -ep のどちらかを指定します。両方同時には指定できません。強力に暗号化されたデータベースでは、キーを指定しないとコマンドが失敗します。

-eu オプション

拡張アップロード・オプションを指定します。

構文

dbmlsync -n *publication-name* **-eu** *keyword=value*;...

備考

コマンド・ラインで -eu オプションを使用して指定した拡張オプションは、その直前の -n オプションで指定される同期のみに適用されます。たとえば、次のコマンド・ラインでは、拡張オプション sv=test は pub2 の同期のみに適用されます。

```
dbmlsync -n pub1 -n pub2 -eu "sv=test"
```

拡張オプションが複数設定された場合の処理方法については、「Mobile Link SQL Anywhere クライアントの拡張オプション」 193 ページを参照してください。

拡張オプションの完全なリストは、「Mobile Link SQL Anywhere クライアントの拡張オプション」 193 ページを参照してください。

-is オプション

スケジュールの指示を無視して、直ちに同期を行います。

構文

dbmlsync -is ...

備考

同期をスケジュールする拡張オプションを無視します。

スケジュールの詳細については、「同期のスケジュール」 128 ページを参照してください。

-k オプション (旧式)

同期が終了したときに dbmlsync を停止します。このオプションは推奨されなくなりました。代わりに -qc を使用してください。

構文

dbmlsync -k ...

参照

- 「-qc オプション」 178 ページ

-l オプション

使用可能な拡張オプションをリストします。

構文

dbmlsync -l ...

備考

dbmlsync コマンド・ラインと一緒に使用すると、使用可能な拡張オプションがすべて表示されます。

参照

- 「Mobile Link SQL Anywhere クライアントの拡張オプション」 193 ページ

-mn オプション

同期する Mobile Link ユーザの新しいパスワードを指定します。

構文

dbmlsync -mn *password ...*

備考

Mobile Link ユーザのパスワードを変更します。

詳細については、「Mobile Link ユーザ」 9 ページを参照してください。

参照

- 「MobiLinkPwd (mp) 拡張オプション」 218 ページ
- 「NewMobiLinkPwd (mn) 拡張オプション」 219 ページ
- 「-mp オプション」 167 ページ

-mp オプション

同期する Mobile Link ユーザのパスワードを指定します。

構文

dbmlsync -mp *password* ...

備考

Mobile Link ユーザ認証用のパスワードを指定します。

詳細については、「Mobile Link ユーザ」 9 ページを参照してください。

参照

- 「MobiLinkPwd (mp) 拡張オプション」 218 ページ
- 「NewMobiLinkPwd (mn) 拡張オプション」 219 ページ
- 「-mn オプション」 166 ページ

-n オプション

同期させるパブリケーションを指定します。

構文

dbmlsync -n *pubname* ...

備考

同期パブリケーションの名前です。

-n オプションを複数指定すると、複数の同期パブリケーションを同期できます。ただし、1 つの同期プロファイルでパブリケーションを 2 回以上指定することはできません。

-n を使用して複数のパブリケーションを同期するには、次の 2 つの方法があります。

- -n pub1,pub2,pub3 と指定して、1 つのアップロードで pub1、pub2、pub3 をアップロードし、その後に 1 つのダウンロードを行う。

  この方法では、各パブリケーションで拡張オプションを設定した場合、リスト内の最初のパブリケーションで設定したオプションのみが使用されます。2 番目以降のパブリケーションで設定した拡張オプションは無視されます。

- -n pub1 -n pub2 -n pub3 と指定して、pub1、pub2、pub3 の 3 つを別個の逐次同期で同期する。

  -n pub1 -n pub2 を指定するなど、連続した同期が非常に速く行われると、サーバがまだ前の同期を処理しているときに、dbmlsync が同期の処理を開始する場合があります。この場合、2 番目の同期は、同時同期ができないことを示すエラーで失敗します。この状況が発生した場合は、sp_hook_dbmlsync_delay ストアド・プロシージャを定義して、各同期の前に遅延を作成できます。通常は数秒から 1 分の遅延で十分です。

  詳細については、「sp_hook_dbmlsync_delay」 265 ページを参照してください。

-o オプション

出力を dbmlsync メッセージ・ログ・ファイルに送信します。

構文

dbmlsync -o *filename* ...

備考

出力をログ・ファイルに追加します。デフォルトでは、画面に出力が表示されます。

参照

- 「-os オプション」 170 ページ
- 「-ot オプション」 171 ページ

-os オプション

dbmlsync メッセージ・ログ・ファイルの最大サイズを指定します (このサイズに達するとログの名前が変更されます)。

構文

dbmlsync -os *size* [**K** | **M** | **G**]...

備考

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、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。デフォルトでは、サイズは無制限となります。最小のサイズ制限は 10K です。

dbmlsync ユーティリティは、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・メッセージのファイル・サイズが指定したサイズを超えた場合、dbmlsync ユーティリティは出力ファイルの名前を *yymmddxx*.dbr に変更します。*yymmdd* は年月日を表し、*xx* は AA から ZZ までの連続した英字を表します。

このオプションを使用すると、古いログ・ファイルを手動で削除してディスク領域を解放できます。

参照

- 「-o オプション」 169 ページ
- 「-ot オプション」 171 ページ

-ot オプション

メッセージ・ログ・ファイルの内容を削除してから、このファイルに出力メッセージのログを取ります。

構文

dbmlsync -ot *logfile* ...

備考

機能は -o オプションと同じですが、dbmlsync の起動時にメッセージ・ログ・ファイルの内容が削除されてからメッセージが書き込まれます。

参照

- 「-o オプション」 169 ページ
- 「-os オプション」 170 ページ

-p オプション

ログスキャンのポーリングを無効にします。

構文

dbmlsync -p ...

備考

アップロードを構築するには、dbmlsync はトランザクション・ログをスキャンする必要があります。通常、これは同期直前に行います。しかし、同期がスケジュールされている場合、または sp_hook_dbmlsync_delay フックが使用されている場合、dbmlsync はデフォルトでは同期の前に発生する一時停止でログをスキャンします。同期が開始されるとき、ログはすでに少なくとも一部がスキャンされているため、この動作はより効率的です。このデフォルトの動作は、ログスキャンのポーリングと呼ばれます。

ログスキャンのポーリングはデフォルトではオンになっていますが、スケジュール・オプションを使用して同期がスケジュールされているとき、または sp_hook_dbmlsync_delay フックが使用されているときしか効果がありません。ポーリングが有効な場合は、設定された間隔で実行されます。これはデフォルトでは 1 分ですが、dbmlsync -pp オプションで変更できます。

このオプションは、拡張オプションの DisablePolling=on とまったく同じです。

参照

- 「DisablePolling (p) 拡張オプション」 202 ページ
- 「PollingPeriod (pp) 拡張オプション」 222 ページ
- 「-pp オプション」 176 ページ

-pc オプション

同期と同期の間で、Mobile Link サーバへの永続的な接続を維持します。

構文

dbmlsync -pc+ ...

備考

このオプションを指定すると、dbmlsync は通常どおり Mobile Link サーバに接続しますが、以降の同期で使用できるようにその接続を開いたままにします。永続的な接続は、次のいずれかが発生すると閉じます。

- エラーが発生し、同期に失敗した場合。
- 活性チェックのタイムアウトが発生した場合。

  「timeout」 71 ページを参照してください。
- 同期が、異なる通信タイプまたはアドレスで開始された場合。つまり、設定が異なったり (別のホストが指定された場合など)、異なる方法で指定された場合 (同じホストとポートが指定されたが、順序が異なる場合など) です。

永続的な接続が閉じている場合は、新しい接続 (永続的) が開きます。

このオプションが最も役に立つのは、クライアントが高い頻度で同期するために、サーバへの接続確立コストが高くなっている場合です。

デフォルトでは、永続的な接続は維持されません。

-pd オプション

Windows Mobile 用の指定された DLL をプリロードします。

構文

dbmlsync -pd *dllname*;...

備考

dbmlsync を Windows Mobile で実行するときに暗号化された通信ストリームを使用している場合は、-pd オプションを使用して起動時に適切な DLL がロードされるようにしてください。そうしなかった場合、dbmlsync では必要になるまで DLL をロードしません。Windows Mobile ではリソースが制限されるため、後で DLL をロードするとエラーが発生しやすくなります。

次に、各通信プロトコル用にロードする必要がある DLL を示します。

プロトコル	DLL
ECC	mlcecc10.dll
RSA	mlcrsa10.dll
FIPS	mlcrsafips10.dll

複数の DLL は、セミコロンで区切ったリストで指定します。次に例を示します。

```
-pd mlcrsafips10.dll;mlcrsa10.dll
```


-pi オプション

Mobile Link サーバを ping します。

構文

dbmlsync -pi -c *connection_string* ...

備考

-pi を使用すると、dbmlsync はリモート・データベースに接続し、Mobile Link サーバへの接続に必要な情報を取り出し、サーバに接続し、指定した Mobile Link ユーザを認証します。Mobile Link サーバは、ping を受信すると、統合データベースに接続し、ユーザを認証し、ユーザ認証ステータスと値をクライアントに送信します。Mobile Link サーバがコマンド・ライン・オプション -zu+ を指定して実行されていると、Mobile Link ユーザ名が ml_user システム・テーブルに見つからない場合は Mobile Link サーバがユーザを ml_user Mobile Link システム・テーブルに追加します。

適切に接続をテストするには、dbmlsync との同期に使用するすべての同期オプションと一緒に -pi オプションを使用します。-pi を使用すると、dbmlsync は同期を実行しません。

ping に成功した場合、Mobile Link サーバは情報メッセージを発行します。ping に失敗した場合は、エラー・メッセージを発行します。

-pi を指定して dbmlsync を起動した場合、Mobile Link サーバが実行できるのは、次のスクリプト (存在する場合) だけです。

- begin_connection
- authenticate_user
- authenticate_user_hashed
- authenticate_parameters
- end_connection

-pp オプション

ログスキャンの頻度を指定します。

構文

dbmlsync -pp *number* [**h** | **m** | **s**]...

備考

アップロードを構築するには、dbmlsync はトランザクション・ログをスキャンする必要があります。通常、これは同期直前に行います。しかし、同期がスケジュールされている場合、または sp_hook_dbmlsync_delay フックが使用されている場合、dbmlsync はデフォルトでは同期の前に発生する一時停止でログをスキャンします。同期が開始されるとき、ログはすでに少なくとも一部がスキャンされているため、この動作はより効率的です。このデフォルトの動作は、ログスキャンのポーリングと呼ばれます。

このオプションは、ログスキャンの間隔を指定します。サフィックス s、m、h、d を使用して、それぞれ秒、分、時間、日を指定します。デフォルトは **1** 分です。サフィックスを指定しない場合、デフォルトの時間単位は分です。

参照

- 「PollingPeriod (pp) 拡張オプション」 222 ページ
- 「DisablePolling (p) 拡張オプション」 202 ページ
- 「-p オプション」 172 ページ

-q オプション

Mobile Link 同期クライアントを最小化ウィンドウで起動します。

構文

dbmlsync -q ...

-qc オプション

同期が終了したときに dbmlsync を停止します。

構文

dbmlsync -qc ...

備考

このオプションを使用すると、同期が成功するか、-o オプションまたは -ot オプションを使用してメッセージ・ログ・ファイルが指定されている場合に、同期の完了後に dbmlsync を終了します。

参照

- 「-o オプション」 169 ページ
- 「-ot オプション」 171 ページ

-r オプション

リモート・データベースと統合データベースのオフセットが一致しないとき、リモート・オフセットを使用するように指定します。

-rb オプションは、リモート・オフセットが統合オフセットよりも小さい場合 (リモート・データベースがバックアップからリストアされたときなど)、リモート・オフセットを使用することを示します。-r オプションは下位互換のために提供されており、-rb と同じです。-ra オプションは、リモート・オフセットが統合オフセットよりも大きい場合、リモート・オフセットを使用することを示します。このオプションは、非常にまれな環境だけのために提供されており、データ損失の原因となる可能性があります。

構文

dbmlsync { -r | -ra | -rb } ...

備考

進行オフセットの詳細については、「進行オフセット」 103 ページを参照してください。

-rb　リモート・データベースがバックアップからリストアされると、デフォルトの動作がデータ損失の原因となることがあります。この場合、リモート・データベースをリストアした後、初めて dbmlsync を実行するときに -rb を指定します。-rb を使用すると、リモート・データベースに記録されたオフセットが統合データベースから取得したオフセットよりも小さい場合、リモート・データベースに記録されているオフセットからアップロードが継続されます。-rb を使用し、リモートのオフセットが統合データベースからのオフセットより小さい場合、エラーがレポートされ、同期がアボートされます。

-rb オプションでは、すでにアップロードされたデータがアップロードされることがあります。これにより統合データベースで競合が起こる可能性があり、これは適切な競合解決スクリプトを使用して処理する必要があります。

-ra　-ra オプションは、非常にまれな場合にのみ使用してください。-ra を使用すると、リモートのオフセットが統合データベースから取得したオフセットよりも大きい場合、アップロードはリモート・データベースから取得したオフセットからリトライされます。-ra を使用し、リモートのオフセットが統合データベースからのオフセットより大きくない場合、エラーがレポートされ、同期がアボートされます。

-ra オプションの使用には注意してください。統合データベースをリストアした結果、オフセットが一致しなければ、2 つのオフセット間の差のうちリモート・データベース内で発生した変更が失われます。-ra オプションは、統合データベースがバックアップからリストアされ、リモート・データベースのトランザクション・ログがリモートのオフセットと同じポイントでトランケートされた場合に役立ちます。この場合、リモート・データベースからアップロードされたすべてのデータは、統合オフセットのポイントからリモート・オフセットのポイントまで失われます。

-sc オプション

dbmlsync が各同期の前にスキーマ情報を再ロードするように指定します。

構文

dbmlsync -sc...

備考

バージョン 9.0 以前では、dbmlsync は各同期の前にデータベースからスキーマ情報を再ロードしていました。再ロードされた情報には、外部キー関係、パブリケーションの定義、データベースに格納された拡張オプション、データベースの設定に関する情報が含まれていました。このような情報のロードには時間がかかります。また、ほとんどの場合、同期と同期の間で情報が変更されることはありません。

バージョン 9.0 以降では、dbmlsync はデフォルトで起動時にのみスキーマ情報をロードします。各同期の前にこの情報をロードする場合は、-sc を指定します。

-sp オプション

-sp を使用すると、指定された同期プロファイルにあるオプションが、その同期に対してコマンド・ラインで指定されたオプションに追加されます。

構文

dbmlsync -sp *sync profile*

備考

コマンド・ラインと同期プロファイルで同等のオプションが指定された場合、コマンド・ラインにあるオプションが、プロファイルにあるオプションよりも優先されます。

参照

- 「CREATE SYNCHRONIZATION PROFILE 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「ALTER SYNCHRONIZATION PROFILE 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「DROP SYNCHRONIZATION PROFILE 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』

-tu オプション

リモート・データベースの各トランザクションを、1 つの同期内で独立したトランザクションとしてアップロードするように指定します。

構文

dbmlsync -tu ...

備考

-tu オプションを使用するときは、「トランザクション・アップロード」を作成します。こうすると、dbmlsync はリモート・データベースの各トランザクションを別個のトランザクションとしてアップロードします。Mobile Link サーバは各トランザクションを受信したときに別々に適用およびコミットします。

-tu オプションを使用すると、リモート・データベースのトランザクションの順序は、常に統合データベースで保持されます。ただし、トランザクションでの操作の順序は保持されないことがあります。これには、以下の 2 つの理由があります。

- Mobile Link は、常に外部キー関係に基づいて更新を適用します。たとえば、子と親のテーブルでデータが変更されると、Mobile Link はデータの挿入を親のテーブル、子のテーブルの順に行いますが、データの削除は子のテーブル、親のテーブルの順に行います。リモートの操作がこの順序に従っていない場合は、統合データベースでは操作の順序が異なります。
- トランザクション内の操作は結合されます。つまり、1 つのトランザクションで同じローを 3 回変更しても、最後の形式のローのみがアップロードされます。

トランザクション・アップロードが中断された場合、送信されなかったデータは、次の同期で送信されます。ほとんどの場合、正常に完了しなかったトランザクションだけが、この時点で送信されます。また、サブスクリプションの最初の同期中にアップロードが失敗した場合などは、dbmlsync はすべてのトランザクションを再送します。

-tu オプションを使用しないと、Mobile Link はリモート・データベースでのすべての変更を結合し、アップロードの 1 つのトランザクションにします。つまり、同期と同期の間に同じローを 3 回変更しても、リモート・トランザクションの数に関係なく、最後の形式のローのみがアップロードされます。このデフォルトの動作は、効率がよく、多くの状況に最適です。

ただし、特定の状況では、統合データベースでリモート・トランザクションを保持したい場合があります。たとえば、リモート・データベースでトランザクションが発生したときに、そのトランザクションに基づいてアクションを行うトリガを統合データベースで定義したいことがあります。

さらに、アップロードを小さいトランザクションに分割することには利点があります。多くの統合データベースは、小さなトランザクションを対象として最適化されているため、巨大なトランザクションは効率的でなく、多数の競合が発生することがあります。また、-tu オプションを使用すると、アップロード中に通信エラーが発生してもアップロード全体を失わずに済むことがあります。-tu オプションを使用している場合にアップロード・エラーが発生しても、正常にアップロードされたすべてのトランザクションが適用されます。

-tu オプションを指定すると、Mobile Link は SQL Remote とほぼ同じように動作します。主な相違点は、SQL Remote がすべての変更を発生順にリモート・データベースにレプリケートし、結

合を行わないことです。このように動作させるには、リモート・データベースで各データベース操作の後にコミットしてください。

Increment 拡張オプションやスクリプト化されたアップロードに -tu を使用することはできません。

参照

- 「-tx オプション」 『Mobile Link - サーバ管理』
- 「自己参照テーブルからのデータのアップロード」 『Mobile Link - サーバ管理』

-u オプション

Mobile Link ユーザ名を指定します。

構文

dbmlsync -u *ml_username* ...

備考

dbmlsync コマンド・ラインでユーザを 1 人指定できます。この場合、*ml_username* は、処理するサブスクリプションに対応する CREATE SYNCHRONIZATION SUBSCRIPTION 文の FOR 句に使用されているユーザ名です。

このオプションを -n *publication* と併用して、dbmlsync が操作するサブスクリプションを識別します。各サブスクリプションは、*ml_username*, *publication* の組み合わせでユニークに識別されます。

コマンド・ラインで指定できるユーザ名は 1 つだけです。1 回の処理で同期するサブスクリプションはすべて、同じユーザと関連していなければなりません。-n オプションを使用してコマンド・ラインで指定した各パブリケーションにサブスクリプションが 1 つしかない場合は、-u オプションを省略できます。

-ui オプション

X-Window Server がサポートされている Linux で、使用可能な表示がない場合にシェル・モードで dbmlsync を起動します。

構文

mlsrv11 -c "*connection-string***" -ui** ...

備考

このオプションを使用すると、X-Window で dbmlsync の起動が試みられます。それが失敗した場合は、シェル・モードで起動されます。

-ui を指定すると、dbmlsync は使用可能な表示を見つけようとします。X-Window Server が実行されていなかったなどの理由で、使用可能な表示が見つからなかった場合は、dbmlsync はシェル・モードで起動されます。

-uo オプション

同期がアップロードだけを含むように指定します。

構文

dbmlsync -uo...

備考

アップロード専用の同期中に、dbmlsync は通常の完全な同期とまったく同じように、Mobile Link へのアップロードを準備し送信します。しかし、ダウンロードを返信する代わりに、Mobile Link はアップロードのコミットが成功したかどうかを示す確認だけを送信します。

ダウンロード専用同期に定義する必要があるスクリプトのリストについては、「必要なスクリプト」『Mobile Link - サーバ管理』を参照してください。

参照

- 「アップロード専用の同期とダウンロード専用の同期」『Mobile Link - サーバ管理』
- 「DownloadOnly (ds) 拡張オプション」 204 ページ
- 「UploadOnly (uo) 拡張オプション」 232 ページ

-urc オプション

同期でアップロードされるロー数の推定値を指定します。

構文

dbmlsync -urc *row-estimate ...*

備考

パフォーマンスを改善するために、同期でアップロードするロー数の推定値を指定できます。この設定は、多数のローをアップロードするときに特に便利です。推定値が大きいほど、アップロードが高速になりますが、多くのメモリを消費します。

指定した推定値に関係なく、同期は正しく続行されます。

参照

- 「Memory (mem) 拡張オプション」 216 ページ
- 「大規模なアップロードのロー数の推定」 『Mobile Link - サーバ管理』

-ux オプション

Linux で、メッセージを表示する、dbmlsync のメッセージ・ウィンドウを開きます。

構文

dbmlsync -ux...

備考

-ux が指定されている場合、dbmlsync は使用可能な表示を見つけます。たとえば、DISPLAY 環境変数が設定されていなかったり、X-Window Server が実行されていなかったりしたために、使用可能な表示が見つからなかった場合、dbmlsync は起動できません。

クワイエット・モードで dbmlsync のメッセージ・ウィンドウを実行するには、-q を使用します。

Windows では、dbmlsync のメッセージ・ウィンドウが自動的に表示されます。

参照

- 「-q オプション」 177 ページ

-v オプション

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。冗長レベルを上げ過ぎるとパフォーマンスに影響する可能性があるので、通常は冗長レベルを上げるのは開発段階だけにしてください。

構文

dbmlsync -v [*levels*] ...

備考

-v オプションはメッセージ・ログ・ファイルと同期ウィンドウに影響します。dbmlsync コマンド・ラインで -o または -ot を指定すると、メッセージ・ログが記録されるだけです。

-v を単独で指定すると、少量の情報のログが取られます。

levels の値は次のとおりです。たとえば -vnrsu や -v+cp などのように、次のオプションを一度に 1 つまたは複数指定できます。

- **+** c と p 以外のすべてのログ・オプションをオンにする
- **c** ログ内の接続文字列を公開する
- **p** ログ内のパスワードを公開する
- **n** アップロードとダウンロードされたロー数のログを取る
- **o** 指定したコマンド・ライン・オプションと拡張オプションに関する情報のログを取る
- **r** アップロードとダウンロードされたローの値のログを取る
- **s** フック・スクリプトに関連するメッセージのログを取る
- **u** アップロード関する情報のログを取る

-v オプションと同様の機能を持つ拡張オプションがあります。-v オプションと拡張オプションの両方を指定して、競合が発生した場合は、-v オプションが拡張オプションよりも優先されます。競合が発生しない場合は、冗長ログ・オプションが追加され、指定したすべてのオプションが使用されます。拡張オプションでロギングの冗長性を設定しても、ロギングは直ちに有効にならないため、起動情報のログは取られません。最初の同期が行われる時点では、-v オプションを指定した場合と拡張オプションを指定した場合のロギングの動作は同じです。

参照

- 「Verbose (v) 拡張オプション」 233 ページ
- 「VerboseHooks (vs) 拡張オプション」 234 ページ
- 「VerboseMin (vm) 拡張オプション」 235 ページ
- 「VerboseOptions (vo) 拡張オプション」 236 ページ
- 「VerboseRowCounts (vn) 拡張オプション」 237 ページ
- 「VerboseRowValues (vr) 拡張オプション」 238 ページ
- 「-o オプション」 169 ページ
- 「-ot オプション」 171 ページ

-wc オプション

ウィンドウ・クラス名を指定します。

構文

dbmlsync -wc *class-name* ...

備考

このオプションは、スケジュールが有効になっているときや、サーバによって開始される同期を使用しているときなどに、停止モードの dbmlsync をウェイクアップするのに使用可能なウィンドウ・クラス名を指定します。

また、このウィンドウ・クラス名によって、ActiveSync 同期用のアプリケーションが識別されます。ActiveSync 同期に使用するアプリケーションの登録時に、クラス名を指定してください。

このオプションが適用されるのは、Windows だけです。

参照

- 「ActiveSync 用 SQL Anywhere クライアントの登録」 126 ページ
- 「ActiveSync 同期の使用」 124 ページ
- 「Schedule (sch) 拡張オプション」 223 ページに示されている INFINITE キーワード
- 「同期のスケジュール」 128 ページ

例

```
dbmlsync -wc dbmlsync_$message_end...
```


-x オプション

出力メッセージがスキャンされた後、トランザクション・ログの名前を変更し、再起動します。

構文

dbmlsync -x [*size* [**K** | **M** | **G**] ...

備考

オプションの *size* は、トランザクション・ログが指定されたサイズより大きい場合にのみ、名前が変更されることを意味します。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。デフォルトのサイズは 0 です。

場合によっては、リモート・データベースのバックアップが実行されたり、データベース・サーバを停止するときにトランザクション・ログの名前を変更する代わりに、統合データベースにデータが同期されます。

リモート・データベース側でバックアップを定期的に実行しないと、トランザクション・ログが大きくなっていきます。トランザクション・ログのサイズを制御するには、-x オプションを使用する代わりに、SQL Anywhere のイベント・ハンドラを使用する方法があります。

参照

- 「スケジュールとイベントの使用によるタスクの自動化」『SQL Anywhere サーバ - データベース管理』
- 「delete_old_logs オプション [Mobile Link クライアント] [SQL Remote] [Replication Agent]」『SQL Anywhere サーバ - データベース管理』
- 「CREATE EVENT 文」『SQL Anywhere サーバ - SQL リファレンス』

Mobile Link SQL Anywhere クライアントの拡張オプション

目次

dbmlsync 拡張オプションの概要

拡張オプションは、dbmlsync コマンド・ライン上で -e オプションまたは -eu オプションを使用して指定するか、データベースに格納できます。拡張オプションをデータベースに格納するには、Sybase Central を使用するか、sp_hook_dbmlsync_set_extended_options イベント・フックを使用するか、次のいずれかの文で OPTION 句を使用します。

- CREATE SYNCHRONIZATION SUBSCRIPTION
- ALTER SYNCHRONIZATION SUBSCRIPTION
- CREATE SYNCHRONIZATION USER
- ALTER SYNCHRONIZATION USER
- 同期ユーザを指定しない CREATE SYNCHRONIZATION SUBSCRIPTION (これによって、拡張オプションがパブリケーションに対応付けられる)

優先順位

dbmlsync は、データベースに格納されるオプションとコマンド・ラインで指定されるオプションを結合します。競合するオプションが指定されている場合、dbmlsync は次のようにして競合を解決します。次のリストで先に示す方法で指定されているオプションが、後に示すものよりも優先されます。

1. sp_hook_dbmlsync_set_extended_options イベント・フックで指定されているオプション。
2. 拡張オプションではないコマンド・ラインで指定されたオプション(たとえば、-ds は -e "ds=off" を上書きします。)
3. コマンド・ラインで -eu オプションを使用して指定されているオプション
4. コマンド・ラインで -e オプションを使用して指定されているオプション
5. SQL 文または Sybase Central でサブスクリプションに対して指定されているオプション。**同期モデル展開ウィザード**を使用して Mobile Link モデルを展開すると、拡張オプションが自動的に設定され、サブスクリプションで指定されます。
6. SQL 文または Sybase Central で Mobile Link ユーザに対して指定されているオプション
7. SQL 文または Sybase Central でパブリケーションに対して指定されているオプション

> **注意**
> この優先順位は、前述の SQL 文の TYPE と ADDRESS の各オプションで指定しているような接続パラメータにも適用されます。

拡張オプションは、ログと SYSSYNC システム・ビューで確認できます。

拡張オプションを使用して同期をチューニングする方法については、「dbmlsync 拡張オプションの使用」 119 ページを参照してください。

参照

- 「-e オプション」 158 ページ
- 「-eu オプション」 162 ページ
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「ALTER SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「SYSSYNC システム・ビュー」『SQL Anywhere サーバ - SQL リファレンス』
- 「sp_hook_dbmlsync_set_extended_options」 304 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を起動するときの拡張オプションの設定方法を示します。

```
dbmlsync -e "adr=host=localhost;dir=c:¥db¥logs"...
```

次の SQL 文は、拡張オプションをデータベースに格納する方法を示します。

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

次の dbmlsync コマンド・ラインは、オプションとその構文をリストする使用画面を開きます。

```
dbmlsync -l
```


CommunicationAddress (adr) 拡張オプション

Mobile Link サーバに接続するネットワーク・プロトコル・オプションを指定します。

構文

adr=*protocol-option*; ...

パラメータ

「Mobile Link クライアント・ネットワーク・プロトコル・オプションの一覧」 37 ページを参照してください。

備考

Mobile Link ユーザのすべてのサブスクリプションが、1 つの統合データベースに対してのみ同期されていることを確認する必要があります。複数の統合データベースに同期されていると、データの損失や予期しない動作が発生する場合があります。

リダイレクタを使用している場合は、「Mobile Link クライアントとサーバのリダイレクタ設定」 『Mobile Link - サーバ管理』を参照してください。

このオプションには省略形と長形式があり、**adr** または **CommunicationAddress** を使用できます。

このオプションも、パブリケーション、サブスクリプション、またはユーザを作成、変更する SQL 文を使用して、データベースに格納できます。詳細については、次の項を参照してください。

- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」 『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」 『SQL Anywhere サーバ - SQL リファレンス』

CommunicationType 拡張オプションを使用してネットワーク・プロトコルのタイプを指定します。

「CommunicationType (ctp) 拡張オプション」 199 ページを参照してください。

参照

- 「Mobile Link クライアント・ネットワーク・プロトコル・オプション」 35 ページ
- 「Mobile Link クライアントとサーバのリダイレクタ設定」 『Mobile Link - サーバ管理』

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "adr=host=localhost"
```

コマンド・ラインで複数のネットワーク・プロトコル・オプションを指定するには、それらを一重引用符で囲みます。次に例を示します。

```
dbmlsync -e "adr='host=somehost;port=5001'"
```


データベースに Address または CommunicationType を格納するには、拡張オプションを使用するか、ADDRESS 句を使用できます。次に例を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  ADDRESS 'host=localhost;port=2439'
```


CommunicationType (ctp) 拡張オプション

Mobile Link サーバへの接続に使用するネットワーク・プロトコルの種類を指定します。

構文

ctp=*network-protocol*; ...

備考

network-protocol には、**tcpip**、**tls**、**http**、**https** のいずれかを指定します。デフォルトは **tcpip** です。

Mobile Link ユーザのすべてのサブスクリプションが、1 つの統合データベースに対してのみ同期されていることを確認する必要があります。複数の統合データベースに同期されていると、データの損失や予期しない動作が発生する場合があります。

このオプションには省略形と長形式があり、**ctp** または **CommunicationType** を使用できます。

参照

- 「Mobile Link クライアント／サーバ通信の暗号化」『SQL Anywhere サーバ - データベース管理』
- 「CommunicationAddress (adr) 拡張オプション」 197 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "ctp=https"
```

データベースに CommunicationType を格納するには、拡張オプションを使用するか、TYPE 句を使用できます。次に例を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  TYPE 'tcpip'
```


ConflictRetries (cr) 拡張オプション

競合のためにダウンロードが失敗した場合のリトライの回数を指定します。

構文

cr=*number*; ...

備考

拡張オプション LockTables を OFF に設定すると (同期されているテーブルに対するロックを dbmlsync で取得しないようにする)、アップロードの構築からダウンロードの適用までにデータベースに適用されるオペレーションを実行できます。ダウンロードでも変更されるローに変更が影響する場合、dbmlsync では競合として処理され、ダウンロード・ストリームには変更内容は適用されません。競合が発生すると、dbmlsync は同期処理全体をリトライします。このオプションは、実行するリトライの回数を制御します。

このオプションは、LockTables オプションが OFF (デフォルトは ON) の場合にだけ役に立ちます。

デフォルトは **-1** です (リトライは無制限に続行されます)。

このオプションには省略形と長形式があり、**cr** または **ConflictRetries** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「競合の解決」 『Mobile Link - サーバ管理』

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "cr=5"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION cr='5';
```


ContinueDownload (cd) 拡張オプション

以前に失敗したダウンロードを再起動します。

構文

cd={ ON | OFF }; ...

備考

ダウンロード中に Mobile Link で障害が発生すると、ダウンロード・データはリモート・データベースに適用されません。ただし、受信したダウンロードの一部はリモート・デバイスのテンポラリ・ファイルに格納されるため、後でダウンロードを再起動できます。拡張オプション cd=on を設定すると、dbmlsync はダウンロードを再起動し、前のダウンロードで受信しなかった部分をダウンロードします。残りのデータをダウンロードできる場合は、完全なダウンロードがリモート・データベースに適用されます。

-cd=on を設定した場合、アップロードされる新しいデータがあると、再起動可能なダウンロードは失敗します。

また、SQL Anywhere リモート・データベースの再起動可能なダウンロードは、-dc オプションまたは sp_hook_dbmlsync_end フックを使用して指定することもできます。

このオプションには省略形と長形式があり、**cd** または **ContinueDownload** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「失敗したダウンロードの再開」 『Mobile Link - サーバ管理』
- 「sp_hook_dbmlsync_set_extended_options」 304 ページ
- 「-dc オプション」 153 ページ
- 「sp_hook_dbmlsync_end」 285 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "cd=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION cd='on';
```


DisablePolling (p) 拡張オプション

ログスキャンの自動ポーリングを無効にします。

構文

p={ ON | OFF }; ...

備考

アップロードを構築するには、dbmlsync はトランザクション・ログをスキャンする必要があります。通常、これは同期直前に行います。しかし、同期がスケジュールされている場合、デフォルトでは dbmlsync はスケジュールされた同期の間の時間にログをスキャンします。また、sp_hook_dbmlsync_delay フックが使用されている場合、デフォルトでは dbmlsync は同期直前に生じる一時停止でログをスキャンします。同期が始まるときログはすでに一部がスキャンされているので、この動作はより効率的です。このデフォルトの動作は、ログスキャンのポーリングと呼ばれます。

ログスキャンのポーリングはデフォルトでは on になっていますが、同期がスケジュールされているとき、または sp_hook_dbmlsync_delay フックが使用されているときしか効果がありません。これが有効な場合、ポーリングは決まった間隔で実行されます。dbmlsync はログの最後までスキャンし、ポーリング周期の間待機した後、ログの新しいトランザクションをスキャンします。デフォルトではポーリング周期は 1 分ですが、dbmlsync -pp オプションまたは PollingPeriod 拡張オプションで変更できます。

デフォルトでは、ログスキャンのポーリングを無効にしません **(OFF)**。

このオプションは、**dbmlsync -p** と同じです。

このオプションには省略形と長形式があり、**p** または **DisablePolling** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「PollingPeriod (pp) 拡張オプション」 222 ページ
- 「-p オプション」 172 ページ
- 「-pp オプション」 176 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "p=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION p='on';
```


DownloadBufferSize (dbs) 拡張オプション

ダウンロード・バッファのサイズを指定します。

構文

dbs=*number*[**K** | **M**]; ...

備考

バッファのサイズは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。

このオプションを 0 に設定すると、dbmlsync はダウンロードのバッファを行いません。このオプションに 0 より大きい値を設定すると、Mobile Link サーバによって通信ストリームからダウンロード・ストリーム全体が読み込まれてから、リモート・データベースに適用されます。オプションで指定された領域にダウンロード・ストリームが収まると、ストリーム全体がメモリ内に保持されます。それ以外の場合は、一部がテンポラリ・ファイルに出力されます。

設定が 0 より大きく 4 KB より小さい場合、dbmlsync は 4 KB のバッファ・サイズを使用し、警告を発します。デフォルトは、Windows Mobile では **32k** であり、他のすべてのオペレーティング・システムでは **1m** です。

このオプションには省略形と長形式があり、**dbs** または **DownloadBufferSize** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "dbs=32k"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION dbs='32k';
```


DownloadOnly (ds) 拡張オプション

同期がダウンロード専用であることを指定します。

構文

ds={ ON | OFF }; ...

備考

ダウンロード専用同期が発生する場合、dbmlsync はローの操作またはデータをアップロードしません。しかし、スキーマと進行オフセットについての情報はアップロードします。

さらに、dbmlsync は、ダウンロード専用同期中に、リモートでの変更が上書きされないようにします。これを実行するには、ログをスキャンし、アップロードされるのを待機している操作に関連するローを検出します。これらのローのいずれかがダウンロードによって修正されると、ダウンロードはロールバックされ、同期は失敗します。この理由で同期が失敗した場合は、問題を訂正するために完全な同期を行う必要があります。

ダウンロード専用同期で同期されたリモートがある場合、ダウンロード専用同期がスキャンするログの量を減らすために、定期的に完全な同期を行ってください。そうしないと、ダウンロード専用同期が完了するのに次第に時間がかかるようになります。これが問題になる場合は、ダウンロード専用パブリケーションを使用すると、同期中のログの問題を防ぐことができます。

ダウンロード専用同期に定義する必要があるスクリプトのリストについては、「必要なスクリプト」 『Mobile Link - サーバ管理』を参照してください。

デフォルトは **OFF** です (アップロードとダウンロード両方の完全な同期)。

このオプションには省略形と長形式があり、**ds** または **DownloadOnly** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「-ds オプション」 157 ページ
- 「ダウンロード専用のパブリケーション」 109 ページ
- 「アップロード専用の同期とダウンロード専用の同期」 『Mobile Link - サーバ管理』

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "ds=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION ds='ON';
```


DownloadReadSize (drs) 拡張オプション

再起動可能なダウンロードについて、通信障害の後に再送する必要があるデータの最大値を指定します。

構文

drs=*number*[**K**]; ...

備考

DownloadReadSize オプションが有用なのは、再起動可能なダウンロードを実行するときだけです。

ダウンロードの読み込みサイズは、バイト単位で指定します。キロバイトの単位を指定するには、サフィックス k を使用します。

dbmlsync は、チャンク単位でダウンロードを読み込みます。このチャンクのサイズを定義するのが DownloadReadSize です。通信エラーが発生すると、処理中のチャンク全体が失われます。エラーが発生した時点によって、失われるバイト数は 0 ～ DownloadReadSize -1 のいずれかになります。たとえば、DownloadReadSize が 100 バイトで、497 バイトを読み込んだ後でエラーが発生した場合は、読み込んだ最後の 97 バイトが失われます。このようにして失われたバイト数は、ダウンロードが再起動されたときに再送信されます。

通常は、DownloadReadSize の値を大きくすると、正常な同期でのパフォーマンスが向上しますが、エラーが発生したときに再送信されるデータが多くなります。

このオプションの一般的な用途は、通信の信頼性が低いときにデフォルトのサイズを減らすことです。

デフォルトは **32767** です。このオプションを 32767 より大きな値に設定すると、32767 が使用されます。

このオプションには省略形と長形式があり、**drs** または **DownloadReadSize** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「-drs オプション」 156 ページ
- 「失敗したダウンロードの再開」 『Mobile Link - サーバ管理』
- 「ContinueDownload (cd) 拡張オプション」 201 ページ
- 「sp_hook_dbmlsync_end」 285 ページ
- 「-dc オプション」 153 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "drs=100"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION drs='100';
```


ErrorLogSendLimit (el) 拡張オプション

同期エラーが発生した場合、dbmlsync からサーバに送信するリモート・メッセージ・ログ・ファイルのサイズを指定します。

構文

el=*number*[**K** | **M**]; ...

備考

このオプションは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。

このオプションは、同期中にエラーが発生した場合に、dbmlsync が Mobile Link サーバに送信するメッセージ・ログのバイト数を指定します。dbmlsync メッセージ・ログを送信しない場合は、このオプションを **0** に設定します。

このオプションを 0 以外に設定すると、クライアント側のエラーが発生したときにエラー・ログがアップロードされます。クライアント側のすべてのエラーで、ログが送信されるわけではありません。通信エラーや、dbmlsync が Mobile Link サーバに接続されていないときに発生したエラーでは、ログは送信されません。アップロード送信後にエラーが発生した場合、エラー・ログがアップロードされるのは、SendDownloadAck 拡張オプションが ON に設定された場合だけです。

ErrorLogSendLimit に十分な大きさの値を設定すると、dbmlsync は現在のセッションのメッセージ・ログ全体を、Mobile Link サーバに送信します。たとえば、メッセージ・ログのメッセージが古いメッセージ・ログ・ファイルに追加された場合、dbmlsync は現在のセッション中に生成された新しいメッセージだけを送信します。新しいメッセージ全体の長さが ErrorLogSendLimit を超える場合、dbmlsync は新しく生成されたエラー・メッセージとログ・メッセージの最後の部分だけを、指定されたサイズの範囲内で送信します。

注意：メッセージ・ログのサイズは、指定されている冗長性の設定の影響を受けます。冗長性を調節するには、dbmlsync -v オプションを使用するか、"verbose" で始まる dbmlsync 拡張オプションを使用します。詳細については、「-v オプション」 189 ページと次に示す -e 冗長性オプションを参照してください。

- 「Verbose (v) 拡張オプション」 233 ページ
- 「VerboseHooks (vs) 拡張オプション」 234 ページ
- 「VerboseMin (vm) 拡張オプション」 235 ページ
- 「VerboseOptions (vo) 拡張オプション」 236 ページ
- 「VerboseRowCounts (vn) 拡張オプション」 237 ページ
- 「VerboseRowValues (vr) 拡張オプション」 238 ページ
- 「VerboseUpload (vu) 拡張オプション」 239 ページ

デフォルトは **32K** です。

このオプションには省略形と長形式があり、**el** または **ErrorLogSendLimit** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "el=32k"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION el='32k';
```


FireTriggers (ft) 拡張オプション

ダウンロードが適用されたときにリモート・データベースでトリガが起動されるように指定します。

構文

ft={ ON | OFF }; ...

備考

デフォルトは **ON** です。

このオプションには省略形と長形式があり、**ft** または **FireTriggers** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "ft=off"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION ft='off';
```


HoverRescanThreshold (hrt) 拡張オプション

スケジュールを使用している場合、再スキャンの実行までに累積可能な廃棄メモリ量を制限します。

構文

hrt=*number*[**K** | **M**]; ...

備考

メモリ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。デフォルトは **1m** です。

コマンド・ラインで複数の -n オプションを指定すると、メモリ破棄の原因となる断片化が dbmlsync で起きる可能性があります。破棄されたメモリは、データベース・トランザクション・ログの再スキャンによってのみリカバリできます。このオプションでは、ログを再スキャンしてメモリをリカバリするまでに累積可能な廃棄メモリ量を制限できます。破棄されたメモリのリカバリを制御するもう 1 つの方法は、sp_hook_dbmlsync_log_rescan ストアド・プロシージャを実行することです。

このオプションには省略形と長形式があり、**hrt** または **HoverRescanThreshold** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「sp_hook_dbmlsync_log_rescan」 288 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "hrt=2m"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION hrt='2m';
```


IgnoreHookErrors (eh) 拡張オプション

フック関数内で発生したエラーを無視するように指定します。

構文

eh={ ON | OFF }; ...

備考

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**eh** または **IgnoreHookErrors** を使用できます。

このオプションは、dbmlsync -eh オプションと同じです。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "eh=off"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION eh='off';
```

IgnoreScheduling (isc) 拡張オプション

スケジュール設定を無視するように指定します。

構文

isc={ ON | OFF }; ...

備考

ON に設定すると、dbmlsync は拡張オプションで指定されたスケジュール情報を無視して、すぐに同期を行います。デフォルトは **OFF** です。

このオプションは、dbmlsync -is オプションと同じです。

このオプションには省略形と長形式があり、**isc** または **IgnoreScheduling** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「同期のスケジュール」 128 ページ
- 「Schedule (sch) 拡張オプション」 223 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "isc=off"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION isc='off';
```


Increment (inc) 拡張オプション

インクリメンタル・アップロードを有効にし、インクリメンタル・アップロードのサイズを制御します。

構文

inc=*number*[**K** | **M**]; ...

備考

このオプションは、インクリメンタル・スキャンの最小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。

このオプションを指定すると、アップロードは 1 つ以上の部分に分けて Mobile Link に送信されます。このオプションは、サイトが完全なアップロードを完了するだけの長い時間接続を維持できない場合に役立ちます。オプションが設定されていない場合、アップロードは 1 つにまとめて送信されます。

このオプションの値は、大まかに各アップロード部分のサイズを指定します。オプションの値は、次のように各アップロード部分のサイズを制御します。dbmlsync は、データベース・トランザクション・ログをスキャンして、アップロードを構築します。このオプションを指定すると、dbmlsync はオプションで設定されたバイト数をスキャンし、未処理トランザクションがない最初の時点、つまりすべてのトランザクションがコミットまたはロールバックされた次の時点までスキャンを続けます。dbmlsync は、スキャンした部分をアップロード部分として送信し、中断したところからログのスキャンを再開します。

Increment 拡張オプションは、スクリプト化されたアップロードやトランザクション単位のアップロードでは使用できません。

このオプションには省略形と長形式があり、**inc** または **Increment** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "inc=32000"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION inc='32k';
```

LockTables (lt) 拡張オプション

同期されるパブリケーション内のテーブルを同期する前にロックするよう指定します。

構文

lt={ ON | OFF | SHARE | EXCLUSIVE }; ...

備考

SHARE は、dbmlsync が共有モードのすべての同期テーブルをロックすることを意味します。EXCLUSIVE は、dbmlsync が排他モードのすべての同期テーブルをロックすることを意味します。Windows Mobile 以外のプラットフォームでは、ON は SHARE と同じです。Windows Mobile デバイスでは、ON は EXCLUSIVE と同じです。

デフォルトは OFF です。つまり、デフォルトでは、次の場合を除いて dbmlsync は同期テーブルをロックしません。

- 現在の同期でスクリプトベースのアップロードを使用するパブリケーションがある場合、およびリモート・データベースで sp_hook_dbmlsync_schema_upgrade フックが定義されている場合、dbmlsync は同期テーブルを ON でロックします。
- 以前の同期でダウンロードされたパススルー・スクリプトがあり、かつそれらのスクリプトを現在の同期で自動的に実行する必要がある場合、パススルー・スクリプトの要件に応じて同期テーブルが SHARE または EXCLUSIVE でロックされます。

同期中の修正を禁止するには、ON に設定します。

共有ロックと排他ロックの詳細については、「ロックの仕組み」『SQL Anywhere サーバ - SQL の使用法』と「LOCK TABLE 文」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

Mobile Link アプリケーションでのテーブルのロックの詳細については、「同期中の同時実行性」 121 ページを参照してください。

同期テーブルが排他モードでロックされているとき (Windows Mobile デバイスではデフォルト)、他の接続はそのテーブルにアクセスできません。このため、別個の接続で実行する dbmlsync ストアド・プロシージャは、同期テーブルのいずれかにアクセスする必要がある場合は実行できません。

別個の接続で実行するフックについては、「SQL Anywhere クライアントのイベント・フック」 247 ページを参照してください。

このオプションには省略形と長形式があり、**lt** または **LockTables** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "lt=on"
```


次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION lt='on';
```

Memory (mem) 拡張オプション

アップロードを構築するときに dbmlsync で使用するキャッシュ・サイズを指定します。

構文

mem=*number*[**K** | **M**]; ...

備考

アップロードを構築するために使用されるキャッシュのサイズをバイト単位で指定します。キャッシュのサイズを大きくすると、メモリ内に保持できるデータのページが多くなるため、ディスクの読み込みと書き込みの回数が減少し、パフォーマンスが向上します。

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。デフォルトは **1M** です。

このオプションには省略形と長形式があり、**mem** または **Memory** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「-urc オプション」 187 ページ
- 「パフォーマンスに関するヒント」 『Mobile Link - サーバ管理』

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "mem=2M"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION mem='2m';
```


MirrorLogDirectory (mld) 拡張オプション

古いトランザクション・ログのミラー・ファイルを削除できるようにその場所を指定します。

構文

mld=*filename*; ...

備考

このオプションを指定すると、次のどちらかの状況になった場合に、dbmlsync は古いトランザクション・ログ・ミラー・ファイルを削除できます。

- オフライン・トランザクション・ログ・ミラーが、トランザクション・ログ・ミラーとは異なるディレクトリに置かれる

  または

- dbmlsync がリモート・データベース・サーバとは異なるコンピュータで実行されている

通常の設定では、アクティブなトランザクション・ログ・ミラーと名前の変更されたトランザクション・ログ・ミラー・ファイルは同じディレクトリに置かれ、dbmlsync はリモート・データベースと同じコンピュータで実行されるため、このオプションは不要であり、古いトランザクション・ログ・ミラー・ファイルは自動的に削除されます。

このディレクトリ内のトランザクション・ログが影響を受けるのは、delete_old_logs データベース・オプションが On、Delay、または *n* 日に設定されている場合だけです。

このオプションには省略形と長形式があり、**mld** または **MirrorLogDirectory** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「delete_old_logs オプション [Mobile Link クライアント] [SQL Remote] [Replication Agent]」『SQL Anywhere サーバ - データベース管理』

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "mld=c:¥tmp¥file"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION mld='c:¥tmp¥file';
```


MobiLinkPwd (mp) 拡張オプション

Mobile Link パスワードを指定します。

構文

mp=*password*; ...

備考

接続に使用するパスワードを指定します。このパスワードは、サブスクリプションが同期されている Mobile Link ユーザの正しいパスワードにする必要があります。このユーザは、dbmlsync -u オプションで指定することもできます。デフォルト値は NULL です。

Mobile Link ユーザがすでにパスワードを持っている場合は、拡張オプション **-e mn** で変更できます。

このオプションには省略形と長形式があり、**mp** または **MobiLinkPwd** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「NewMobiLinkPwd (mn) 拡張オプション」 219 ページ
- 「-mn オプション」 166 ページ
- 「-mp オプション」 167 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "mp=password"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION mp='SQL';
```


NewMobiLinkPwd (mn) 拡張オプション

新しいパスワードを指定します。

構文

mn=*new-password*; ...

備考

サブスクリプションが同期されている Mobile Link ユーザの新しいパスワードを指定します。このオプションは、既存のパスワードを変更する場合に使用します。デフォルトでは、パスワードは変更されません。

このオプションには省略形と長形式があり、**mn** または **NewMobiLinkPwd** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「MobiLinkPwd (mp) 拡張オプション」 218 ページ
- 「-mn オプション」 166 ページ
- 「-mp オプション」 167 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "mp=oldpassword; mn=newpassword"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION mn='SQL';
```


NoSyncOnStartup (nss) 拡張オプション

dbmlsync が起動時に同期するのを防ぎます。このオプションを指定しない場合は、スケジューリング・オプションにより、dbmlsync が起動時に同期されます。

構文

nss={ on | off }; ...

備考

このオプションが有効なのは、スケジュールが EVERY または INFINITE 句と一緒に使用されている場合だけです。これらのスケジュール・オプションでは、dbmlsync は起動時に自動的に同期します。

デフォルトは off です。

NoSyncOnStartup を on に設定し、INFINITE 句と一緒にスケジュールを使用すると、ウィンドウ・メッセージが受信されるまで同期は行われません。

NoSyncOnStartup を on に設定して、EVERY 句と一緒にスケジュールを使用すると、起動後の最初の同期は、EVERY 句で指定した期間の経過後に行われます。

この設定は、dbmlsync 起動時以外、スケジュールの動作に影響することはありません。

このオプションには省略形と長形式があり、**nss** または **NoSyncOnStartup** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「Schedule (sch) 拡張オプション」 223 ページ
- 「同期のスケジュール」 128 ページ

例

次の dbmlsync コマンド・ラインの一部は、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "schedule=EVERY:01:00;nss=off"...
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION nss='off', schedule='EVERY:01:00';
```


OfflineDirectory (dir) 拡張オプション

オフライン・トランザクションのログを含むパスを指定します。

構文

dir=*path*; ...

備考

dbmlsync はデフォルトで、オンライン・トランザクション・ログと同じディレクトリ内で、名前の変更されたログを確認できます。このオプションは、名前の変更されたオフライン・トランザクション・ログが別のディレクトリにある場合にのみ指定する必要があります。

このオプションには省略形と長形式があり、**dir** または **OfflineDirectory** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "dir=c:¥db¥logs"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION dir='c:¥db¥logs';
```


PollingPeriod (pp) 拡張オプション

ログスキャンのポーリング周期を指定します。

構文

pp=*number*[**S** | **M** | **H** | **D**]; ...

備考

このオプションは、ログスキャンの間隔を指定します。サフィックス s、m、h、d を使用して、それぞれ秒、分、時間、日を指定します。デフォルトは **1** 分です。サフィックスを指定しない場合、デフォルトの時間単位は分です。

ログスキャンのポーリングは、同期をスケジュールしているとき、または sp_hook_dbmlsync_delay フックを使用しているときだけ実行されます。

ログスキャンのポーリングの説明は、「DisablePolling (p) 拡張オプション」 202 ページを参照してください。

このオプションは、**dbmlsync -pp** と同じです。

このオプションには省略形と長形式があり、**pp** または **PollingPeriod** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「DisablePolling (p) 拡張オプション」 202 ページ
- 「-pp オプション」 176 ページ
- 「-p オプション」 172 ページ
- 「sp_hook_dbmlsync_delay」 265 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "pp=5"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION pp='5';
```


Schedule (sch) 拡張オプション

同期のスケジュールを指定します。

構文

sch=*schedule*; ...

schedule : { **EVERY**:*hhhh*:*mm* | **INFINITE** | *singleSchedule* }

hhhh : **00** ... **9999**

mm : **00** ... **59**

singleSchedule : *day* **@***hh*:*mm*[**AM** | **PM**] [**-***hh*:*mm*[**AM** | **PM**]] ,...

hh : **00** ... **24**

mm : **00** ... **59**

day :
EVERYDAY | **WEEKDAY** | **MON** | **TUE** | **WED** | **THU** | **FRI** | **SAT** | **SUN** | *dayOfMonth*

dayOfMonth : **0**... **31**

パラメータ

EVERY　EVERY キーワードを指定すると、同期は起動時に発生し、その後は指定した期間の経過後に無限に繰り返されます。指定した期間より同期処理に長時間かかる場合、同期はすぐに再開されます。

dbmlsync の起動時に同期が発生するのを防ぐには、拡張オプション NoSyncOnStartup を使用します。「NoSyncOnStartup (nss) 拡張オプション」 220 ページを参照してください。

singleSchedule　1 つ以上の単一スケジュールを指定すると、同期は指定した日時にのみ発生します。

間隔は *@hh:mm-hh:mm* (AM または PM はオプション指定) のように指定します。AM または PM を指定しない場合は、24 時間制とみなされます。24:00 は翌日の午前 00:00 とみなされます。間隔を指定すると、同期は間隔中の任意の時点から始まります。間隔を指定すると同期を行う時間帯に幅ができるため、同じスケジュールを持つ複数のリモート・データベースが、まったく同じ時刻に同期を行うことがなく、Mobile Link サーバ側では輻輳が生じなくなります。

間隔の終了時刻は常に開始時刻より後の時刻として解釈されます。間隔に真夜中が含まれている場合は、翌日に終了します。dbmlsync が間隔の途中で始まる場合、同期は終了時刻より前の任意の時点で発生します。

EVERYDAY　EVERYDAY とは週の 7 日間すべてのことです。

WEEKDAY　WEEKDAY とは月曜日から金曜日までのことです。

曜日は、Mon、Tue のようになります。Monday のような完全形も使用できます。使用言語が英語でない場合、クライアントによって接続文字列で要求された言語でない場合、サーバ・ウィンドウに表示される言語でない場合は、完全形を使用します。

dayOfMonth　月の長さに関係なく、月の最終日を指定するには、*dayOfMonth* を 0 に設定します。

INFINITE　INFINITE キーワードを指定すると、dbmlsync の同期は起動時に発生します。その後同期が発生するのは、ウィンドウ・メッセージを dbmlsync に送信する別のプログラムから同期が開始されたときだけです。待機中、dbmlsync が実行され、ログが定期的にスキャンされます。dbmlsync 拡張オプション NoSyncOnStartup を使用すると、初期同期を防ぐことができます。

詳細については、「NoSyncOnStartup (nss) 拡張オプション」 220 ページを参照してください。

このオプションを dbmlsync -wc オプションと組み合わせて使用すると、dbmlsync をウェイクアップし、同期を行うことができます。

詳細については、「-wc オプション」 190 ページを参照してください。

備考

直前の同期がスケジュール時刻にまだ完了していない場合は、その同期が完了した時点で、スケジュールされた同期が開始されます。

デフォルトは、スケジュール設定なしです。

このオプションには省略形と長形式があり、**sch** または **Schedule** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

schedule オプションの構文は、同期 SQL 文と dbmlsync コマンド・ラインのどちらの場合も同じです。

IgnoreScheduling 拡張オプションと -is オプションは、dbmlsync がスケジュールを無視して即時の同期を行えるようにします。詳細については、「IgnoreScheduling (isc) 拡張オプション」 212 ページを参照してください。

スケジュールの詳細については、「同期のスケジュール」 128 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "sch=WEEKDAY@8:00am,SUN@9:00pm"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION sch='WEEKDAY@8:00am,SUN@9:00pm';
```


ScriptVersion (sv) 拡張オプション

スクリプト・バージョンを指定します。

構文

sv=*version-name*; ...

備考

スクリプト・バージョンは、同期中に統合データベースの Mobile Link によって実行されるスクリプトを決定します。デフォルトのスクリプト・バージョンは **default** です。

このオプションには省略形と長形式があり、**sv** または **ScriptVersion** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "sv=SyaAd001"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION sv='SysAd001';
```


SendColumnNames (scn) 拡張オプション

ダイレクト・ロー・ハンドリングで使用するために、アップロード時にカラム名が送信されるように指定します。

構文

scn={ ON | OFF }; ...

備考

カラム名は、Mobile Link サーバによってダイレクト・ロー・ハンドリングで使用されます。ダイレクト・ロー・ハンドリング API を使用して、インデックスではなく、名前でカラムを参照する場合は、このオプションを設定してください。このオプションで送信されるカラム名は、このような場合にのみ使用されます。

「ダイレクト・ロー・ハンドリング」『Mobile Link - サーバ管理』を参照してください。

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**scn** または **SendColumnNames** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "scn=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION scn='on';
```


SendDownloadACK (sa) 拡張オプション

クライアントからサーバにダウンロード確認が送信されるように指定します。

構文

sa={ ON | OFF }; ...

備考

SendDownloadAck を OFF に設定することをおすすめします。ダウンロードに失敗しても、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。このオプションは、サーバ側のスクリプトの動作を変更するために使用します。「nonblocking_download_ack 接続イベント」『Mobile Link - サーバ管理』を参照してください。

ダウンロード確認を OFF にすることによるパフォーマンスの向上については、「非ブロッキング・ダウンロード確認の使用」『Mobile Link - サーバ管理』を参照してください。

注意：SendDownloadAck が ON に設定され、冗長モードである場合、受信確認行はクライアント・ログに書き込まれます。

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**sa** または **SendDownloadACK** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "sa=off"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION sa='off';
```


SendTriggers (st) 拡張オプション

アップロード時にトリガの動作が送信されるように指定します。

構文

st={ ON | OFF }; ...

備考

カスケード削除もトリガの動作と見なされます。

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**st** または **SendTriggers** を使用できます。

2 つのパブリケーションに重なりがある (つまり、両方のパブリケーションに同じテーブルが 1 つ以上含まれている) 場合は、SendTriggers オプションについては同じ設定を使用して、両方のパブリケーションを同期する必要があります。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "st=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION st='on';
```


TableOrder (tor) 拡張オプション

アップロードでのテーブルの順序を指定します。

構文

tor=*tables*; ...

tables = *table-name* [**,***table-name*], ...

備考

このオプションを使用すると、アップロードされるリモート・データベースのテーブルの順序を指定できます。*tables* はカンマで区切ったリストで指定します。アップロードされるすべてのテーブルを指定する必要があります。同期に含まれていないテーブルを指定すると、そのテーブルは無視されます。

指定するテーブルの順序は、参照整合性を保つようにします。つまり、Table1 が Table2 を外部キー参照する場合、Table2 は Table1 の前にアップロードする必要があります。適切な順序でテーブルを指定しないと、次の 2 つの場合を除いてエラーが発生します。

- TableOrderChecking=OFF を設定した場合。
- テーブルが環状外部キーに関連付けられている場合(この場合、ルールを満たす順序はないので、循環されるテーブルはどんな順序でもアップロードできます)

TableOrder を指定しないと、dbmlsync は、参照整合性を満たす順序を選択します。

ダウンロードのテーブルの順序は、アップロードと同じです。アップロード・テーブルの順序の制御により、サーバ側のスクリプトを簡単に記述することができます。特に、リモート・データベースと統合データベースの外部キー制約が異なる場合に効果を発揮します。

このオプションを使用しなければならないケースはありません。このパラメータは、特定の順序でテーブルをアップロードするユーザのために提供されています。

このオプションには省略形と長形式があり、**tor** または **TableOrder** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「TableOrderChecking (toc) 拡張オプション」 231 ページ
- 「アップロードの処理方法」 『Mobile Link - クイック・スタート』
- 「参照整合性と同期」 『Mobile Link - クイック・スタート』

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "tor=admin,parent,child"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION tor='admin,parent,child';
```


TableOrderChecking (toc) 拡張オプション

TableOrder を指定する場合、テーブルが、別のテーブルに対する外部キーを持つときに、別のテーブルよりも前にアップロードされないように、dbmlsync でチェックする必要があるかどうかが決定されます。

構文

tor={ OFF | ON }; ...

備考

ほとんどのアプリケーションで、リモート・データベースと統合データベースのテーブルには、同じ外部キーが関連付けられています。このような場合、TableOrderChecking をデフォルトの値 ON のままにしておくと、dbmlsync によって、テーブルが別のテーブルに対する外部キーを持つときに、別のテーブルよりも前にアップロードされないようにすることができます。これにより、参照整合性が確保されます。

このオプションは、統合データベースとリモート・データベースの外部キーの関係が異なる場合に便利です。TableOrder 拡張オプションと一緒に使用すると、外部キー関係のあるテーブルよりも前にテーブルがアップロードされてはならないという規則に従わないテーブルの順序を指定できます。

このオプションは、TableOrder 拡張オプションが指定された場合のみ役立ちます。

デフォルトは **ON** です。

このオプションには省略形と長形式があり、**toc** または **TableOrderChecking** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「TableOrder (tor) 拡張オプション」 229 ページ
- 「アップロードの処理方法」 『Mobile Link - クイック・スタート』
- 「参照整合性と同期」 『Mobile Link - クイック・スタート』

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "toc=OFF"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION toc='Off';
```

UploadOnly (uo) 拡張オプション

同期がアップロードだけを含むように指定します。

構文

uo={ ON | OFF }; ...

備考

アップロード専用の同期中に、dbmlsync は通常の完全な同期とまったく同じように、Mobile Link サーバへのアップロードを準備し送信します。しかし、ダウンロードを返信する代わりに、Mobile Link はアップロードのコミットが成功したかどうかを示す確認だけを送信します。

ダウンロード専用同期に定義する必要があるスクリプトのリストについては、「必要なスクリプト」 『Mobile Link - サーバ管理』を参照してください。

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**uo** または **UploadOnly** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「アップロード専用の同期とダウンロード専用の同期」 『Mobile Link - サーバ管理』
- 「DownloadOnly (ds) 拡張オプション」 204 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "uo=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION uo='on';
```


Verbose (v) 拡張オプション

完全冗長を指定します。

構文

v={ ON | OFF }; ...

備考

このオプションが指定する高いレベルの冗長性は、パフォーマンスに影響する場合があるので、通常は開発段階にだけ使用してください。

このオプションは、**dbmlsync -v+** と同じです。-v オプションと拡張オプションの両方を指定して、競合が発生した場合は、-v オプションが拡張オプションよりも優先されます。競合が発生しない場合は、冗長ログ・オプションが追加され、指定したすべてのオプションが使用されます。拡張オプションでロギングの冗長性を設定しても、ロギングは直ちに有効にならないため、起動情報のログは取られません。最初の同期が行われる時点では、-v オプションを指定した場合と拡張オプションを指定した場合のロギングの動作は同じです。

詳細については、「-v オプション」 189 ページを参照してください。

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**v** または **Verbose** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「VerboseHooks (vs) 拡張オプション」 234 ページ
- 「VerboseMin (vm) 拡張オプション」 235 ページ
- 「VerboseOptions (vo) 拡張オプション」 236 ページ
- 「VerboseRowCounts (vn) 拡張オプション」 237 ページ
- 「VerboseRowValues (vr) 拡張オプション」 238 ページ
- 「VerboseUpload (vu) 拡張オプション」 239 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "v=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION v='on';
```


VerboseHooks (vs) 拡張オプション

フック・スクリプトに関するメッセージのログを取るように指定します。

構文

vs={ ON | OFF }; ...

備考

このオプションは、**dbmlsync -vs** と同じです。-v オプションと拡張オプションの両方を指定して、競合が発生した場合は、-v オプションが拡張オプションよりも優先されます。競合が発生しない場合は、冗長ログ・オプションが追加され、指定したすべてのオプションが使用されます。拡張オプションでロギングの冗長性を設定しても、ロギングは直ちに有効にならないため、起動情報のログは取られません。最初の同期が行われる時点では、-v オプションを指定した場合と拡張オプションを指定した場合のロギングの動作は同じです。

詳細については、「-v オプション」 189 ページを参照してください。

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**vs** または **VerboseHooks** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「SQL Anywhere クライアントのイベント・フック」 247 ページ
- 「Verbose (v) 拡張オプション」 233 ページ
- 「VerboseMin (vm) 拡張オプション」 235 ページ
- 「VerboseOptions (vo) 拡張オプション」 236 ページ
- 「VerboseRowCounts (vn) 拡張オプション」 237 ページ
- 「VerboseRowValues (vr) 拡張オプション」 238 ページ
- 「VerboseUpload (vu) 拡張オプション」 239 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "vs=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION vs='on';
```


VerboseMin (vm) 拡張オプション

少量の情報のログを取るように指定します。

構文

vm={ ON | OFF }; ...

備考

このオプションは、**dbmlsync -v** と同じです。-v オプションと拡張オプションの両方を指定して、競合が発生した場合は、-v オプションが拡張オプションよりも優先されます。競合が発生しない場合は、冗長ログ・オプションが追加され、指定したすべてのオプションが使用されます。拡張オプションでロギングの冗長性を設定しても、ロギングは直ちに有効にならないため、起動情報のログは取られません。最初の同期が行われる時点では、-v オプションを指定した場合と拡張オプションを指定した場合のロギングの動作は同じです。

詳細については、「-v オプション」 189 ページを参照してください。

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**vm** または **VerboseMin** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「Verbose (v) 拡張オプション」 233 ページ
- 「Verbose (v) 拡張オプション」 233 ページ
- 「VerboseOptions (vo) 拡張オプション」 236 ページ
- 「VerboseRowCounts (vn) 拡張オプション」 237 ページ
- 「VerboseRowValues (vr) 拡張オプション」 238 ページ
- 「VerboseUpload (vu) 拡張オプション」 239 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "vm=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION vm='on';
```


VerboseOptions (vo) 拡張オプション

指定したコマンド・ライン・オプション (拡張オプションを含む) に関する情報のログを取るように指定します。

構文

vo={ ON | OFF }; ...

備考

このオプションは、**dbmlsync -vo** と同じです。-v オプションと拡張オプションの両方を指定して、競合が発生した場合は、-v オプションが拡張オプションよりも優先されます。競合が発生しない場合は、冗長ログ・オプションが追加され、指定したすべてのオプションが使用されます。拡張オプションでロギングの冗長性を設定しても、ロギングは直ちに有効にならないため、起動情報のログは取られません。最初の同期が行われる時点では、-v オプションを指定した場合と拡張オプションを指定した場合のロギングの動作は同じです。

詳細については、「-v オプション」 189 ページを参照してください。

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**vo** または **VerboseOptions** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「Verbose (v) 拡張オプション」 233 ページ
- 「Verbose (v) 拡張オプション」 233 ページ
- 「VerboseMin (vm) 拡張オプション」 235 ページ
- 「VerboseRowCounts (vn) 拡張オプション」 237 ページ
- 「VerboseRowValues (vr) 拡張オプション」 238 ページ
- 「VerboseUpload (vu) 拡張オプション」 239 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "vo=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION vo='on';
```


VerboseRowCounts (vn) 拡張オプション

アップロードおよびダウンロードされるローの数のログを取るように指定します。

構文

vn={ ON | OFF }; ...

備考

このオプションは、**dbmlsync -vn** と同じです。-v オプションと拡張オプションの両方を指定して、競合が発生した場合は、-v オプションが拡張オプションよりも優先されます。競合が発生しない場合は、冗長ログ・オプションが追加され、指定したすべてのオプションが使用されます。拡張オプションでロギングの冗長性を設定しても、ロギングは直ちに有効にならないため、起動情報のログは取られません。最初の同期が行われる時点では、-v オプションを指定した場合と拡張オプションを指定した場合のロギングの動作は同じです。

詳細については、「-v オプション」 189 ページを参照してください。

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**vn** または **VerboseRowCounts** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「Verbose (v) 拡張オプション」 233 ページ
- 「Verbose (v) 拡張オプション」 233 ページ
- 「VerboseMin (vm) 拡張オプション」 235 ページ
- 「VerboseOptions (vo) 拡張オプション」 236 ページ
- 「VerboseRowValues (vr) 拡張オプション」 238 ページ
- 「VerboseUpload (vu) 拡張オプション」 239 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "vn=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION vn='on';
```


VerboseRowValues (vr) 拡張オプション

アップロードおよびダウンロードされるローの値のログを取るように指定します。

構文

vr={ ON | OFF }; ...

備考

このオプションは、**dbmlsync -vr** と同じです。-v オプションと拡張オプションの両方を指定して、競合が発生した場合は、-v オプションが拡張オプションよりも優先されます。競合が発生しない場合は、冗長ログ・オプションが追加され、指定したすべてのオプションが使用されます。拡張オプションでロギングの冗長性を設定しても、ロギングは直ちに有効にならないため、起動情報のログは取られません。最初の同期が行われる時点では、-v オプションを指定した場合と拡張オプションを指定した場合のロギングの動作は同じです。

詳細については、「-v オプション」 189 ページを参照してください。

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**vr** または **VerboseRowValues** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「Verbose (v) 拡張オプション」 233 ページ
- 「Verbose (v) 拡張オプション」 233 ページ
- 「VerboseMin (vm) 拡張オプション」 235 ページ
- 「VerboseOptions (vo) 拡張オプション」 236 ページ
- 「VerboseRowCounts (vn) 拡張オプション」 237 ページ
- 「VerboseUpload (vu) 拡張オプション」 239 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "vr=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION vr='on';
```


VerboseUpload (vu) 拡張オプション

アップロード・ストリームに関する情報のログを取るように指定します。

構文

vu={ ON | OFF }; ...

備考

このオプションは、**dbmlsync -vu** と同じです。-v オプションと拡張オプションの両方を指定して、競合が発生した場合は、-v オプションが拡張オプションよりも優先されます。競合が発生しない場合は、冗長ログ・オプションが追加され、指定したすべてのオプションが使用されます。拡張オプションでロギングの冗長性を設定しても、ロギングは直ちに有効にならないため、起動情報のログは取られません。最初の同期が行われる時点では、-v オプションを指定した場合と拡張オプションを指定した場合のロギングの動作は同じです。

詳細については、「-v オプション」 189 ページを参照してください。

デフォルトは **OFF** です。

このオプションには省略形と長形式があり、**vu** または **VerboseUpload** を使用できます。

拡張オプションのデータベースへの格納もできます。dbmlsync 拡張オプションの詳細については、「dbmlsync 拡張オプションの概要」 195 ページを参照してください。

参照

- 「Verbose (v) 拡張オプション」 233 ページ
- 「Verbose (v) 拡張オプション」 233 ページ
- 「VerboseMin (vm) 拡張オプション」 235 ページ
- 「VerboseOptions (vo) 拡張オプション」 236 ページ
- 「VerboseRowCounts (vn) 拡張オプション」 237 ページ
- 「VerboseRowValues (vr) 拡張オプション」 238 ページ

例

次の dbmlsync コマンド・ラインは、dbmlsync を使用するときのこのオプションの設定方法を示します。

```
dbmlsync -e "vu=on"
```

次の SQL 文は、このオプションのデータベースへの格納方法を示します。

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION vu='on';
```


Mobile Link SQL 文

目次

Mobile Link 文

次に、Mobile Link SQL Anywhere クライアントの構成と実行に使用する SQL 文を示します。

- 「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「ALTER SYNCHRONIZATION PROFILE 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「ALTER SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「ALTER SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE SYNCHRONIZATION PROFILE 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「DROP PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「DROP SYNCHRONIZATION PROFILE 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「DROP SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「DROP SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「START SYNCHRONIZATION DELETE 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』

Ultra Light クライアント

「Ultra Light SQL 文」『Ultra Light データベース管理とリファレンス』を参照してください。

Mobile Link 同期プロファイル

目次

Mobile Link 同期プロファイル

同期プロファイルを使用すると、いくつかの dbmlsync オプションをデータベースに配置できます。作成するプロファイルには、さまざまな同期オプションを含めることができます。

同期プロファイルを作成したら、使用するときに、dbmlsync -sp オプションを使用し、同期プロファイルの名前を指定します。同期プロファイルで指定されたオプションは、指定したコマンド・ライン・オプションとマージされます。コマンド・ラインと同期プロファイルで同等のオプションが指定された場合、コマンド・ラインにあるオプションが、プロファイルにあるオプションよりも優先されます。「-sp オプション」 181 ページを参照してください。

同期プロファイルを作成、変更、削除するには、次の文を使用します。

- 「ALTER SYNCHRONIZATION PROFILE 文 [Mobile Link]」 『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE SYNCHRONIZATION PROFILE 文 [Mobile Link]」 『SQL Anywhere サーバ - SQL リファレンス』
- 「DROP SYNCHRONIZATION PROFILE 文 [Mobile Link]」 『SQL Anywhere サーバ - SQL リファレンス』

Ultra Light での同期プロファイルの使用については、「同期プロファイル・オプション」 『Ultra Light データベース管理とリファレンス』を参照してください。

同期プロファイルでは次のオプションを指定できます。

オプション名	省略名	有効な値	説明
AuthParms	ap	文字列	authenticate_parameters スクリプトと認証パラメータにパラメータを入力します。「-ap オプション」 146 ページを参照してください。
ApplyDnldFile	ba	文字列	ダウンロード・ファイルを適用します。「-ba オプション」 147 ページを参照してください。
ContinueDownload	dc	ブール式	以前に失敗したダウンロードを再起動します。「-dc オプション」 153 ページを参照してください。
CreateDnldFile	bc	文字列	ダウンロード・ファイルを作成します。「-bc オプション」 148 ページを参照してください。
DnldFileExtra	be	文字列	ダウンロード・ファイルを作成するとき、このオプションはファイルに含まれる追加の文字列を指定します。「-be オプション」 149 ページを参照してください。
DownloadOnly	ds	ブール式	ダウンロード専用同期を実行します。「-ds オプション」 157 ページを参照してください。

オプション名	省略名	有効な値	説明
DownloadReadSize	drs	整数	再起動可能なダウンロードについて、通信障害の後に再送する必要があるデータの最大値を指定します。「-drs オプション」 156 ページを参照してください。
ExtOpt	e	文字列	拡張オプションを指定します。「-e オプション」 158 ページを参照してください。
IgnoreHookErrors	eh	ブール式	フック関数で発生したエラーを無視します。「-eh オプション」 159 ページを参照してください。
IgnoreScheduling	is	ブール式	スケジュールの指示を無視して、直ちに同期を行います。「-is オプション」 163 ページを参照してください。
KillConnections	d	ブール式	リモート・データベースに対する競合ロックを削除します。「-d オプション」 152 ページを参照してください。
LogRenameSize	x	整数。オプションで後ろに K または M が付きます。	アップロード・データがスキャンされた後、トランザクション・ログの名前を変更し、再起動します。「-x オプション」 191 ページを参照してください。
MobiLinkPwd	mp	文字列	Mobile Link ユーザのパスワードを指定します。「-mp オプション」 167 ページを参照してください。
MLUser	u	文字列	Mobile Link ユーザ名を指定します。「-u オプション」 184 ページを参照してください。
NewMLPassword	mn	文字列	Mobile Link ユーザの新しいパスワードを指定します。このオプションは、既存のパスワードを変更する場合に使用します。「-mn オプション」 166 ページを参照してください。
Ping	pi	ブール式	Mobile Link サーバに対して ping を実行し、クライアントと Mobile Link 間の通信を確認します。「-pi オプション」 175 ページを参照してください。
Publication	n	文字列	同期させるパブリケーションを指定します。パブリケーションは同期プロファイル内で 1 回しか指定できませんが、コマンド・ライン・オプションは複数回指定できます。「-n オプション」 168 ページを参照してください。

オプション名	省略名	有効な値	説明
RemoteProgressGreater	ra	ブール式	リモート・オフセットが統合オフセットよりも大きい場合にリモート・オフセットが使用されるように指定します。これは、-ra オプションと同じです。「-r オプション」 179 ページを参照してください。
RemoteProgressLess	rb	ブール式	リモート・オフセットが統合オフセットよりも小さい場合に (リモート・データベースがバックアップからリストアされたときなど) リモート・オフセットが使用されるように指定します。これは、-rb オプションと同じです。「-r オプション」 179 ページを参照してください。
TransactionalUpload	tu	ブール式	リモート・データベースの各トランザクションを、1 つの同期内で独立したトランザクションとしてアップロードするように指定します。「-tu オプション」 182 ページを参照してください。
UpdateGenNum	bg	ブール式	ダウンロード・ファイルを作成するとき、このオプションはまだ同期していないリモート・データベースで使用できるファイルを作成します。「-bg オプション」 150 ページを参照してください。
UploadOnly	uo	ブール式	同期がアップロードだけを含み、ダウンロードが発生しないように指定します。「-uo オプション」 186 ページを参照してください。
UploadRowCnt	urc	整数	同期でアップロードされるロー数の推定値を指定します。「-urc オプション」 187 ページを参照してください。
Verbosity		文字列 (オプションのカンマ区切りのリスト)	dbmlsync の冗長性を制御します。「-v オプション」 189 ページに似ています。値は、次のオプションの 1 つ以上を含むカンマ区切りのリストにします。次に示すように、各オプションは既存の -v オプションに対応しています。 ● BASIC は -v と同じです。 ● HIGH は -v+ と同じです。 ● CONNECT_STR は -vc と同じです。 ● ROW_CNT は -vn と同じです。 ● OPTIONS は -vo と同じです。 ● ML_PASSWORD は -vp と同じです。 ● ROW_DATA は -vr と同じです。 ● HOOK は -vs と同じです。

SQL Anywhere クライアントのイベント・フック

目次

dbmlsync のフックの概要

SQL Anywhere 同期クライアント dbmlsync には、一連のイベント・フックがオプションで用意されています。これを使用して、同期処理をカスタマイズできます。フックを実装した場合は、同期処理における特定の時点で呼び出されます。

イベント・フックを実装するには、特定の名前の SQL ストアド・プロシージャを作成します。ほとんどのイベント・フック・ストアド・プロシージャは、同期自体と同じ接続で実行されます。

イベント・フックを使用して同期イベントのログを取り、処理することができます。たとえば、論理イベントに基づいた同期のスジュール、接続障害のリトライ、またはエラーや参照整合性違反の処理などが可能です。

また、イベント・フックを使用して、パブリケーションで簡単に定義できないデータのサブセットを同期することもできます。たとえば、2 つのイベント・フック・プロシージャを記述して、テンポラリ・テーブル内のデータを同期することができます。この場合、一方のイベント・フック・プロシージャでは、同期の前にテンポラリ・テーブルから永久テーブルにデータをコピーし、他方では同期後にデータを逆にコピーします。

警告
同期処理の整合性は、組み込みトランザクションの順序に依存します。イベント・フック・プロシージャ内では、暗黙的または明示的なコミットもロールバックも実行しないでください。

また、フック内の接続設定を変更すると、予期しない結果になる場合があります。フック内の接続設定の変更が必要な場合は、フックに古い値を復元してから、フックを完了してください。

dbmlsync インタフェース

クライアント・イベント・フックは、dbmlsync コマンド・ライン・ユーティリティと一緒に使用できます。または、SQL Anywhere クライアントの同期に使用するいずれのプログラミング・インタフェースとも一緒に使用できます。これには、dbmlsync API と dbmlsync 用の DBTools インタフェースが含まれます。

「dbmlsync の同期のカスタマイズ」 130 ページを参照してください。

同期イベント・フックの順序

次の疑似コードは、使用可能なイベントと、同期処理中に各イベントが呼び出されるポイントを示します。たとえば、sp_hook_dbmlsync_abort は最初に呼び出されるイベント・フックです。

各フックには、プロシージャの実装時に使用できるパラメータ値が用意されています。パラメータ値の中には、新しい値を返すように変更できるものがあります。それ以外のものは、読み込み専用です。これらのパラメータは、ストアド・プロシージャの引数ではありません。いずれのイベント・フック・ストアド・プロシージャにも、引数は渡されません。代わりに、#hook_dict テーブル内のローを読み込んだり修正したりすることで、引数がやりとりされます。

たとえば、sp_hook_dbmlsync_begin プロシージャには Mobile Link ユーザのパラメータが 1 つあります。これは、同期している Mobile Link ユーザです。この値は、#hook_dict テーブルから取り出すことができます。

順序は Mobile Link サーバでのイベントの順序と類似していますが、統合データベースとリモート・データベースに追加する論理の種類には、重複はほとんどありません。したがって、2 つのインタフェースは別のものとなります。

*_begin フックが正常に実行されると、*_begin フックの後にどのようなエラーが発生しても、対応する *_end フックが呼び出されます。*_end フックが定義されていても、*_begin フックが定義されていない場合は、通常 *_begin フックが呼び出される時点より前にエラーが発生しないかぎり、*_end フックが呼び出されます。

フックがデータベース内のデータを変更すると、sp_hook_dbmlsync_logscan_begin での変更を含むこれまでのすべての変更が、現在の同期セッションで同期されます。それ以降の変更は、次のセッションで同期されます。

```
sp_hook_dbmlsync_abort
sp_hook_dbmlsync_set_extended_options
loop until return codes direct otherwise (
   sp_hook_dbmlsync_abort
   sp_hook_dbmlsync_delay
)
sp_hook_dbmlsync_abort
// start synchronization
sp_hook_dbmlsync_begin
// upload events
for each upload segment
// a normal synchronization has one upload segment
// a transactional upload has one segment per transaction
// an incremental upload has one segment per upload piece
 sp_hook_dbmlsync_logscan_begin  //not called for scripted upload
 sp_hook_dbmlsync_logscan_end  //not called for scripted upload
 sp_hook_dbmlsync_set_ml_connect_info //only called during first upload
 sp_hook_dbmlsync_upload_begin
 sp_hook_dbmlsync_set_upload_end_ progress  //only called for scripted upload
 sp_hook_dbmlsync_upload_end
next upload segment
// download events
sp_hook_dbmlsync_validate_download_file (only called
   when -ba option is used)
for each download segment
sp_hook_dbmlsync_download_begin
for each table
   sp_hook_dbmlsync_download_table_begin
   sp_hook_dbmlsync_download_table_end
next table
sp_hook_dbmlsync_download_end

sp_hook_dbmlsync_schema_upgrade
// end synchronization
sp_hook_dbmlsync_end
sp_hook_dbmlsync_process_exit_code
sp_hook_dbmlsync_log_rescan
```

アップロード・オプションの詳細については、「-tu オプション」 182 ページと「Increment (inc) 拡張オプション」 213 ページを参照してください。

イベント・フック・プロシージャの使用

この項では、イベント・フック・プロシージャの設計と使用におけるいくつかの注意事項について説明します。

注意

- イベント・フック・プロシージャでは、COMMIT 操作も ROLLBACK 操作も実行しないでください。プロシージャは同期と同じ接続で実行されるので、COMMIT または ROLLBACK を実行すると同期が妨害されます。
- 接続設定は変更しないでください。フック内の接続設定を変更すると、予期しない結果になる場合があります。フック内の接続設定の変更が必要な場合は、フックに古い値を復元してから、フックを完了してください。
- イベント・フック接続は、ストアド・プロシージャを、所有者で識別せずに呼び出します。したがって、ストアド・プロシージャは、dbmlsync 接続で使用されるユーザ名 (通常は、REMOTE DBA 権限を持つユーザ)、または dbmlsync ユーザがメンバであるグループの ID のどちらかで所有されなければなりません。
- リモート・データベースは、各フックのインスタンスを 1 つだけ持ちます。所有者が異なるフックのインスタンスは、複数作成しないでください。
- フック・プロシージャは DBA 権限を持つユーザが作成してください。
- *_begin フックが正常に実行されると、*_begin フックの後にどのようなエラーが発生しても、対応する *_end フックが呼び出されます。*_end フックが定義されていても、*_begin フックが定義されていない場合は、通常 *_begin フックが呼び出される時点より前にエラーが発生しないかぎり、*_end フックが呼び出されます。

#hook_dict テーブル

フックが呼び出される直前に、dbmlsync は次の CREATE 文を使用してリモート・データベースに #hook_dict テーブルを作成します。テーブル名の前の # は、そのテーブルがテンポラリであることを意味します。

```
CREATE TABLE #hook_dict(
name VARCHAR(128) NOT NULL UNIQUE,
value VARCHAR(10240) NOT NULL)
```

dbmlsync ユーティリティは #hook_dict テーブルを使用してフック関数に値を渡し、フック関数は #hook_dict テーブルを使用して dbmlsync に値を戻します。

各フックは、パラメータ値を受け取ります。パラメータ値の中には、新しい値を返すように変更できるものがあります。それ以外のものは、読み込み専用です。このテーブルの各ローには、1 つのパラメータの値があります。

たとえば、次の dbmlsync コマンド・ラインでは、sp_hook_dbmlsync_abort フックを呼び出す時点では、#hook_dict テーブルには次のようなローが含まれています。

```
dbmlsync -c 'dsn=MyDsn' -n pub1,pub2 -u MyUser
```

#hook_dict ロー	値
publication_0	pub1
publication_1	pub2
MobiLink user	MyUser
Abort synchronization	false

アボート・フックを使用して、#hook_dict テーブルから値を取り出したり、その動作をカスタマイズしたりできます。たとえば、Mobile Link ユーザを取り出すには、次のように SELECT 文を使用します。

```
SELECT value
FROM #hook_dict
WHERE name = 'MobiLink user'
```

In/out パラメータは、dbmlsync の動作をフックで修正することによって更新できます。たとえば、次のような文を使用してテーブルの abort synchronization ローを更新することで、同期のアボートをフックから dbmlsync に命令することができます。

```
UPDATE #hook_dict
SET value='true'
WHERE name='abort synchronization'
```

各フックの記述には、#hook_dict テーブルのローがリストされます。

例

次のサンプル sp_hook_dbmlsync_delay プロシージャは、#hook_dict テーブルを使用して引数を受け渡す様子を示します。このプロシージャでは、Mobile Link システムのスケジュールされたダウン時間 (18:00 ～ 19:00) 以外にのみ同期を行います。

```
CREATE PROCEDURE sp_hook_dbmlsync_delay()
BEGIN
  DECLARE delay_val integer;
 SET delay_val=DATEDIFF(
  second, CURRENT TIME, '19:00');
 IF (delay_val>0 AND
    delay_val<3600)
 THEN
 UPDATE #hook_dict SET value=delay_val
  WHERE name='delay duration';
 END IF;
END
```

次のプロシージャは、同期の開始時にリモート・データベース内で実行されます。現在の Mobile Link ユーザ名 (sp_hook_dbmlsync_begin イベントに使用可能なパラメータの 1 つ) を取り出して、SQL Anywhere のメッセージ・ウィンドウに出力します。

```
CREATE PROCEDURE sp_hook_dbmlsync_begin()
BEGIN
  DECLARE syncdef VARCHAR(150);
  SELECT '>>>syncdef = ' || value INTO syncdef
    FROM #hook_dict
```



```
    WHERE name ='MobiLink user name';
  MESSAGE syncdef TYPE INFO TO CONSOLE;
END
```

イベント・フック・プロシージャ用の接続

各イベント・フック・プロシージャは、同期自体と同じ接続で実行されます。ただし、次のプロシージャは例外です。

- sp_hook_dbmlsync_all_error
- sp_hook_dbmlsync_communication_error
- sp_hook_dbmlsync_download_com_error
- sp_hook_dbmlsync_download_fatal_sql_error
- sp_hook_dbmlsync_download_log_ri_violation
- sp_hook_dbmlsync_misc_error
- sp_hook_dbmlsync_sql_error

これらのプロシージャは、同期が失敗する前に呼び出されます。失敗すると、同期アクションがロールバックされます。別の接続で実行すると、これらのプロシージャを使用して失敗情報のログを取ることができ、このとき、ログ・アクションは同期アクションとともにはロールバックされません。

イベント・フック・プロシージャ内でのエラーと警告の処理

イベント・フック・ストアド・プロシージャを作成すると、同期エラー、Mobile Link の接続障害、参照整合性違反を処理することができます。この項では、エラーや警告の処理に使用するイベント・フック・プロシージャについて説明します。実装された各プロシージャは、指定したタイプのエラーが発生するたびに自動的に実行されます。

参照整合性違反の処理

ダウンロード内のローがリモート・データベース上の外部キー関係に違反すると、参照整合性違反が発生します。次のイベント・フックを使用して参照整合性違反のログを取り、処理します。

- 「sp_hook_dbmlsync_download_log_ri_violation」 275 ページ
- 「sp_hook_dbmlsync_download_ri_violation」 277 ページ

Mobile Link 接続障害の処理

sp_hook_dbmlsync_ml_connect_failed イベント・フックを使用すると、Mobile Link サーバへの接続が失敗した場合に、異なる通信タイプまたはアドレスを使用してリトライすることができます。リトライしても接続が失敗した場合、dbmlsync は sp_hook_dbmlsync_communication_error と sp_hook_dbmlsync_all_error フックを呼び出します。

「sp_hook_dbmlsync_ml_connect_failed」 297 ページを参照してください。

dbmlsync エラーの処理

dbmlsync エラー・メッセージが生成されるたびに、次のフックが呼び出されます。

- まず、エラーのタイプに応じて、sp_hook_dbmlsync_communication_error、sp_hook_dbmlsync_misc_error、sp_hook_dbmlsync_sql_error のいずれかのフックが呼び出されます。これらのフックには、エラーのタイプに固有の情報が含まれています。たとえば、SQL エラーでは sqlcode と sqlstate が返されます。
- 次に、sp_hook_dbmlsync_all_error が呼び出されます。このフックには、発生したすべてのエラーのログを取る場合に役立ちます。

次の項を参照してください。

- 「sp_hook_dbmlsync_communication_error」 262 ページ
- 「sp_hook_dbmlsync_sql_error」 309 ページ
- 「sp_hook_dbmlsync_misc_error」 294 ページ
- 「sp_hook_dbmlsync_all_error」 257 ページ

エラーを受けて同期を再度開始するには、sp_hook_dbmlsync_end 内の user state パラメータを使用します。

「sp_hook_dbmlsync_end」 285 ページを参照してください。

エラーの無視

イベント・フック・プロシージャ内でエラーが発生した場合、デフォルトでは同期は停止します。dbmlsync ユーティリティで -eh オプションを指定すると、イベント・フック・プロシージャ内でエラーが発生しても無視するように指定できます。

「IgnoreHookErrors (eh) 拡張オプション」 211 ページを参照してください。

sp_hook_dbmlsync_abort

このストアド・プロシージャは、同期処理をキャンセルする場合に使用します。

#hook_dict テーブルのロー

名前	値	説明
abort synchronization (in|out)	**true** | **false**	#hook_dict テーブルの abort synchronization ローを **true** に設定すると、dbmlsync はイベント終了後すぐに終了します。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
exit code (in|out)	数値	abort synchronization を TRUE に設定すると、この値を使用して、アボートされた同期の終了コードを設定できます。0 は同期が成功したことを示します。他の数は同期が失敗したことを示します。
script version (in|out)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、そのプロシージャは dbmlsync の起動時に呼び出され、sp_hook_dbmlsync_delay フックにより各同期が遅延した後に再度呼び出されます。

abort synchronization の値を True に設定することによりフックがアボートを要求する場合、終了コードが sp_hook_dbmlsync_process_exit_code フックに渡されます。sp_hook_dbmlsync_process_exit_code フックが定義されていない場合、終了コードがプログラムの終了コードとして使用されます。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「同期イベント・フックの順序」 249 ページ
- 「sp_hook_dbmlsync_process_exit_code」 300 ページ

例

次のプロシージャは、毎日 19:00 ～ 20:00 にスケジュールされた保守作業時間中に、同期が行われないようにします。

```
CREATE PROCEDURE sp_hook_dbmlsync_abort()
BEGIN
  DECLARE down_time_start TIME;
  DECLARE is_down_time VARCHAR(128);
  SET down_time_start='19:00';
  IF datediff( hour,down_time_start,now(*) ) < 1
  THEN
    set is_down_time='true';
  ELSE
    SET is_down_time='false';
  END IF;
  UPDATE #hook_dict
  SET value = is_down_time
  WHERE name = 'abort synchronization'
END;
```

次の 2 つの理由のいずれかにより、同期をアボート可能なアボート・フックがあるとします。1 つ目の理由は、同期の正常終了を示すのに、dbmlsync に終了コード 0 を含ませるためです。また、2 つ目の理由は、エラー状態を示すのに、dbmlsync に 0 以外の終了コードを含ませるためです。sp_hook_dbmlsync_abort フックを次のように定義すれば、上記のことが可能です。

```
BEGIN
  IF [condition that defines the normal abort case] THEN
    UPDATE #hook_dict SET value =  '0'
  WHERE name = 'exit code';
    UPDATE #hook_dict SET value =  'TRUE'
  WHERE name = 'abort synchronization';
  ELSEIF [condition that defines the error abort case] THEN
    UPDATE #hook_dict SET value =  '1'
  WHERE name = 'exit code';
    UPDATE #hook_dict SET value =  'TRUE'
  WHERE name = 'abort synchronization';
  END IF;
END;
```


sp_hook_dbmlsync_all_error

このストアド・プロシージャを使用して、すべてのタイプの dbmlsync エラー・メッセージを処理します。たとえば、sp_hook_dbmlsync_all_error フックを実装すると、特定のエラーが発生した場合に、ログを取ったり特定のアクションを実行したりすることができます。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。
error message (in)	エラー・メッセージ・テキスト	これは、dbmlsync ログに表示されるテキストと同じです。
error id (in)	整数	メッセージをユニークに識別する ID。このローを使用すると、エラー・メッセージ・テキストが変更されたときに、エラー・メッセージを識別できます。
error hook user state (in|out)	整数	この値はフックによって設定して、今後の呼び出しに対するステータス情報を、sp_hook_dbmlsync_all_error、sp_hook_dbmlsync_communication_error、sp_hook_dbmlsync_misc_error、sp_hook_dbmlsync_sql_error、または sp_hook_dbmlsync_end フックに渡すことができます。これらのフックの 1 つが最初に呼び出されるとき、ローの値は 0 です。フックがローの値を変更した場合は、次回のフックの呼び出しには新しい値が使用されます。 このフックを使用して、sp_hook_dbmlsync_end フックにステータス情報を渡す場合は、sp_hook_dbmlsync_end フックにより同期のリトライなどのアクションを実行することができます。

備考

dbmlsync エラー・メッセージが生成されるたびに、次のフックが呼び出されます。

- まず、エラーのタイプに応じて、sp_hook_dbmlsync_communication_error、sp_hook_dbmlsync_misc_error、sp_hook_dbmlsync_sql_error のいずれかのフックが呼び出されます。これらのフックには、エラーのタイプに固有の情報が含まれています。たとえば、SQL エラーでは sqlcode と sqlstate が返されます。
- 次に、sp_hook_dbmlsync_all_error が呼び出されます。このフックには、発生したすべてのエラーのログを取る場合に役立ちます。

同期を開始する前の起動中にエラーが発生した場合、#hook_dict 内の Mobile Link ユーザとスクリプト・バージョンのエントリは空の文字列に設定され、#hook_dict テーブルで設定される publication_*n* ローはありません。

このプロシージャは別個の接続で実行されるため、同期接続でロールバックが実行されても、このプロシージャで実行する操作が失われることはありません。dbmlsync が別個の接続を確立できないと、プロシージャは呼び出されません。

Windows Mobile デバイスのデフォルトでは、同期テーブルは排他モードでロックされます。このため、同期テーブルへのアクセスが必要な場合、このフックは正常に実行されません。また、このフックは、同期テーブルへのアクセスが必要で、ユーザが dbmlsync 拡張オプション LockTables を EXCLUSIVE に設定している場合にも実行できません。「LockTables (lt) 拡張オプション」 214 ページを参照してください。

このプロシージャのアクションは、フックが完了した直後にコミットされます。

参照

- 「イベント・フック・プロシージャ内でのエラーと警告の処理」 253 ページ
- 「sp_hook_dbmlsync_communication_error」 262 ページ
- 「sp_hook_dbmlsync_misc_error」 294 ページ
- 「sp_hook_dbmlsync_sql_error」 309 ページ

例

次のテーブルを使用して、リモート・データベース内のエラーのログを取るとします。

```
CREATE TABLE error_log
(
 pk INTEGER DEFAULT AUTOINCREMENT PRIMARY KEY,
 err_id INTEGER,
 err_msg VARCHAR(10240),
);
```

次の例では、sp_hook_dbmlsync_all_error を設定して、エラー・ログを取ります。

```
CREATE PROCEDURE sp_hook_dbmlsync_all_error()
BEGIN
 DECLARE msg VARCHAR(10240);
 DECLARE id INTEGER;

 // get the error message text
 SELECT value INTO msg
  FROM #hook_dict
  WHERE name ='error message';
```



```
    // get the error id
    SELECT value INTO id
     FROM #hook_dict
     WHERE name = 'error id';

    // log the error information
    INSERT INTO error_log(err_msg, err_id)
     VALUES (msg, id);
   END;
```

可能性のあるエラーの ID 値を表示するには、dbmlsync をテスト実行します。たとえば、dbmlsync が「Mobile Link サーバに接続できません。」というエラーを返すと、sp_hook_dbmlsync_all_error が次のローを error_log に挿入します。

```
1,14173,
'Unable to connect to MobiLink server'
```

これで、「Mobile Link サーバに接続できません。」というエラーとエラー ID 14173 を関連付けることができます。

次の例では、エラー 14173 が発生したときに必ず同期をリトライするようにフックを設定します。

```
CREATE PROCEDURE sp_hook_dbmlsync_all_error()
BEGIN
 IF EXISTS( SELECT value FROM #hook_dict
    WHERE name = 'error id' AND value = '14173' )
 THEN
    UPDATE #hook_dict SET value = '1'
       WHERE name = 'error hook user state';
   END IF;
END;

CREATE PROCEDURE sp_hook_dbmlsync_end()
BEGIN
 IF EXISTS( SELECT value FROM #hook_dict
    WHERE name='error hook user state' AND value='1')
 THEN
    UPDATE #hook_dict SET value = 'sync'
       WHERE name='restart';
   END IF;
END;
```

「sp_hook_dbmlsync_end」 285 ページを参照してください。

sp_hook_dbmlsync_begin

このストアド・プロシージャを使用して、同期処理の開始時にカスタム・アクションを追加します。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、同期処理の開始時に呼び出されます。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「同期イベント・フックの順序」 249 ページ

例

次のテーブルを使用して、リモート・データベース上の同期イベントのログを取るとします。

```
CREATE TABLE SyncLog
(
 "event_id"          integer NOT NULL DEFAULT autoincrement ,
  "event_name"       varchar(128) NOT NULL ,
  "ml_user"          varchar(128) NULL ,
  "event_time"       timestamp NULL,
  "table_name"       varchar(128) NULL ,
  "upsert_count"     varchar(128) NULL ,
  "delete_count"     varchar(128) NULL ,
  "exit_code"        integer NULL ,
  "status_retval"    varchar(128) NULL ,
  "pubs"             varchar(128) NULL ,
  "sync_descr "      varchar(128) NULL ,
   PRIMARY KEY ("event_id"),
)
```

次に、パブリケーションのリストをコンパイルする例を示します。同期処理の初めにパブリケーション・リストなどの同期情報のログを取ります。

```
CREATE PROCEDURE sp_hook_dbmlsync_begin ()
BEGIN

   DECLARE pubs_list VARCHAR(1024);
```



```
    DECLARE temp_str VARCHAR(128);
    DECLARE qry VARCHAR(128);

-- insert publication list into pubs_list
    SELECT LIST(value) INTO pubs_list
     FROM #hook_dict
     WHERE name LIKE 'publication_%';

-- log publication and synchronization information
    INSERT INTO SyncLog(event_name,ml_user,pubs,event_time)
     SELECT 'dbmlsync_begin',#hook_dict.value,pubs_list,CURRENT TIMESTAMP
     FROM #hook_dict
     WHERE name='MobiLink user';
END
```

sp_hook_dbmlsync_communication_error

このストアド・プロシージャは、通信エラーを処理する場合に使用します。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。
error message (in)	エラー・メッセージ・テキスト	これは、dbmlsync ログに表示されるテキストと同じです。
error id (in)	数値	メッセージをユニークに識別する ID。このローを使用すると、エラー・メッセージ・テキストが変更されたときに、エラー・メッセージを識別できます。
error hook user state (in|out)	整数	この値はフックによって設定して、今後の呼び出しに対するステータス情報を、sp_hook_dbmlsync_all_error、sp_hook_dbmlsync_communication_error、sp_hook_dbmlsync_misc_error、sp_hook_dbmlsync_sql_error、または sp_hook_dbmlsync_end フックに渡すことができます。これらのフックの 1 つが最初に呼び出されるとき、ローの値は 0 です。フックがローの値を変更した場合は、次回のフックの呼び出しには新しい値が使用されます。 このフックを使用して、sp_hook_dbmlsync_end フックにステータス情報を渡す場合は、_end フックにより同期のリトライなどのアクションを実行することができます。
stream error code (in)	整数	ストリームによってレポートされるエラー。

名前	値	説明
system error code (in)	整数	システム固有のエラー・コード。

備考

同期を開始する前の起動中にエラーが発生した場合、#hook_dict 内の Mobile Link ユーザとスクリプト・バージョンのエントリは空の文字列に設定され、#hook_dict テーブルで設定される publication_*n* ローはありません。

dbmlsync と Mobile Link サーバ間で通信エラーが発生した場合、このフックを使用すると、ストリーム固有のエラー情報にアクセスすることができます。

stream error code パラメータは、通信エラーのタイプを示す整数です。

エラー・コード値のリストについては、「Mobile Link 通信エラー・メッセージ」 『エラー・メッセージ』を参照してください。

このプロシージャは別個の接続で実行されるため、同期接続でロールバックが実行されても、このプロシージャで実行する操作が失われることはありません。dbmlsync が別個の接続を確立できないと、プロシージャは呼び出されません。

Windows Mobile デバイスのデフォルトでは、同期テーブルは排他モードでロックされます。このため、同期テーブルへのアクセスが必要な場合、このフックは正常に実行されません。また、このフックは、同期テーブルへのアクセスが必要で、ユーザが dbmlsync 拡張オプション LockTables を EXCLUSIVE に設定している場合にも実行できません。「LockTables (lt) 拡張オプション」 214 ページを参照してください。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「イベント・フック・プロシージャ内でのエラーと警告の処理」 253 ページ
- 「sp_hook_dbmlsync_all_error」 257 ページ
- 「sp_hook_dbmlsync_misc_error」 294 ページ
- 「sp_hook_dbmlsync_sql_error」 309 ページ

例

次のテーブルを使用して、リモート・データベース内の通信エラーのログを取るとします。

```
CREATE TABLE communication_error_log
(
  error_msg VARCHAR(10240),
  error_code VARCHAR(128)
);
```

次の例では、sp_hook_dbmlsync_communication_error を設定して、通信エラーのログを取ります。

```
CREATE PROCEDURE sp_hook_dbmlsync_communication_error()
BEGIN
 DECLARE msg VARCHAR(255);
 DECLARE code INTEGER;

 // get the error message text
 SELECT value INTO msg
```

```
  FROM #hook_dict
  WHERE name ='error message';

 // get the error code
 SELECT value INTO code
  FROM #hook_dict
  WHERE name = 'stream error code';

 // log the error information
 INSERT INTO communication_error_log(error_code,error_msg)
  VALUES (code,msg);
END
```


sp_hook_dbmlsync_delay

このストアド・プロシージャを使用して、同期が行われるタイミングを制御します。

#hook_dict テーブルのロー

名前	値	説明
delay duration (in|out)	秒数	プロシージャが delay duration の値を 0 に設定すると、dbmlsync の同期が進行します。delay duration が 0 以外の値の場合は、遅延フックが再び呼び出されるまでの秒数を示します。
maximum accumulated delay (in|out)	秒数	最大累積遅延は、各同期前の最大秒数を指定します。dbmlsync は、最後に実行された同期以降の遅延フックへのすべての呼び出しによって生じた合計遅延時間を追跡します。dbmlsync が実行を開始してから同期が行われないと、dbmlsync の起動時間から合計遅延時間が計算されます。合計遅延時間が Maximum Accumulated delay 値を上回ると、遅延フックを呼び出さずに同期が開始されます。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、同期処理の開始時の、**sp_hook_dbmlsync_begin** の前に呼び出されます。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「同期イベント・フックの順序」 249 ページ
- 「イベント・フックを使用した同期の開始」 129 ページ
- 「sp_hook_dbmlsync_download_end」 271 ページ

例

次のテーブルを使用して、リモート・データベース上の注文のログを取るとします。

```
CREATE TABLE OrdersTable(
  "id" INTEGER PRIMARY KEY DEFAULT AUTOINCREMENT,
  "priority" VARCHAR(128)
);
```

次に、最大累積遅延 (1 時間) の間、同期を遅らせる例を示します。10 秒ごとにフックが呼び出され、OrdersTable 内に優先度の高いローがないかをチェックします。優先度の高いローが存在する場合、遅延期間は 0 に設定して同期処理を開始します。

```
CREATE PROCEDURE sp_hook_dbmlsync_delay()
BEGIN
     -- Set the maximum delay between synchronizations
    -- or before the first synchronization starts to 1 hour
    UPDATE #hook_dict SET value = '3600'  // 3600 seconds
     WHERE name = 'maximum accumulated delay';

    -- check if a high priority order exists in OrdersTable
    IF EXISTS (SELECT * FROM OrdersTable where priority='high') THEN
     -- start the synchronization to process the high priority row
     UPDATE #hook_dict
      SET value = '0'
      WHERE name='delay duration';
    ELSE
     -- set the delay duration to call this procedure again
     -- following a 10 second delay
     UPDATE #hook_dict
      SET value = '10'
      WHERE name='delay duration';
    END IF;
END;
```

sp_hook_dbmlsync_end フックでは、優先度の高いローを処理済みとしてマークすることができます。

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_end()
    BEGIN
       IF EXISTS( SELECT value FROM #hook_dict
     WHERE name = 'Upload status'
     AND value = 'committed' ) THEN
     UPDATE OrderTable SET priority = 'high-processed'
    WHERE priority = 'high';
       END IF;
    END;
```

「sp_hook_dbmlsync_end」 285 ページを参照してください。

sp_hook_dbmlsync_download_begin

このストアド・プロシージャを使用して、同期処理のダウンロード処理開始時にカスタム・アクションを追加します。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、同期処理のダウンロード処理開始時に呼び出されます。

ダウンロードがコミットまたはロールバックされると、このプロシージャのアクションがコミットまたはロールバックされます。

参照

- 「同期イベント・フックの順序」 249 ページ

例

次のテーブルを使用して、リモート・データベース上の同期イベントのログを取るとします。

```
CREATE TABLE SyncLog
(
 "event_id"         INTEGER NOT NULL DEFAULT AUTOINCREMENT ,
  "event_name"       VARCHAR(128) NOT NULL ,
  "ml_user"          VARCHAR(128) NULL ,
  "event_time"        TIMESTAMP NULL,
  "table_name"        VARCHAR(128) NULL ,
  "upsert_count"      VARCHAR(128) NULL ,
  "delete_count"      VARCHAR(128) NULL ,
  "exit_code"        INTEGER NULL ,
  "status_retval"     VARCHAR(128) NULL ,
  "pubs"             VARCHAR(128) NULL ,
  "sync_descr "       VARCHAR(128) NULL ,
   PRIMARY KEY ("event_id"),
);
```

次に、パブリケーションのリストをコンパイルする例を示します。同期のダウンロード処理の初めにパブリケーション・リストなどの同期情報のログを取ります。

```
CREATE PROCEDURE sp_hook_dbmlsync_download_begin ()
BEGIN

    DECLARE pubs_list VARCHAR(1024);
    DECLARE temp_str VARCHAR(128);
    DECLARE qry VARCHAR(128);

-- insert publication list into pubs_list
    SELECT LIST(value) INTO pubs_list
     FROM #hook_dict
     WHERE name LIKE 'publication_%';

-- log publication and synchronization information
    INSERT INTO SyncLog(event_name,ml_user,pubs,event_time)
     SELECT 'dbmlsync_download_begin',#hook_dict.value,
      pubs_list,CURRENT TIMESTAMP
     FROM #hook_dict
     WHERE name='MobiLink user';
END;
```


sp_hook_dbmlsync_download_com_error (旧式)

Mobile Link サーバによって送信されたダウンロードの読み込み中に通信エラーが発生した場合は、このストアド・プロシージャを使用してカスタム・アクションを追加します。

このフックは使用されなくなりました。「イベント・フック・プロシージャ内でのエラーと警告の処理」 253 ページを参照してください。

#hook_dict テーブルのロー

名前	値	説明
table name (in)	テーブル名	エラーの発生時に操作中だったテーブル。dbmlsync がテーブルを識別できない場合、値は空の文字列になります。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、同期のダウンロード・フェーズ中に通信エラーが検出されると呼び出されます。その場合、ダウンロードは終了します。

このプロシージャは別個の接続で実行されるため、障害のログを取ることができます。それ以外の場合、ログのアクションは同期アクションとともにロールバックされます。dbmlsync が別個の接続を確立できないと、プロシージャは呼び出されません。

Windows Mobile デバイスのデフォルトでは、同期テーブルは排他モードでロックされます。このため、同期テーブルへのアクセスが必要な場合、このフックは正常に実行されません。また、このフックは、同期テーブルへのアクセスが必要で、ユーザが dbmlsync 拡張オプション LockTables を EXCLUSIVE に設定している場合にも実行できません。「LockTables (lt) 拡張オプション」 214 ページを参照してください。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「同期イベント・フックの順序」 249 ページ

例

次のテーブルを使用して、通信エラーのログを取るとします。

```
CREATE TABLE SyncLogComErrorTable
(
  " user_name "  VARCHAR(255) NOT NULL ,
  " event_time "   TIMESTAMP NOT NULL ,
);
```

次に、Mobile Link サーバによって送信されたダウンロードを読み込み中に通信エラーが発生した場合に、Mobile Link ユーザと現在のタイム・スタンプのログを取る例を示します。この情報は、リモート・データベースの SyncLogComErrorTable テーブルに格納されます。

```
CREATE PROCEDURE sp_hook_dbmlsync_download_com_error ()
BEGIN
 INSERT INTO SyncLogComErrorTable (user_name, event_time)
  SELECT #hook_dict.value, CURRENT TIMESTAMP
  FROM #hook_dict
  WHERE name = 'MobiLink user';
END;
```


sp_hook_dbmlsync_download_end

このストアド・プロシージャを使用して、同期処理のダウンロード処理終了時にカスタム・アクションを追加します。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、同期処理のダウンロード処理終了時に呼び出されます。

ダウンロードがコミットまたはロールバックされると、このプロシージャのアクションがコミットまたはロールバックされます。

参照

- 「同期イベント・フックの順序」 249 ページ
- 「イベント・フックを使用した同期の開始」 129 ページ
- 「sp_hook_dbmlsync_delay」 265 ページ

例

次のテーブルを使用して、リモート・データベース上の同期イベントのログを取るとします。

```
CREATE TABLE SyncLog
(
 "event_id"         INTEGER NOT NULL DEFAULT AUTOINCREMENT ,
  "event_name"       VARCHAR(128) NOT NULL ,
  "ml_user"          VARCHAR(128) NULL ,
  "event_time"        TIMESTAMP NULL,
  "table_name"        VARCHAR(128) NULL ,
  "upsert_count"      VARCHAR(128) NULL ,
  "delete_count"      VARCHAR(128) NULL ,
  "exit_code"        INTEGER NULL ,
  "status_retval"     VARCHAR(128) NULL ,
  "pubs"             VARCHAR(128) NULL ,
  "sync_descr "       VARCHAR(128) NULL ,
   PRIMARY KEY ("event_id"),
)
```

次に、パブリケーションのリストをコンパイルする例を示します。同期のダウンロード処理の終りにパブリケーション・リストなどの同期情報のログを取ります。

```
CREATE PROCEDURE sp_hook_dbmlsync_download_end ()
BEGIN

    DECLARE pubs_list VARCHAR(1024);
    DECLARE temp_str VARCHAR(128);
    DECLARE qry VARCHAR(128);

-- insert publication list into pubs_list
    SELECT LIST(value) INTO pubs_list
     FROM #hook_dict
     WHERE name LIKE 'publication_%';

-- log publication and synchronization information
    INSERT INTO SyncLog(event_name,ml_user,pubs,event_time)
     SELECT 'dbmlsync_download_end',#hook_dict.value,
      pubs_list,CURRENT TIMESTAMP
     FROM #hook_dict
     WHERE name='MobiLink user';
END
```


sp_hook_dbmlsync_download_fatal_sql_error (旧式)

データベース・エラーによって同期ダウンロードのロールバックが発生すると動作します。

このフックは使用されなくなりました。「イベント・フック・プロシージャ内でのエラーと警告の処理」 253 ページを参照してください。

#hook_dict テーブルのロー

名前	値	説明
table name (in)	テーブル名	エラーの発生時に操作中だったテーブル。dbmlsync がテーブルを識別できない場合、値は空の文字列になります。
SQL error code (in)	SQL エラー・コード	操作が失敗した時にデータベースから返される SQL エラー・コードを識別します。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、同期のダウンロード・フェーズ中に通信エラーが検出されると呼び出されます。これが発生するのは、無視できない SQL エラーが発生した場合、または sp_hook_dbmlsync_download_SQL_error フックがすでに呼び出されていて、エラーを無視しないように選択されている場合です。

このプロシージャは別個の接続で実行されるため、障害のログを取ることができます。それ以外の場合、ログのアクションは同期アクションとともにロールバックされます。dbmlsync が別個の接続を確立できないと、プロシージャは呼び出されません。

Windows Mobile デバイスのデフォルトでは、同期テーブルは排他モードでロックされます。このため、同期テーブルへのアクセスが必要な場合、このフックは正常に実行されません。また、このフックは、同期テーブルへのアクセスが必要で、ユーザが dbmlsync 拡張オプション LockTables を EXCLUSIVE に設定している場合にも実行できません。「LockTables (lt) 拡張オプション」 214 ページを参照してください。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「同期イベント・フックの順序」 249 ページ
- 「sp_hook_dbmlsync_download_sql_error (旧式)」 279 ページ

例

次のテーブルを使用して、SQL エラーのログを取るとします。

```
CREATE TABLE "DBA"."SyncLogComErrorTable"
(
  " error_code "      VARCHAR(255) NOT NULL ,
  " event_time "      TIMESTAMP NOT NULL ,
);
```

次に、ダウンロードの読み込み中に SQL エラーが発生した場合に、SQL エラー・コードと現在のタイム・スタンプのログを取る例を示します。この情報は、リモート・データベースの SyncLogSQLErrorTable に格納されます。

```
CREATE PROCEDURE sp_hook_dbmlsync_download_fatal_sql_error ()
BEGIN
 INSERT INTO SyncLogSQLErrorTable (error_code, event_time)
  SELECT #hook_dict.value, CURRENT TIMESTAMP
  FROM #hook_dict
  WHERE name = 'SQL error code';
END;
```


sp_hook_dbmlsync_download_log_ri_violation

ダウンロード・プロセスの参照整合性違反のログを取ります。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
foreign key table (in)	テーブル名	フック呼び出し対象の外部キー・カラムを含むテーブル。
primary key table (in)	テーブル名	フック呼び出し対象の外部キーが参照するテーブル。
role name (in)	ロール名	フック呼び出し対象の外部キーのロール名。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

ダウンロード内のローがリモート・データベース上の外部キー関係に違反すると、ダウンロード参照整合性違反が発生します。このフックを使用すると、発生した参照整合性違反のログを取り、後でその原因を調べることができます。

ダウンロードが完了すると、コミットされる前に、dbmlsync は参照整合性違反があるかどうかをチェックします。参照整合性違反が見つかると、参照整合性違反を含む外部キーを識別して、sp_hook_dbmlsync_download_log_ri_violation を呼び出します (実装されている場合)。次に、sp_hook_dbmlsync_download_ri_conflict を呼び出します (実装されている場合)。それでも矛盾がある場合、dbmlsync は外部キー制約に違反するローを削除します。参照整合性違反を含む残りの外部キーに対して、このプロセスが繰り返されます。

このフックは、現在同期中のテーブルに関する参照整合性違反がある場合にのみ呼び出されます。同期中ではないテーブルに関する参照整合性違反がある場合、このフックは呼び出されず、同期が失敗します。

このフックは、dbmlsync がダウンロードに使用する接続とは別の接続で呼び出されます。このフックが使用する接続の独立性レベルは 0 のため、ダウンロードから適用されていてまだコミットされていないローを判別できます。フックのアクションは完了直後にコミットされるので、ダウンロードがコミットされるかロールバックされるかに関係なく、このフックによる変更は保存されます。

Windows Mobile デバイスのデフォルトでは、同期テーブルは排他モードでロックされます。このため、同期テーブルへのアクセスが必要な場合、このフックは正常に実行されません。また、このフックは、同期テーブルへのアクセスが必要で、ユーザが dbmlsync 拡張オプション LockTables を EXCLUSIVE に設定している場合にも実行できません。「LockTables (lt) 拡張オプション」 214 ページを参照してください。

参照整合性違反の問題を解決するために、このフックを使用しないでください。このフックはロギングにのみ使用してください。参照整合性違反を解決するには、sp_hook_dbmlsync_download_ri_violation を使用します。

参照

- 「sp_hook_dbmlsync_download_ri_violation」 277 ページ
- 「同期イベント・フックの順序」 249 ページ

例

次のテーブルを使用して、参照整合性違反のログを取るとします。

```
CREATE TABLE DBA.LogRIViolationTable
(
   entry_time  TIMESTAMP,
   pk_table  VARCHAR( 255 ),
   fk_table  VARCHAR( 255 ),
   role_name  VARCHAR( 255 )
);
```

次に、リモート・データベース上で参照整合性違反が検出されたときに、外部キー・テーブル名、プライマリ・キー・テーブル名、役割名のログを取る例を示します。この情報は、リモート・データベースの LogRIErrorTable に格納されます。

```
CREATE PROCEDURE sp_hook_dbmlsync_download_log_ri_violation()
BEGIN
 INSERT INTO DBA.LogRIViolationTable VALUES(
 CURRENT_TIMESTAMP,
 (SELECT value FROM #hook_dict WHERE name = 'Primary key table'),
 (SELECT value FROM #hook_dict WHERE name = 'Foreign key table'),
 (SELECT value FROM #hook_dict WHERE name = 'Role name' ) );
END;
```


sp_hook_dbmlsync_download_ri_violation

ダウンロード・プロセスの参照整合性違反を解決できます。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
foreign key table (in)	テーブル名	フック呼び出し対象の外部キー・カラムを含むテーブル。
primary key table (in)	テーブル名	フック呼び出し対象の外部キーが参照するテーブル。
role name (in)	ロール名	フック呼び出し対象の外部キーのロール名。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

ダウンロード内のローがリモート・データベース上の外部キー関係に違反すると、ダウンロード参照整合性違反が発生します。このフックを使用すると、dbmlsync が競合の原因であるローを削除する前に、参照整合性違反を解決できます。

ダウンロードが完了すると、コミットされる前に、dbmlsync は参照整合性違反があるかどうかをチェックします。参照整合性違反が見つかると、参照整合性違反を含む外部キーを識別して、sp_hook_dbmlsync_download_log_ri_violation を呼び出します (実装されている場合)。次に、sp_hook_dbmlsync_download_ri_conflict を呼び出します (実装されている場合)。それでも競合が解決されない場合は、dbmlsync がそのローを削除します。参照整合性違反を含む残りの外部キーに対して、このプロセスが繰り返されます。

このフックは、現在同期中のテーブルに関する参照整合性違反がある場合にのみ呼び出されます。同期中ではないテーブルに関する参照整合性違反がある場合、このフックは呼び出されず、同期が失敗します。

このフックは、dbmlsync がダウンロードに使用する接続と同じ接続で呼び出されます。データベースでデータの不整合が発生する可能性があるため、このフックに明示的または暗黙的なコミットが含まれないようにしてください。ダウンロードがコミットまたはロールバックされると、このフックのアクションがコミットまたはロールバックされます。

他のフック・アクションとは異なり、このフック中に実行される操作は次の同期中にアップロードされません。

参照

- 「sp_hook_dbmlsync_download_log_ri_violation」 275 ページ

例

この例は、次に示す Department テーブルと Employee テーブルを使用します。

```
CREATE TABLE Department(
 "department_id"  INTEGER primary key
);

CREATE TABLE Employee(
  "employee_id"      INTEGER PRIMARY KEY,
  "department_id" INTEGER,
  FOREIGN KEY EMPLOYEE_FK1 (department_id) REFERENCES Department
);
```

次の sp_hook_dbmlsync_download_ri_violation の定義は、Department テーブルと Employee テーブル間の参照整合性違反をクリーンアップします。この定義によって、外部キーの役割名が検証され、欠落している department_id の値が Department テーブルに挿入されます。

```
CREATE PROCEDURE sp_hook_dbmlsync_download_ri_violation()
BEGIN

IF EXISTS (SELECT * FROM #hook_dict WHERE name = 'role name'
 AND value = 'EMPLOYEE_FK1') THEN

 -- update the Department table with missing department_id values
 INSERT INTO Department
  SELECT distinct department_id FROM Employee
  WHERE department_id NOT IN (SELECT department_id FROM Department)
END IF;
END
```


sp_hook_dbmlsync_download_sql_error (旧式)

Mobile Link サーバによって送信されたダウンロードの適用中に発生したデータベース・エラーを処理します。

このフックは使用されなくなりました。「イベント・フック・プロシージャ内でのエラーと警告の処理」 253 ページを参照してください。

#hook_dict テーブルのロー

名前	値	説明
table name (in)	テーブル名	エラーの発生時に操作中だったテーブル。dbmlsync がテーブルを識別できない場合、値は空の文字列になります。
continue (in|out)	**true** | **false**	エラーを無視して同期を継続するかどうかを示します。sp_hook_dbmlsync_download_fatal_sql_error フックを呼び出し、同期を停止するには、このパラメータを **false** に設定してください。このパラメータを **true** に設定すると、dbmlsync はエラーを無視し、同期を続行します。この結果、データが失われることがあります。
SQL error code (in)	SQL エラー・コード	操作が失敗した時にデータベースから返される SQL エラー・コードを識別します。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、同期のダウンロード・フェーズ中にデータベース・エラーが検出されたときに呼び出されます。このプロシージャは、エラーを無視して同期を継続できる場合にのみ呼び出されます。致命的なエラーの場合は、sp_hook_dbmlsync_download_fatal_SQL_error プロシージャが呼び出されます。

警告
continue を True に設定すると、dbmlsync はデータベース・エラーを無視し、同期を続行します。失敗した操作をもう一度実行することはありません。このため、ダウンロードの一部またはすべてが失われることがあります。失われるデータの量は、発生したエラーの種類、発生したタイミング、リカバリするためにフックが使用される段階によって異なります。どのデータが失われるかを予測するのは非常に困難であるため、この機能を使用するときは最大限の注意を払ってください。ほとんどの場合、SQL エラーが発生した後は、処理を続行しないようにしてください。

ダウンロードがコミットまたはロールバックされると、このプロシージャのアクションがコミットまたはロールバックされます。

参照

- 「同期イベント・フックの順序」 249 ページ
- 「sp_hook_dbmlsync_download_fatal_sql_error (旧式)」 273 ページ

sp_hook_dbmlsync_download_table_begin

このストアド・プロシージャを使用して、各テーブルがダウンロードされる直前にカスタム・アクションを追加します。

#hook_dict テーブルのロー

名前	値	説明
table name (in)	テーブル名	操作が適用される予定のテーブル。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、ダウンロードされた操作がテーブルに適用される直前にテーブルごとに呼び出されます。ダウンロードがコミットまたはロールバックされると、このプロシージャのアクションがコミットまたはロールバックされます。

参照

- 「同期イベント・フックの順序」 249 ページ

例

次のテーブルを使用して、リモート・データベース上の同期イベントのログを取るとします。

```
CREATE TABLE SyncLog
(
 "event_id"         INTEGER NOT NULL DEFAULT AUTOINCREMENT ,
  "event_name"       VARCHAR(128) NOT NULL ,
  "ml_user"          VARCHAR(128) NULL ,
  "event_time"        TIMESTAMP NULL,
  "table_name"        VARCHAR(128) NULL ,
  "upsert_count"      VARCHAR(128) NULL ,
  "delete_count"      VARCHAR(128) NULL ,
  "exit_code"        INTEGER NULL ,
  "status_retval"     VARCHAR(128) NULL ,
  "pubs"             VARCHAR(128) NULL ,
  "sync_descr "       VARCHAR(128) NULL ,
   PRIMARY KEY ("event_id"),
);
```

次に、テーブルがダウンロードされる直前に Mobile Link ユーザ、テーブル名、現在のタイムスタンプのログを取る例を示します。

```
CREATE PROCEDURE sp_hook_dbmlsync_download_table_begin()
BEGIN
    DECLARE tbl VARCHAR(255);

   -- load the table name from #hook_dict
  SELECT #hook_dict.value
   INTO tbl
   FROM #hook_dict
   WHERE #hook_dict.name = 'table name';

  INSERT INTO SyncLog (event_name, ml_user, table_name
    ,event_time)
   SELECT 'download_table_begin', #hook_dict.value, tbl
    ,CURRENT TIMESTAMP
   FROM #hook_dict
   WHERE name = 'MobiLink user' ;
END;
```


sp_hook_dbmlsync_download_table_end

このストアド・プロシージャを使用して、各テーブルがダウンロードされた直後にカスタム・アクションを追加します。

#hook_dict テーブルのロー

名前	値	説明
table name (in)	テーブル名	操作が直前に適用されたテーブル。
delete count (in)	ローの数	ダウンロードによってこのテーブルから削除されたローの数。
upsert count (in)	ローの数	ダウンロードによってこのテーブルで更新または挿入されたローの数。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、ダウンロードでの操作がすべてテーブルに適用された直後に呼び出されます。

ダウンロードがコミットまたはロールバックされると、このプロシージャのアクションがコミットまたはロールバックされます。

参照

- 「同期イベント・フックの順序」 249 ページ

例

次のテーブルを使用して、リモート・データベース上の同期イベントのログを取るとします。

```
CREATE TABLE SyncLog
(
 "event_id"         INTEGER NOT NULL DEFAULT AUTOINCREMENT ,
  "event_name"       VARCHAR(128) NOT NULL ,
  "ml_user"          VARCHAR(128) NULL ,
  "event_time"        TIMESTAMP NULL,
  "table_name"        VARCHAR(128) NULL ,
  "upsert_count"      VARCHAR(128) NULL ,
  "delete_count"      VARCHAR(128) NULL ,
```

```
    "exit_code"          INTEGER NULL ,
    "status_retval"      VARCHAR(128) NULL ,
    "pubs"               VARCHAR(128) NULL ,
    "sync_descr "        VARCHAR(128) NULL ,
    PRIMARY KEY ("event_id"),
);
```

次に、テーブルがダウンロードされた直後に Mobile Link ユーザ、テーブル名、挿入または更新されたローの数のログを取る例を示します。

```
CREATE PROCEDURE sp_hook_dbmlsync_download_table_end()
BEGIN
    -- declare variables
    DECLARE tbl VARCHAR(255);
    DECLARE upsertCnt VARCHAR(255);
    DECLARE deleteCnt VARCHAR(255);

    -- load the table name from #hook_dict
    SELECT #hook_dict.value
     INTO tbl
     FROM #hook_dict
     WHERE #hook_dict.name = 'table name';

    -- load the upsert count from #hook_dict
    SELECT #hook_dict.value
     INTO upsertCnt
     FROM #hook_dict
     WHERE #hook_dict.name = 'upsert count';

    -- load the delete count from #hook_dict
    SELECT #hook_dict.value
     INTO deleteCnt
     FROM #hook_dict
     WHERE #hook_dict.name = 'delete count';

    INSERT INTO SyncLog (event_name, ml_user, table_name,
      upsert_count, delete_count, event_time)
     SELECT 'download_table_end', #hook_dict.value, tbl,
      upsertCnt, deleteCnt, CURRENT TIMESTAMP
     FROM #hook_dict
     WHERE name = 'MobiLink user' ;
END;
```


sp_hook_dbmlsync_end

このストアド・プロシージャを使用して、同期が完了する直前にカスタム・アクションを追加します。

#hook_dict テーブルのロー

名前	値	説明
restart (out)	**sync** | **download** | **none**	**sync** に設定すると、dbmlsync は完了した同期のリトライを行います。**true** も同じですが、廃止され、値 **sync** に置き換えられました。 **none** (デフォルト) に設定すると、dbmlsync はコマンド・ライン引数の指定に従って、停止するか、または再起動します。**false** も同じですが、廃止され、値 **none** に置き換えられました。 **download** に設定した場合、restartable download パラメータが **true** であると、dbmlsync は失敗したダウンロードを再起動します。
exit code (in)	数値	0 (デフォルト) 以外の値が設定された場合は、同期エラーを表します。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。

名前	値	説明
upload status (in)	**not sent** | **committed** | **failed**	dbmlsync がアップロードの受信確認を行おうとしたときに、Mobile Link サーバから返されるステータスを指定します。次のいずれかのステータスになります。 ● **not sent** - エラーが原因で、または要求された同期で不要だったので、アップグレードは Mobile Link サーバに送信されませんでした。これは、ダウンロード専用の同期、再起動したダウンロード、ファイルベースのダウンロードなどで発生します。 ● **committed** - Mobile Link サーバがアップロードを受信し、コミットしました。 ● **failed** - Mobile Link サーバは、アップロードをコミットしませんでした。トランザクション単位のアップロードについては、すべてではないものの、一部のトランザクションが、サーバによって正しくアップロードされ、受信確認されたときは、アップロード・ステータスは 'failed' になります。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。
restartable download (in)	**true**|**false**	**true** の場合、現在の同期のダウンロードが失敗しており、再起動できます。**false** の場合、ダウンロードが正常に行われたか、再起動できません。
restartable download size (in)	整数	restartable download パラメータが **true** である場合、このパラメータはダウンロードが失敗する前に受信したバイト数を示します。restartable download が **false** の場合、このパラメータの値は無効です。
error hook user state (in)	整数	この値にはエラーについての情報が含まれ、フック sp_hook_dbmlsync_all_error、sp_hook_dbmlsync_communication_error、sp_hook_dbmlsync_misc_error、または sp_hook_dbmlsync_sql_error から送信できます

備考

この名前のプロシージャが存在する場合、各同期の最後に呼び出されます。

このプロシージャのアクションは、実行直後にコミットされます。

再起動パラメータを常に **sync** に設定するように sp_hook_dbmlsync_end フックが定義されており、ユーザが dbmlsync のコマンド・ラインで -n pub1, -n pub2, ... といった形式で複数のパブリケーションを指定している場合、dbmlsync は最初のパブリケーションを繰り返し同期し、2 番目のパブリケーションを同期しません。

参照

- 「dbmlsync のフックの概要」 249 ページ
- 「同期イベント・フックの順序」 249 ページ
- 「失敗したダウンロードの再開」 『Mobile Link - サーバ管理』
- 「イベント・フック・プロシージャ内でのエラーと警告の処理」 253 ページ

例

次の例では、現在の同期のダウンロードが失敗して再起動が可能な場合、ダウンロードは手動で再起動されます。

```
CREATE PROCEDURE sp_hook_dbmlsync_end()
BEGIN
  -- Restart the download if the download for the current sync
  --  failed and can be restarted
  IF EXISTS (SELECT * FROM #hook_dict
    WHERE name = 'restartable download' AND value='true')
      THEN
   UPDATE #hook_dict SET value ='download' WHERE name='restart';
  END IF;
END;
```


sp_hook_dbmlsync_log_rescan

このストアド・プロシージャを使用して、再スキャンがいつ必要かプログラム的に決定できます。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
discarded storage (in)	数値	最後の同期の後で破棄されるメモリのバイト数。
rescan (in|out)	**true** | **false**	フックによって True と設定されている場合、dbmlsync は次の同期の前に完全な再スキャンを行います。エントリ時には、この値は False に設定されます。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

コマンド・ラインで複数の -n オプションを指定すると、メモリ破棄の原因となる断片化が dbmlsync で起きる可能性があります。破棄されたメモリは、データベース・トランザクション・ログの再スキャンによってリカバリできます。このフックにより、dbmlsync を使用してデータベース・トランザクション・ログの再スキャンを行いメモリを回復させるかどうかを決定できます。

再スキャンを強制する他の条件を満たさない場合、このフックは sp_hook_dbmlsync_process_exit_code フックの直後に呼び出されます。

参照

- 「HoverRescanThreshold (hrt) 拡張オプション」 210 ページ

例

次の例では、破棄された記憶領域が 1000 バイトを超える場合に #hook_dict テーブル内の再スキャン・フィールドを TRUE に設定します。

```
CREATE PROCEDURE sp_hook_dbmlsync_log_rescan ()
BEGIN
 IF EXISTS(SELECT * FROM #hook_dict
```



```
 WHERE name = 'Discarded storage' AND value>1000)
 THEN
  UPDATE #hook_dict SET value ='true' WHERE name='Rescan';
 END IF;
END;
```

sp_hook_dbmlsync_logscan_begin

このストアド・プロシージャを使用して、アップロード用にトランザクション・ログがスキャンされる直前にカスタム・アクションを追加します。

#hook_dict テーブルのロー

名前	値	説明
starting log offset_*n* (in)	数値	スキャンの開始位置を示すログ・オフセット値。アップロードされるパブリケーションごとに 1 つの値があります。*n* の番号は 0 から始まります。この値は、Publication-*n* に一致します。たとえば、log offset_0 は publication_0 のオフセットです。
log scan retry (in)	**true** | **false**	この同期でトランザクション・ログが初めてスキャンされる場合、この値は False、それ以外の場合は True。Mobile Link サーバと dbmlsync でスキャン開始位置の情報が異なっている場合、ログは 2 回スキャンされます。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、dbmlsync がトランザクション・ログをスキャンしてアップロードをアセンブルする直前に呼び出されます。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「同期イベント・フックの順序」 249 ページ

例

次のテーブルを使用して、リモート・データベース上の同期イベントのログを取るとします。


```
CREATE TABLE SyncLog
(
 "event_id"          INTEGER NOT NULL DEFAULT AUTOINCREMENT ,
   "event_name"       VARCHAR(128) NOT NULL ,
   "ml_user"          VARCHAR(128) NULL ,
   "event_time"        TIMESTAMP NULL,
   "table_name"        VARCHAR(128) NULL ,
   "upsert_count"      VARCHAR(128) NULL ,
   "delete_count"      VARCHAR(128) NULL ,
   "exit_code"         INTEGER NULL ,
   "status_retval"     VARCHAR(128) NULL ,
   "pubs"              VARCHAR(128) NULL ,
   "sync_descr "       VARCHAR(128) NULL ,
    PRIMARY KEY ("event_id"),
);
```

次に、アップロードのためにトランザクション・ログがスキャンされる直前に Mobile Link ユーザと現在のタイムスタンプのログを取る例を示します。

```
CREATE PROCEDURE sp_hook_dbmlsync_logscan_begin ()
BEGIN
 -- log the synchronization event
 INSERT INTO SyncLog (event_name, ml_user,event_time)
  SELECT 'logscan_begin', #hook_dict.value, CURRENT TIMESTAMP
  FROM #hook_dict
  WHERE name = 'MobiLink user' ;
END;
```

sp_hook_dbmlsync_logscan_end

このストアド・プロシージャを使用して、アップロード用にトランザクション・ログがスキャンされた直後にカスタム・アクションを追加します。

#hook_dict テーブルのロー

名前	値	説明
ending log offset (in)	数値	スキャンの終了位置を示すログ・オフセット値。
starting log offset_*n* (in)	数値	同期する各サブスクリプションの初期進行値。*n* 値は、Publication_*n* の値に対応します。たとえば、Starting log offset_1 は Publication_1 のオフセットです。
log scan retry (in)	**true** | **false**	この同期でトランザクション・ログが初めてスキャンされる場合、この値は False、それ以外の場合は True。Mobile Link サーバと dbmlsync でスキャン開始位置の情報が異なっている場合、ログは 2 回スキャンされます。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、dbmlsync がアップロードをアセンブルするためにトランザクション・ログをスキャンした直後に呼び出されます。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「同期イベント・フックの順序」 249 ページ

例

次のテーブルを使用して、リモート・データベース上の同期イベントのログを取るとします。


```
CREATE TABLE SyncLog
(
 "event_id"         INTEGER NOT NULL DEFAULT AUTOINCREMENT ,
  "event_name"       VARCHAR(128) NOT NULL ,
  "ml_user"          VARCHAR(128) NULL ,
  "event_time"        TIMESTAMP NULL,
  "table_name"        VARCHAR(128) NULL ,
  "upsert_count"      VARCHAR(128) NULL ,
  "delete_count"      VARCHAR(128) NULL ,
  "exit_code"         INTEGER NULL ,
  "status_retval"     VARCHAR(128) NULL ,
  "pubs"             VARCHAR(128) NULL ,
  "sync_descr "       VARCHAR(128) NULL ,
   PRIMARY KEY ("event_id"),
);
```

次に、アップロードのためにトランザクション・ログがスキャンされた直後に Mobile Link ユーザと現在のタイムスタンプのログを取る例を示します。#hook_dict の log scan retry パラメータは、トランザクション・ログが複数回スキャンされるかどうかを示します。

```
CREATE PROCEDURE sp_hook_dbmlsync_logscan_end ()
BEGIN

 DECLARE scan_retry VARCHAR(128);

 -- load the scan retry parameter from #hook_dict
 SELECT #hook_dict.value
  INTO scan_retry
  FROM #hook_dict
  WHERE #hook_dict.name = 'log scan retry';

 -- Determine if the log is being rescanned
 -- and log the synchronization event

 IF scan_retry='true' THEN
   INSERT INTO SyncLog (event_name, ml_user,event_time,sync_descr)
    SELECT 'logscan_end', #hook_dict.value, CURRENT TIMESTAMP,
     'Transaction log rescanned'
    FROM #hook_dict
    WHERE name = 'MobiLink user' ;
 ELSE
   INSERT INTO SyncLog (event_name, ml_user,event_time,sync_descr)
    SELECT 'logscan_end', #hook_dict.value, CURRENT TIMESTAMP,
     'Transaction log scanned normally'
    FROM #hook_dict
    WHERE name = 'MobiLink user' ;
 END IF;
END;
```

sp_hook_dbmlsync_misc_error

このストアド・プロシージャを使用して、データベース・エラーまたは通信エラーに分類されない dbmlsync エラーを処理します。たとえば、sp_hook_dbmlsync_misc_error フックを実装すると、特定のエラーが発生した場合に、ログを取ったり特定のアクションを実行したりすることができます。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。
error message (in)	エラー・メッセージ・テキスト	これは、dbmlsync ログに表示されるテキストと同じです。
error id (in)	整数	メッセージをユニークに識別する ID。このローを使用すると、エラー・メッセージ・テキストが変更されたときに、エラー・メッセージを識別できます。
error hook user state (in|out)	整数	この値はフックによって設定して、今後の呼び出しに対するステータス情報を、sp_hook_dbmlsync_all_error、sp_hook_dbmlsync_communication_error、sp_hook_dbmlsync_misc_error、sp_hook_dbmlsync_sql_error、または sp_hook_dbmlsync_end フックに渡すことができます。これらのフックの 1 つが最初に呼び出されるとき、ローの値は 0 です。フックがローの値を変更した場合は、次回のフックの呼び出しには新しい値が使用されます。 このフックを使用して、sp_hook_dbmlsync_end フックにステータス情報を渡す場合は、_end フックにより同期のリトライなどのアクションを実行することができます。

備考

同期を開始する前の起動中にエラーが発生した場合、#hook_dict 内の Mobile Link ユーザとスクリプト・バージョンのエントリは空の文字列に設定され、#hook_dict テーブルで設定される publication_*n* ローはありません。

このプロシージャは別個の接続で実行されるため、同期接続でロールバックが実行されても、このプロシージャで実行する操作が失われることはありません。dbmlsync が別個の接続を確立できないと、プロシージャは呼び出されません。

Windows Mobile デバイスのデフォルトでは、同期テーブルは排他モードでロックされます。このため、同期テーブルへのアクセスが必要な場合、このフックは正常に実行されません。また、このフックは、同期テーブルへのアクセスが必要で、ユーザが dbmlsync 拡張オプション LockTables を EXCLUSIVE に設定している場合にも実行できません。「LockTables (lt) 拡張オプション」 214 ページを参照してください。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「イベント・フック・プロシージャ内でのエラーと警告の処理」 253 ページ
- 「sp_hook_dbmlsync_communication_error」 262 ページ
- 「sp_hook_dbmlsync_all_error」 257 ページ
- 「sp_hook_dbmlsync_sql_error」 309 ページ

例

次のテーブルを使用して、リモート・データベース内のエラーのログを取るとします。

```
CREATE TABLE error_log
(
 pk INTEGER DEFAULT AUTOINCREMENT PRIMARY KEY,
 err_id INTEGER,
 err_msg VARCHAR(10240),
);
```

次の例では、sp_hook_dbmlsync_misc_error を設定して、全種類のエラー・メッセージのログを取ります。

```
CREATE PROCEDURE sp_hook_dbmlsync_misc_error()
BEGIN
 DECLARE msg VARCHAR(10240);
 DECLARE id INTEGER;

 // get the error message text
 SELECT value INTO msg
  FROM #hook_dict
  WHERE name ='error message';

 // get the error id
 SELECT value INTO id
  FROM #hook_dict
  WHERE name = 'error id';

 // log the error information
 INSERT INTO error_log(err_msg,err_id)
  VALUES (msg,id);
END;
```

可能性のあるエラーの ID 値を表示するには、dbmlsync をテスト実行します。たとえば、次の dbmlsync コマンド・ラインは、無効なパブリケーションを参照します。

```
dbmlsync -c eng=custdb;uid=DBA;pwd=sql -n test
```

ここで、error_log テーブルには次のローが含まれ、このエラーはエラー ID 9931 に関連付けられます。

```
1,9931,
'There is no synchronization subscription to publication "test"'
```

カスタム・エラー処理を行うには、sp_hook_dbmlsync_misc_error でエラー ID 9931 を確認します。

```
ALTER PROCEDURE sp_hook_dbmlsync_misc_error()
BEGIN
 DECLARE msg VARCHAR(10240);
 DECLARE id INTEGER;

 // get the error message text
 SELECT value INTO msg
  FROM #hook_dict
  WHERE name ='error message';

 // get the error id
 SELECT value INTO id
  FROM #hook_dict
  WHERE name = 'error id';

 // log the error information
 INSERT INTO error_log(err_msg,err_id)
  VALUES (msg,id);

 IF id = 9931 THEN
  // handle invalid publication
 END IF;

END;
```


sp_hook_dbmlsync_ml_connect_failed

このストアド・プロシージャを使用して、Mobile Link サーバに対する接続が失敗した場合に異なる通信タイプまたはアドレスを使用してリトライします。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。
connection address (in|out)	接続アドレス	フックが呼び出されるとき、これは失敗した直前の通信の試行で使用されたアドレスです。この値を新しい接続アドレスに設定して通信を試行できます。retry を true に設定すると、次の通信の試行でこの値が使用されます。プロトコル・オプションのリストについては、「Mobile Link クライアント・ネットワーク・プロトコル・オプションの一覧」 37 ページを参照してください。
connection type (in|out)	ネットワーク・プロトコル	フックが呼び出されるとき、これは失敗した直前の通信の試行で使用されたネットワーク・プロトコル (TCPIP など) です。この値を新しいネットワーク・プロトコルに設定して通信を試行できます。retry を true に設定すると、次の通信の試行でこの値が使用されます。ネットワーク・プロトコルのリストについては、「CommunicationType (ctp) 拡張オプション」 199 ページを参照してください。
user data (in|out)	ユーザ定義のデータ	次の通信の試行が失敗した場合に使用されるステータス情報。たとえば、発生したリトライの回数を保存すると便利です。デフォルトは、空の文字列です。

名前	値	説明
allow remote ahead (in|out)	**true** | **false**	dbmlsync を -ra オプションで起動した場合だけ true です。このローは、現在の同期のために -ra オプションの読み込みまたは変更を行う場合だけ使用できます。「-r オプション」 179 ページを参照してください。
allow remote behind (in|out)	**true** | **false**	dbmlsync を -rb オプションを指定して起動した場合だけ true です。このローは、現在の同期のために -rb オプションの読み込みまたは変更を行う場合だけ使用できます。「-r オプション」 179 ページを参照してください。
retry (in|out)	**true** | **false**	失敗した接続の試行をリトライする場合にこの値を true に設定します。デフォルト値は false です。

備考

この名前のプロシージャが存在する場合、Mobile Link サーバへの接続で dbmlsync が失敗したときにそのプロシージャが呼び出されます。

このフックが適用されるのは、データベースへの接続の試行ではなく、Mobile Link サーバへの接続の試行に対してだけです。

進行オフセット不一致が発生した場合、dbmlsync は Mobile Link サーバから切断してから再接続します。この種の再接続では、このフックが呼び出されず、再接続が失敗すると同期も失敗します。

このプロシージャのアクションは、実行直後にコミットされます。

例

この例は、sp_hook_dbmlsync_ml_connect_failed フックを使用して最大 5 回まで接続をリトライします。

```
CREATE PROCEDURE sp_hook_dbmlsync_ml_connect_failed ()
BEGIN
   DECLARE idx integer;

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

 IF idx <= 5 THEN
      UPDATE #hook_dict
      SET value = idx
      WHERE name = 'user data';

      UPDATE #hook_dict
```



```
        SET value = 'TRUE'
        WHERE name = 'retry';
  END IF;
END;
```

次に、接続情報が含まれるテーブルを使用する例を示します。接続の試行が失敗すると、フックはリストの次のサーバを試行します。

```
CREATE TABLE conn_list (
   label   INTEGER PRIMARY KEY,
   addr  VARCHAR( 128 ),
   type  VARCHAR( 64 )
);
INSERT INTO conn_list
 VALUES ( 1, 'host=server1;port=91', 'tcpip' );
INSERT INTO conn_list
 VALUES ( 2, 'host=server2;port=92', 'http' );
INSERT INTO conn_list
 VALUES ( 3, 'host=server3;port=93', 'tcpip' );
COMMIT;

CREATE PROCEDURE sp_hook_dbmlsync_ml_connect_failed ()
BEGIN
 DECLARE idx INTEGER;
 DECLARE cnt INTEGER;

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

 SELECT COUNT( label ) INTO cnt FROM conn_list;

 IF idx <= cnt THEN
   UPDATE #hook_dict
      SET value = ( SELECT addr FROM conn_list WHERE label = idx )
      WHERE name = 'connection address';
         UPDATE #hook_dict
    SET value = (SELECT type FROM conn_list WHERE label=idx)
    WHERE name = 'connection type';

   UPDATE #hook_dict
    SET value = idx
       WHERE name = 'user data';

      UPDATE #hook_dict
       SET value = 'TRUE'
       WHERE name = 'retry';
 END IF;
END;
```

sp_hook_dbmlsync_process_exit_code

このストアド・プロシージャを使用して、終了コードを管理します。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
fatal error (in)	**true** | **false**	dbmlsync を終了させる原因となるエラーのためにこのフックが呼び出されるときは true。
aborted synchronization (in)	**true** | **false**	sp_hook_dbmlsync_abort フックからのアボート要求のためにこのフックが呼び出されるときは true。
exit code (in)	数値	直近の同期試行からの終了コード。0 は同期が成功したことを示します。他の値は同期が失敗したことを示します。この値は、そのフックを使用して同期をアボートするとき、sp_hook_dbmlsync_abort によって設定できます。
last exit code (in)	数値	最後にこのフックが呼び出されたときに #hook_dict テーブルの **new exit code** ローに格納される値、またはこれがフックへの最初の呼び出しの場合は 0。
new exit code (in|out)	数値	そのプロセスに対して選択した終了コード。dbmlsync が終了するとき、dblmsync の **exit code** はそのフックへの最後の呼び出しによってこのローに格納される値です。この値は -32768 から 32767 になります。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

コマンド・ラインで -n オプションを複数回指定する場合、スケジューリングを使用する場合、および sp_hook_dbmlsync_end で restart パラメータを使用する場合、dbmlsync セッションは複数の同期を実行できます。これらの状況では、1 つ以上の同期が失敗すると、デフォルトの終了コードによってどれが失敗したのかが示されません。このフックを使用して、同期からの終了コード値に基づいた dbmlsync プロセスの終了コード値を定義できます。また、このフックを使用して終了コード値のログを取ることもできます。

同期を開始する前の起動中にエラーが発生した場合、#hook_dict 内の Mobile Link ユーザとスクリプト・バージョンのエントリは空の文字列に設定され、#hook_dict テーブルで設定される publication_*n* ローはありません。

例

dbmlsync を実行して 5 つの同期を行い、終了コードで失敗した同期の数を示すとします。たとえば、終了コード 0 は失敗がないことを示し、1 は 1 つの同期が失敗したことを示します。これを実現するには、sp_hook_dbmlsync_process_exit_code フックを次のように定義します。この場合、3 つの同期が失敗すると新しい終了コードは 3 になります。

```
CREATE PROCEDURE sp_hook_dbmlsync_process_exit_code()
BEGIN
  DECLARE  rc INTEGER;

  SELECT value INTO rc FROM #hook_dict WHERE name = 'exit code';
  IF rc <> 0 THEN
    SELECT value INTO rc FROM #hook_dict WHERE name = 'last exit code';
    UPDATE #hook_dict SET value = rc + 1 WHERE name = 'new exit code';
  END IF;
END;
```

参照

- 「同期イベント・フックの順序」 249 ページ
- 「sp_hook_dbmlsync_abort」 255 ページ

sp_hook_dbmlsync_schema_upgrade

このストアド・プロシージャを使用して、スキーマを修正する SQL スクリプトを実行します。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョンの名前	同期に使用されるスクリプト・バージョン。
drop hook (out)	**never** | **always** | **on success**	次のいずれかの値を取ります。 **never** - (デフォルト) データベースから sp_hook_dbmlsync_schema_upgrade フックを削除しません。 **always** - フックの実行を試みた後、データベースから sp_hook_dbmlsync_schema_upgrade フックを削除します。 **on success** - フックの実行に成功した場合、データベースから sp_hook_dbmlsync_schema_upgrade フックを削除します。dbmlsync の -eh オプションが使用されている場合、または dbmlsync の拡張オプション IgnoreHookErrors が True に設定されている場合、on success は always と同じです。

備考

このストアド・プロシージャは、配備されたリモート・データベースでスキーマの変更を行うためのものです。スキーマ更新のためにこのフックを使用すると、スキーマが更新される前にリモート・データベースのすべての変更が同期されます。そうされることによって、データベースの同期が確実に継続されます。このフックが使用中の場合、dbmlsync の拡張オプション LockTables を off に設定しないでください (LockTables はデフォルトでは on です)。

同期中に Mobile Link によってアップロードが正常に適用され、受信確認されると、このフックは sp_hook_dbmlsync_download_end フックの後、sp_hook_dbmlsync_end フックの前に呼び出されます。このフックは、ダウンロード専用の同期中、またはファイル・ベースのダウンロードが作成または適用されるときには呼び出されません。

このフックで実行されるアクションは、フックが完了した直後にコミットされます。

参照

- 「リモート・クライアントでのスキーマの変更」 83 ページ

例

次の例では、sp_hook_dbmlsync_schema_upgrade プロシージャを使用して、リモート・データベース上の Dealer テーブルにカラムを追加します。アップグレードが成功すると、sp_hook_dbmlsync_schema_upgrade フックは削除されます。

```
CREATE PROCEDURE sp_hook_dbmlsync_schema_upgrade()
BEGIN
 -- Upgrade the schema of the Dealer table. Add a column:
 ALTER TABLE Dealer
  ADD dealer_description VARCHAR(128);

 -- If the schema upgrade is successful, drop this hook:
 UPDATE #hook_dict
  SET value = 'on success'
  WHERE name = 'drop hook';
END;
```

sp_hook_dbmlsync_set_extended_options

同期に適用される拡張オプションを指定して、次の同期の動作をプログラムによってカスタマイズするには、このストアド・プロシージャを使用します。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
extended options (out)	*opt=val*;...	次の同期のために追加される拡張オプション。

備考

この名前のプロシージャが存在する場合、各同期の前に 1 回以上呼び出されます。

このフックで指定される拡張オプションは、パブリケーションと Mobile Link ユーザ・エントリによって識別される同期にのみ適用され、このフックが同じ同期を対象として次に呼び出されるまで適用されます。

このフックを使用して、スケジュール・オプションを指定することはできません。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「同期イベント・フックの順序」 249 ページ
- 「Mobile Link SQL Anywhere クライアントの拡張オプション」 193 ページ
- 「優先順位」 195 ページ

例

次の例では、sp_hook_dbmlsync_set_extended_options を使用して、SendColumnNames 拡張オプションを指定します。この拡張オプションは、pub1 が同期される場合だけ適用されます。

```
CREATE PROCEDURE sp_hook_dbmlsync_set_extended_options ()
BEGIN
 IF exists(SELECT * FROM #hook_dict
  WHERE name LIKE 'publication_%' AND value='pub1')
 THEN
   -- specify the SendColumnNames=on extended option
   UPDATE #hook_dict
         SET value = 'SendColumnNames=on'
    WHERE name = 'extended options';
 END IF;
END;
```


sp_hook_dbmlsync_set_ml_connect_info

このストアド・プロシージャを使用して、ネットワーク・プロトコルとネットワーク・プロトコル・オプションを設定します。

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。
connection type (in/out)	tcpip、tls、http、https	Mobile Link サーバへの接続に使用するネットワーク・プロトコル。
connection address (in/out)	プロトコル・オプション	Mobile Link サーバへの接続に使用する通信アドレス。「Mobile Link クライアント・ネットワーク・プロトコル・オプションの一覧」 37 ページを参照してください。

備考

このフックを使用すると、ネットワーク・プロトコルとネットワーク・プロトコル・オプションを設定できます。

プロトコルとオプションは sp_hook_dbmlsync_set_extended_options でも設定できます。sp_hook_dbmlsync_set_extended_options は、同期の開始時に呼び出されるフックです。sp_hook_dbmlsync_set_ml_connect_info は、dbmlsync が Mobile Link サーバへの接続を開始する直前に呼び出されます。

このフックは、フック内でオプションを設定したいが、同期プロセスで sp_hook_dbmlsync_set_extended_options よりも後で設定したい場合に便利です。たとえば、使用しているネットワークの信号の強さを取得できるかどうかに基づいてオプションを設定できます。

参照

- 「dbmlsync のフックの概要」 249 ページ
- 「同期イベント・フックの順序」 249 ページ
- 「CommunicationType (ctp) 拡張オプション」 199 ページ
- 「Mobile Link クライアント・ネットワーク・プロトコル・オプションの一覧」 37 ページ
- 「sp_hook_dbmlsync_set_extended_options」 304 ページ

例

```
CREATE PROCEDURE sp_hook_dbmlsync_set_ml_connect_info()
begin
   UPDATE #hook_dict
   SET VALUE = 'tcpip'
     WHERE name = 'connection type';

   UPDATE #hook_dict
   SET VALUE = 'host=localhost'
     WHERE name = 'connection address';
end
```


sp_hook_dbmlsync_set_upload_end_progress

このストアド・プロシージャは、スクリプト化されたアップロード・サブスクリプションが同期されるときの終了進行状況を定義するために使用されます。このプロシージャが呼び出されるのは、スクリプト化されたアップロード・パブリケーションの同期中だけです。

#hook_dict テーブルのロー

名前	値	説明
generating download exclusion list (in)	TRUE | FALSE	同期中にアップロードが送信されない場合 (ダウンロード専用の同期や、ファイルベースのダウンロードが適用される場合など) は TRUE。この場合でもアップロード・スクリプトは呼び出され、生成した操作は、アップロードする必要があるローを変更するダウンロード操作の識別に使用します。このような操作が見つかると、ダウンロードは適用されません。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
start progress as timestamp_*n*	進行状況 (タイムスタンプ)	同期中の各パブリケーションの開始進行状況がタイムスタンプで示されます。ここで、*n* は、パブリケーション名の識別に使用する整数と同じです。
start progress as bigint_*n*	進行状況 (bigint)	同期中の各パブリケーションの開始進行状況が bigint で示されます。ここで、*n* は、パブリケーション名の識別に使用する整数と同じです。
script version (n)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。

名前	値	説明
end progress is bigint (in|out)	TRUE | FALSE	このローが TRUE に設定されている場合は、終了進行状況の値は、文字列 ('12345' など) として表される符号なし bigint であると見なされます。 このローが FALSE に設定されている場合は、終了進行状況の値は、文字列 ('1900/01/01 12:00:00.000' など) として表されるタイムスタンプであると見なされます。 デフォルト値は FALSE です。
end progress (in|out)	タイムスタンプ	フックはこのローを変更して、アップロード・スクリプトに渡される "end progress as bigint" と "end progress as timestamp" の値を変更できます。これらの値は、生成中のアップロードにすべての操作が含まれているかどうかの境界となる時点を定義します。 このローの値は、"progress is bigint" ローの設定に応じて、符号なし bigint またはタイムスタンプのいずれかに設定できます。このローのデフォルト値は、現在のタイムスタンプです。

備考

スクリプト化されたアップロードの場合は、アップロード・プロシージャが呼び出されるたびに、開始進行状況と終了進行状況の値が渡されます。プロシージャは、これら 2 つの値で定義された期間に発生した適切な操作をすべて返す必要があります。開始進行状況値は、最後に成功した同期での終了進行状況値と同じです。ただし、今回初めて同期を行っている場合、開始進行状況値は "January 1, 1900, 00:00:00.000" になります。終了進行状況値は、デフォルトで、dbmlsync がアップロードを構築し始めた時間です。

このフックを使用すると、デフォルトの終了進行状況値を上書きできます。アップロードには短い時間を定義したり、タイムスタンプ以外の方法 (世代番号など) に基づいた、進行状況を追跡するスキームを実装することができます。

"end progress is bigint" が true に設定されている場合、終了進行状況は、1900-01-01 00:00:00 から 9999-12-31 23:59:59:9999 の間のミリ秒数 (255,611,203,259,999) 以下の整数でなければなりません。

参照

- 「スクリプト化されたアップロードのカスタム進行状況値」 397 ページ
- 「同期イベント・フックの順序」 249 ページ
- 「スクリプト化されたアップロード」 385 ページ

sp_hook_dbmlsync_sql_error

このストアド・プロシージャを使用して、同期中に発生したデータベース・エラーを処理します。たとえば、sp_hook_dbmlsync_sql_error フックを実装すると、特定の SQL エラーが発生した場合に、特定のアクションを実行することができます。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。
error message (in)	エラー・メッセージ・テキスト	これは、dbmlsync ログに表示されるテキストと同じです。
error id (in)	数値	メッセージをユニークに識別する ID。このローを使用すると、エラー・メッセージ・テキストが変更されたときに、エラー・メッセージを識別できます。
error hook user state (in|out)	整数	この値はフックによって設定して、今後の呼び出しに対するステータス情報を、sp_hook_dbmlsync_all_error、sp_hook_dbmlsync_communication_error、sp_hook_dbmlsync_misc_error、sp_hook_dbmlsync_sql_error、または sp_hook_dbmlsync_end フックに渡すことができます。これらのフックの 1 つが最初に呼び出されるとき、ローの値は 0 です。フックがローの値を変更した場合は、次回のフックの呼び出しには新しい値が使用されます。 このフックを使用して、sp_hook_dbmlsync_end フックにステータス情報を渡す場合は、_end フックにより同期のリトライなどのアクションを実行することができます。

名前	値	説明
SQL code (in)	SQL エラー・コード	操作が失敗した時にデータベースから返される SQL エラー・コード。これらの値は、SQL Anywhere 11 インストール環境の *SDK¥Include* サブディレクトリにある *sqlerr.h* で定義されます。
SQL state (in)	SQLSTATE 値	操作が失敗した時にデータベースから返される SQL ステータス。

備考

同期を開始する前の起動中にエラーが発生した場合、#hook_dict 内の Mobile Link ユーザとスクリプト・バージョンのエントリは空の文字列に設定され、#hook_dict テーブルで設定される publication_*n* ローはありません。

SQL エラーは、SQL Anywhere の SQLCODE または ANSI SQL 規格の SQLSTATE を使用して識別できます。SQLCODE または SQLSTATE 値のリストについては、「SQL Anywhere のエラー・メッセージ」 『エラー・メッセージ』を参照してください。

このプロシージャは別個の接続で実行されるため、同期接続でロールバックが実行されても、このプロシージャで実行する操作が失われることはありません。dbmlsync が別個の接続を確立できないと、プロシージャは呼び出されません。

Windows Mobile デバイスのデフォルトでは、同期テーブルは排他モードでロックされます。このため、同期テーブルへのアクセスが必要な場合、このフックは正常に実行されません。また、このフックは、同期テーブルへのアクセスが必要で、ユーザが dbmlsync 拡張オプション LockTables を EXCLUSIVE に設定している場合にも実行できません。「LockTables (lt) 拡張オプション」 214 ページを参照してください。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「イベント・フック・プロシージャ内でのエラーと警告の処理」 253 ページ
- 「sp_hook_dbmlsync_all_error」 257 ページ
- 「sp_hook_dbmlsync_communication_error」 262 ページ
- 「sp_hook_dbmlsync_misc_error」 294 ページ
- 「SQL Anywhere のエラー・メッセージ」 『エラー・メッセージ』

sp_hook_dbmlsync_upload_begin

このストアド・プロシージャを使用して、アップロード転送の直前にカスタム・アクションを追加します。

#hook_dict テーブルのロー

名前	値	説明
Publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
Script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

備考

この名前のプロシージャが存在する場合、dbmlsync がアップロードを送信する直前に呼び出されます。

このプロシージャのアクションは、実行直後にコミットされます。

参照

- 「同期イベント・フックの順序」 249 ページ

例

次のテーブルを使用して、リモート・データベース上の同期イベントのログを取るとします。

```
CREATE TABLE SyncLog
(
 "event_id"         INTEGER NOT NULL DEFAULT AUTOINCREMENT ,
  "event_name"       VARCHAR(128) NOT NULL ,
  "ml_user"          VARCHAR(128) NULL ,
  "event_time"        TIMESTAMP NULL,
  "table_name"        VARCHAR(128) NULL ,
  "upsert_count"      VARCHAR(128) NULL ,
  "delete_count"      VARCHAR(128) NULL ,
  "exit_code"        INTEGER NULL ,
  "status_retval"     VARCHAR(128) NULL ,
  "pubs"             VARCHAR(128) NULL ,
  "sync_descr "       VARCHAR(128) NULL ,
   PRIMARY KEY ("event_id"),
);
```

次に、アップロードの転送の直前に Mobile Link ユーザと現在のタイムスタンプのログを取る例を示します。

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_begin ()
BEGIN
```

```
 INSERT INTO SyncLog (event_name, ml_user,event_time)
  SELECT 'upload_begin', #hook_dict.value, CURRENT TIMESTAMP
  FROM #hook_dict
  WHERE name = 'MobiLink user';
END;
```


sp_hook_dbmlsync_upload_end

Mobile Link サーバによってアップロードが受信されたことを dbmlsync で確認した後に、このストアド・プロシージャを使用してカスタム・アクションを追加します。

#hook_dict テーブルのロー

名前	値	説明
failure cause (in)	下の「備考」の値の範囲を参照	アップロード障害の原因。詳細については、下の「説明」を参照してください。
upload status (in)	**retry** | **committed** | **failed** | **unknown**	dbmlsync がアップロードの受信確認を行おうとしたときに、Mobile Link サーバから返されるステータスを指定します。 **retry** - アップロードの開始位置であるログ・オフセットの値が、Mobile Link サーバと dbmlsync で異なっていました。Mobile Link サーバは、アップロードをコミットしませんでした。dbmlsync ユーティリティは、新しいログ・オフセットから始まる別のアップロードを送信しようとします。 **committed** - Mobile Link サーバがアップロードを受信し、コミットしました。 **failed** - Mobile Link サーバは、アップロードをコミットしませんでした。 **unknown** - dbmlsync が -tu オプションを使用して起動され、トランザクション・レベルのアップロードが発生しました。アップロードされる各トランザクションでは、sp_hook_dbmlsync_upload_begin フックと sp_hook_dbmlsync_upload_end フックが呼び出され、upload status の値は、最後のものを除き、常に **unknown** です。
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
script version (in)	スクリプト・バージョン名	同期に使用される Mobile Link スクリプト・バージョン。

名前	値	説明
authentication value (in)	値	この値は、サーバ上の authenticate_user スクリプト、authenticate_user_hashed スクリプト、または authenticate_parameters スクリプトによって生成されます。upload status が unknown であるとき、またはリモート・データベースと統合データベースに格納されているログ・オフセット間の競合が原因でアップロードが再送信された後に upload_end フックが呼び出されたとき、値は空の文字列になります。

備考

この名前のプロシージャが存在する場合、dbmlsync がアップロードを送信し、Mobile Link サーバから受信確認を受け取った直後に呼び出されます。

このプロシージャのアクションは、実行直後にコミットされます。

#hook_dict テーブルの failure cause ローに有効なパラメータ値の範囲は、次のとおりです。

- **UPLD_ERR_ABORTED_UPLOAD** リモートでエラーが発生したため、アップロードに失敗しました。この失敗の一般的な原因には、通信エラーとメモリ不足が挙げられます。
- **UPLD_ERR_COMMUNICATIONS_FAILURE** 通信エラーが発生しました。
- **UPLD_ERR_LOG_OFFSET_MISMATCH** リモート・データベースと統合データベースに格納されているログ・オフセット間に競合があるため、アップロードに失敗しました。
- **UPLD_ERR_GENERAL_FAILURE** アップロードに失敗しました。原因は不明です。
- **UPLD_ERR_INVALID_USERID_OR_PASSWORD** ユーザ ID またはパスワードが正しくありません。
- **UPLD_ERR_USERID_OR_PASSWORD_EXPIRED** ユーザ ID またはパスワードの有効期限が切れています。
- **UPLD_ERR_USERID_ALREADY_IN_USE** このユーザ ID はすでに使用されています。
- **UPLD_ERR_DOWNLOAD_NOT_AVAILABLE** 統合データベース側でアップロードがコミットされましたが、Mobile Link はダウンロードを生成できなかったため、エラーが発生しました。
- **UPLD_ERR_PROTOCOL_MISMATCH** dbmlsync が Mobile Link サーバから予期しないデータを受信しました。
- **UPLD_ERR_SQLCODE_n** この *n* は整数です。統合データベースに SQL エラーが発生しました。指定された整数は、発生したエラーを表す SQLCODE です。

参照

- 「同期イベント・フックの順序」 249 ページ

例

次のテーブルを使用して、リモート・データベース上の同期イベントのログを取るとします。

```
CREATE TABLE SyncLog(
   "event_id"          INTEGER NOT NULL DEFAULT AUTOINCREMENT ,
   "event_name"        VARCHAR(128) NOT NULL ,
   "ml_user"           VARCHAR(128) NULL ,
   "event_time"        TIMESTAMP NULL,
   "table_name"        VARCHAR(128) NULL ,
   "upsert_count"      VARCHAR(128) NULL ,
   "delete_count"      VARCHAR(128) NULL ,
   "exit_code"         INTEGER NULL ,
   "status_retval"     VARCHAR(128) NULL ,
   "pubs"              VARCHAR(128) NULL ,
   "sync_descr "       VARCHAR(128) NULL ,
    PRIMARY KEY ("event_id"),
);
```

次に、Mobile Link サーバがアップロードを受信したかを dbmlsync が検証した後で Mobile Link ユーザと現在のタイムスタンプのログを取る例を示します。

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_end ()
BEGIN

 DECLARE status_return_value VARCHAR(255);

 -- store status_return_value
  SELECT #hook_dict.value
  INTO status_return_value
  FROM #hook_dict
  WHERE #hook_dict.name = 'upload status';

   INSERT INTO SyncLog (event_name, ml_user,
     status_retval, event_time)
   SELECT 'upload_end', #hook_dict.value,
     status_return_value, CURRENT TIMESTAMP
  FROM #hook_dict
  WHERE name = 'MobiLink user';
END;
```

sp_hook_dbmlsync_validate_download_file

このフックを使用して、ダウンロード・ファイルがリモート・データベースに適用できるかどうかを決定するためのカスタム論理を実装します。このフックは、ファイルベースのダウンロードが適用される場合のみ呼び出されます。

#hook_dict テーブルのロー

名前	値	説明
publication_*n* (in)	パブリケーション名	同期されているパブリケーション (*n* は整数)。アップロードされるパブリケーションごとに 1 つの publication_*n* エントリがあります。publication_*n* と generation number_*n* の *n* は一致します。*n* の番号は 0 から始まります。
MobiLink user (in)	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ。
file last download time (in)		ダウンロード・ファイルの最後のダウンロード時刻 (ダウンロード・ファイルには、最後のダウンロード時刻とその前のダウンロード時刻の間に変更されたすべてのローが含まれます)。
file next last download time (in)		ダウンロード・ファイルの最後から 2 番目のダウンロード時刻 (ダウンロード・ファイルには、最後のダウンロード時刻とその前のダウンロード時刻の間に変更されたすべてのローが含まれます)。
file creation time (in)		ダウンロード・ファイルが作成された時間。
file generation number_*n* (in)	数値	ダウンロード・ファイルからの世代番号。各 publication_*n* エントリに対して、1 つのファイル世代番号 number_*n* があります。publication_*n* と generation number_*n* の *n* は一致します。*n* の番号は 0 から始まります。
user data (in)	文字列	ダウンロード・ファイルが作成されたときに、dbmlsync -be オプションで指定した文字列。

名前	値	説明
apply file (in|out)	**True**|**False**	True (デフォルト) の場合、ダウンロード・ファイルは dbmlsync の他の検証チェックを通過したときだけ適用されます。False の場合、ダウンロード・ファイルはリモート・データベースに適用されません。
check generation number (in|out)	**True**|**False**	True (デフォルト) の場合、dbmlsync は世代番号を検証します。ダウンロード・ファイル内の世代番号がリモート・データベース内の世代番号と一致しない場合、dbmlsync はダウンロード・ファイルを適用しません。False の場合、dbmlsync は世代番号をチェックしません。
setting generation number (in)	**true** | **false**	ダウンロード・ファイルが作成されたときに -bg オプションが使用された場合は true。-bg が使用された場合、リモート・データベースの世代番号はダウンロード・ファイルから更新され、通常の世代番号チェックは行われません。

備考

このストアド・プロシージャを使用して、ダウンロード・ファイルが適用できるか決定するためのカスタム・チェックを実装します。

ダウンロード・ファイルに含まれる世代番号またはタイムスタンプと、リモート・データベースに格納されている世代番号またはタイムスタンプを比較する場合、それらのデータは SYSSYNC と SYSPUBLICATION システム・ビューから問い合わせることができます。

-ba オプションが指定されていると、このフックが呼び出されます。ダウンロード・ファイルがリモート・データベースに適用される前に呼び出されます。

このフックのアクションは、フックが完了した直後にコミットされます。

参照

- 「-be オプション」 149 ページ
- 「-bg オプション」 150 ページ
- 「Mobile Link ファイルベースのダウンロード」 『Mobile Link - サーバ管理』

例

次の例では、ユーザ文字列「sales manager data」を含まないダウンロードファイルを適用しません。

```
CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file ()
BEGIN
  IF NOT exists(SELECT * FROM #hook_dict
   WHERE name = 'User data' AND value='sales manager data')
```

```
  THEN
    UPDATE #hook_dict
          SET value = 'false' WHERE name = 'Apply file';
  END IF;
END;
```


dbmlsync API

目次

dbmlsync API の概要

dbmlsync API は、C++ または .NET で記述された Mobile Link クライアント・アプリケーションが同期を起動し、要求した同期の進行状況に関するフィードバックを受け取れるようにするプログラミング・インタフェースです。この API は、同期をアプリケーションにシームレスに統合するためのものです。

この新しいプログラミング・インタフェースを使用することで、同期の結果に関してアクセスできる情報が大幅に増え、同期のキューイングも可能になるため、同期の管理が容易になります。

アーキテクチャ

dbmlsync API を使用するときは、クライアント・アプリケーションは DbmlsyncClient クラスをインスタンス化し、そのクラスのメソッドを呼び出します。このクラスは、TCP/IP を使用して別のプロセス (dbmlsync サーバ) と通信します。このプロセスが実際に、Mobile Link サーバとリモート・データベースに接続することによって同期を実行します。同期によって生成されたステータス情報は、DbmlsyncClient クラスの GetEvent メソッドを通じてクライアント・アプリケーションに送り返されます。

複数のクライアントで、同じ dbmlsync サーバを共有できます。ただし、各 dbmlsync サーバが同期できるのは 1 つのリモート・データベースのみであり、各リモート・データベースには同期する dbmlsync サーバを 1 つのみを設定できます。

dbmlsync サーバは、一度に 1 つの同期を実行します。同期の実行中に同期要求を受信した場合、その要求をキューイングし、後で履行します。

dbmlsync API のインタフェース

dbmlsync API には、C++ 用と .NET 用の 2 つのバージョンがあります。

C++ バージョンは、dbmlsynccli11.dll という DLL で実装されています。このバージョンを使用するクライアントは、C++ コードにヘッダ dbmlsynccli.hpp を含め、インポート・ライブラリ dbmlsynccli11.lib とリンクする必要があります。その上で、クライアントは、アプリケーションとともに dbmlsynccli11.dll ファイルを配布する必要があります。「C++ 用 dbmlsync API」 321 ページを参照してください。

API の .NET バージョンは、iAnywhere.MobiLink.Client.dll という DLL で実装されています。.NET アプリケーションは、この DLL への参照を含むことによってインタフェースを使用できます。「.NET 用 dbmlsync API」 335 ページを参照してください。

C++ 用 dbmlsync API

この項では、DbmlsyncClient クラスの C++ 実装のメソッドについて説明します。

この後に示す例は、dbmlsync API の C++ バージョンを使用して同期を行い、出力イベントを受け取る典型的なアプリケーションを示したものです。この例では、わかりやすいようにエラー処理を省略しています。各 API 呼び出しの戻り値を確認する習慣をつけることをおすすめします。

dbmlsync C++ の例

```
#include <stdio.h>
#include "dbmlsynccli.h"


int main( void ) {
    DbmlsyncClient *client;
    DBSC_SyncHdl    syncHdl;
    DBSC_Event     *ev1;

    client = DbmlsyncClient::InstantiateClient();
    if( client == NULL ) return( 1 );
    client->Init();

    // Setting the "server path" is usually required on Windows Mobile/CE.
    // In other environments the server path is usually not required unless
    // you SA install is not in your path or you have multiple versions of the
    // product installed
    client->SetProperty( "server path", "C:¥¥SQLAnywhere¥¥bin32" );

    client->StartServer( 3426,
            "-c eng=remote;dbn=rem1;uid=dba;pwd=sql -v+ -ot c:¥¥dbsync1.txt",
            5000, NULL );
    client->Connect( NULL, 3426, "dba", "sql");
    syncHdl = client->Sync( "my_sync_profile", "" );
    while( client->GetEvent( &ev1, 5000) == DBSC_GETEVENT_OK ) {
        if( ev1->hdl == syncHdl ) {
            //
            // Process events that interest you here
            //

            if( ev1->type == DBSC_EVENTTYPE_SYNC_DONE ) {
                client->FreeEventInfo( ev1 );
                break;
            }
            client->FreeEventInfo( ev1 );
        }
    }
    client->ShutdownServer( DBSC_SHUTDOWN_ON_EMPTY_QUEUE );
    client->WaitForServerShutdown( 10000 );
    client->Disconnect();
    client->Fini();
    delete client;

    return( 0 );
}
```

API の C++ バージョンの DbmlsyncClient クラスのパブリック・メソッドを次に示します。

InstantiateClient メソッド

InstantiateClient メソッドを呼び出して、DbmlsyncClient クラスをインスタンス化します。

構文

static DbmlsyncClient ***InstantiateClient( );**

備考

このメソッドによって返されるポインタを使用して、クラス内の残りのメソッドを呼び出すことができます。InstantiateClient によって返されたインスタンスを使い終わったら、ポインタで delete を呼び出すことによって、インスタンスを破棄できます。

戻り値

作成された新しいインスタンスへのポインタを返します。

エラーが発生した場合は NULL を返します。

Init メソッド

クラスのインスタンスを初期化します。

構文

bool **Init( );**

備考

InstantiateClient メソッドを使用して DbmlSyncClient クラスがインスタンス化された後に呼び出される最初のメソッドは、このメソッドにしてください。

戻り値

クラス・インスタンスが正常に初期化された場合は true を返します。

クラス・インスタンスが正常に初期化されなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。インスタンスが正常に初期化されるまで、他のメソッドを呼び出すことはできません。「GetErrorInfo メソッド」 329 ページを参照してください。

Fini メソッド

クラスのこのインスタンスによって使用されているすべてのリソースを解放します。

構文

bool **Fini**

備考

Fini メソッドを呼び出してから、クラスのインスタンスを削除してください。

インスタンスがサーバに接続されている場合は、Disconnect メソッドを呼び出してから、Fini メソッドを呼び出してください。

戻り値

メソッドが正常終了した場合は true を返します。

メソッドが正常終了しなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

StartServer メソッド

このメソッドはまず、指定のポートで受信している dbmlsync サーバがあるかどうかをチェックします。サーバがある場合、メソッドは starttype パラメータを DBSC_SS_ALREADY_RUNNING に設定し、その他の処理を行わずに戻ります。サーバが見つからない場合、cmdline 引数で指定されたオプションを使用して新しいサーバを起動し、そのサーバが要求を受け入れ始めるまで待ってから、戻ります。

注意

Windows Mobile デバイスでは通常、StartServer を正常に呼び出すには、先にサーバの path プロパティを設定する必要があります。ただし、次の場合はサーバの path プロパティを設定する必要はありません。

- アプリケーションが *dbmlsync.exe* と同じディレクトリにある。
- *dbmlsync.exe* が Windows ディレクトリにある。

構文

bool **StartServer(** unsigned port, const char *cmdline, unsigned timeout, DBSC_Starttype *starttype **);**

パラメータ

- **port** 既存の dbmlsync サーバがないかどうかをチェックする TCP ポート。サーバは、新しく起動される場合、このポートで受信するように設定されます。
- **cmdline** dbmlsync サーバを起動するための有効なコマンド・ライン。コマンド・ラインには、次のオプションのみを含めることができます。これらのオプションは、dbmlsync ユーティリティに対して持つ意味と同じ意味を持ちます。
  - -a、-c、-dl、-do、-ek、-ep、-k、-l、-o、-os、-ot、-p、-pc+、-pc-、-pd、-pp、-q、-qi、-qc、-sc、-sp、-uc、-ud、-ui、-um、-un、-ux、-v[cnoprsut]、-wc、-wh。「dbmlsync 構文」 139 ページを参照してください。

  -c オプションを指定してください。

- **timeout** dbmlsync サーバが起動された後、要求を受け入れる準備が完了するまでの最大待ち時間 (ミリ秒単位)。待ち時間を制限しない場合は、DBSC_INFINITY を使用します。
- **starttype** これは OUT パラメータです。starttype がエントリ時に NULL 以外の値であり、かつ StartServer が true を返した場合、終了時に starttype が指している変数が次のいずれかの値に設定されます。
  - **DBSC_SS_STARTED** 新しい dbmlsync サーバが起動されたことを示します。
  - **DBSC_SS_ALREADY_RUNNING** 既存の dbmlsync サーバが見つかったため、新しいサーバが起動されなかったことを示します。

戻り値

サーバがすでに実行されている場合、および正常に起動された場合は true を返します。

サーバが正常に起動されなかった場合、およびタイムアウトまでにサーバが要求の処理を開始しなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

Connect メソッド

すでにこのコンピュータで実行されている dbmlsync サーバとの接続を開きます。

構文

bool **Connect(** const char *host, unsigned port, const char *uid, const char *pwd **);**

備考

渡されたデータベース・ユーザ ID とパスワードを使用して、このクライアントがデータベースの同期に必要なパーミッションを持っているかどうかが検証されます。同期が実行されるときは、dbmlsync サーバの起動時に -c オプションで指定したユーザ ID が使用されます。

パラメータ

- **host** 予約。NULL を使用します。
- **port** サーバが受信している TCP ポート。StartServer メソッドを使用してサーバを起動したときに指定した port 値と同じ port 値を使用してください。
- **uid** 同期されるリモート・データベースに対する DBA 権限または REMOTE DBA 権限を持つ、有効なデータベース・ユーザ ID。
- **pwd** uid で指定されたユーザのデータベース・パスワード。

戻り値

サーバへの接続が確立された場合は true を返します。

サーバへの接続を確立できなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

Disconnect メソッド

Connect メソッドを使用して作成された dbmlsync サーバとの接続を切断します。

構文

bool **Disconnect( )**

備考

接続を使い終わったら、必ず Disconnect を呼び出してください。

戻り値

サーバへの接続が正常に切断された場合は true を返します。

サーバへの接続を切断できなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

Ping メソッド

dbmlsync サーバに ping 要求を送信して、接続が正常であるかどうか、およびサーバが活動状態であって要求に応答しているかどうかを確認します。

構文

bool **Ping(** unsigned timeout **)**

備考

このメソッドを呼び出すには、サーバに接続されている必要があります。

パラメータ

- **timeout**　サーバが ping 要求に応答するのを待つ最大のミリ秒数。待ち時間を制限しない場合は、DBSC_INFINITY を使用します。

戻り値

ping 要求に対する応答をサーバから受信した場合は true を返します。

ping 要求に対する応答を受信しなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

Sync メソッド

同期を実行するよう、dbmlsync サーバに要求します。

構文

DBSC_SyncHdl **Sync(** const char *profile_name, const char *extra_opts **)**

備考

サーバに接続されてから、このメソッドを呼び出してください。

profile_name と extra_opts のうち少なくとも 1 つを NULL 以外の値にしてください。

パラメータ

- **profile_name**　同期のオプションを含む同期プロファイルの名前で、リモート・データベース内に定義されているもの。profile_name が NULL の場合、プロファイルは使用されないため、extra_opts パラメータに、同期に使用するすべてのオプションが指定されている必要があります。
- **extra_opts**　同期プロファイルのオプション文字列を定義するときと同じルールに従った形式の文字列。profile_name が NULL 以外の場合、extra_opts で指定したオプションは、profile_name で指定した同期プロファイルにすでにあるオプションに追加されます。プロファイルに文字列のオプションがすでに存在する場合は、プロファイルにすでに格納済みの値が文字列の値に置き換わります。profile_name が NULL の場合、extra_opts に、同期に使用するすべてのオプションが指定されている必要があります。

戻り値

この同期要求をユニークに識別する DBSC_SyncHdl 値を返します。返されるハンドルは、クライアントがサーバから切断されるまでの間のみ有効です。

エラーのために同期要求を作成できなかった場合は NULL_SYNCHDL を返します。NULL_SYNCHDL が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

ShutdownServer メソッド

クライアントの接続先である dbmlsync サーバを停止します。

構文

bool **ShutdownServer(** DBSC_ShutdownType how **)**

備考

Shutdown メソッドはすぐに戻りますが、サーバが実際に停止するまでに遅延が生じることがあります。

ShutdownServer を呼び出した後に Disconnect を呼び出す必要があります。

WaitForServerShutdown メソッドを使用すると、サーバが実際に停止するまで待つことができます。「WaitForServerShutdown メソッド」 327 ページを参照してください。

パラメータ

- **how**　サーバを停止する緊急度を示します。次の値がサポートされます。

- **DBSC_SHUTDOWN_ON_EMPTY_QUEUE**　サーバが未処理の同期要求を完了してから停止する必要があることを示します。サーバは、停止要求を受信すると、それよりも後の同期要求を受け入れません。
- **DBSC_SHUTDOWN_CLEANLY**　サーバができるだけ早く正常に停止する必要があることを示します。未処理の同期要求は実行されません。実行中の同期は中断されることがあります。

戻り値

停止要求が正常にサーバに送信された場合は true を返します。

停止要求を送信できなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

WaitForServerShutdown メソッド

WaitForServerShutdown メソッドは、サーバが停止したときかタイムアウトになったときのどちらか早い方で戻ります。

構文

bool **WaitForServerShutdown(** unsigned timeout **)**

備考

WaitForServerShutdown を呼び出すには、先に ShutdownServer メソッドを呼び出す必要があります。「ShutdownServer メソッド」 326 ページを参照してください。

パラメータ

- **timeout**　サーバが停止するまでの最大待ち時間 (ミリ秒) を示します。待ち時間を制限しない場合は、DBSC_INFINITY を使用します。

戻り値

サーバが停止したためにメソッドが戻った場合は true を返します。

それ以外の場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

CancelSync メソッド

Sync メソッドを使用して以前に行われた同期要求を、クライアントがキャンセルできるようにします。

構文

bool **CancelSync(** DBSC_SyncHdl hdl **)**

備考

キャンセルできるのは、サービスが提供されるのを待っている同期要求のみです。サーバは、すでに開始した同期をキャンセルしません。

このメソッドを使用するには、サーバへの接続が確立されている必要があります。Sync メソッドを呼び出した後、クライアントがサーバから切断されている場合は、このメソッドを使用することはできません。

パラメータ

- **hdl**　同期が要求されたときに Sync メソッドによって返された同期ハンドル。

戻り値

同期要求が正常にキャンセルされた場合は true を返します。

同期要求がキャンセルされなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

GetEvent メソッド

dbmlsync サーバは、同期を実行しながら、同期の進行状況に関する情報を含む一連のイベントを生成します。これらのイベントは、サーバから DbmlsyncClient クラスに送信され、そこでキューイングされます。GetEvent メソッドを呼び出すと、キュー内で待機している次のイベントがある場合、そのイベントが返されます。

構文

DBSC_GetEventRet **GetEvent(** DBSC_Event **event, unsigned timeout **)**

備考

キュー内で待機しているイベントがない場合、このメソッドは、イベントが実行可能になるまで、または指定されたタイムアウトになるまで待ってから、戻ります。

同期用に生成されるイベントのタイプは、プロパティを使用して制御できます。「SetProperty メソッド」 331 ページを参照してください。

パラメータ

- **event**　戻り値が DBSC_GETEVENT_OK である場合、event パラメータに、取得されたイベントに関する情報を含む DBSC_Event 構造体へのポインタが入力されます。event の構造体を使い終わったら、FreeEventInfo メソッドを呼び出して、event の構造体に関連付けられていたメモリを解放してください。「DBSC_Event 構造体」 333 ページと「FreeEventInfo メソッド」 329 ページを参照してください。
- **timeout**　すぐに返すことができるイベントがない場合に待つ最大のミリ秒数を指定します。待ち時間を制限しない場合は、DBSC_INFINITY を使用します。

戻り値

次のいずれかを返します。

- **DBSC_GETEVENT_OK**　イベントが正常に取得されたことを示します。
- **DBSC_GETEVENT_TIMED_OUT**　返すことができるイベントがないまま、タイムアウトになったことを示します。
- **DBSC_GETEVENT_FAILED**　エラーが発生したためにイベントが返されなかったことを示します。DBSC_GETEVENT_FAILED が返されたときは、GetErrorInfo メソッドを呼び出して、メソッドが失敗した原因の詳細を調べることができます。「GetErrorInfo メソッド」 329 ページを参照してください。

FreeEventInfo メソッド

GetEvent メソッドによって返された DBSC_Event 構造体に関連付けられていたメモリを解放します。

構文

bool **FreeEventInfo(** DBSC_Event *event **)**

備考

GetEvent メソッドによって返された DBSC_Event 構造体それぞれに対して、FreeEventInfo を呼び出してください。

パラメータ

- **event**　解放する DBSC_Event 構造体へのポインタ。

戻り値

メモリが正常に解放された場合は true を返します。

メモリを解放できなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

GetErrorInfo メソッド

インタフェース内の別のメソッドが失敗した直後にこのメソッドを使用して、失敗に関する追加情報を取得します。

構文

const DBSC_ErrorInfo ***GetErrorInfo( )**

戻り値

失敗に関する情報を含む DBSC_ErrorInfo 構造体へのポインタを返します。DBSC_ErrorInfo 構造体は、次のように定義されます。

```
typedef struct {
    DBSC_ErrorType type;
    const char  *str1;
    const char  *str2;
    long int  val1;
    long int  val2;
    DBSC_SyncHdl hdl1;
} DBSC_ErrorInfo;
```

この構造体の内容は、クラス・メソッドが次回呼び出されたときに上書きされることがあります。

備考

type フィールドは、失敗の理由を示す値を格納します。現在、type は次の値を取ります。

- **DBSC_ERR_OK**　エラーは発生しませんでした。
- **DBSC_ERR_NOT_INITIALIZED**　クラスが、Init メソッドを呼び出すことによって初期化されていません。
- **DBSC_ERR_ALREADY_INITIALIZED**　すでに初期化されたクラスに対して、Init メソッドが呼び出されました。
- **DBSC_ERR_NOT_CONNECTED**　dbmlsync サーバへの接続がありません。
- **DBSC_ERR_CANT_RESOLVE_HOST**　ホスト情報を解決できません。
- **DBSC_ERR_CONNECT_FAILED**　接続が失敗しました。
- **DBSC_ERR_INITIALIZING_TCP_LAYER**　TCP レイヤの初期化中にエラーが発生しました。
- **DBSC_ERR_ALREADY_CONNECTED**　すでに接続が確立されているため、Connect メソッドが失敗しました。
- **DBSC_ERR_PROTOCOL_ERROR**　これは内部エラーです。
- **DBSC_ERR_CONNECTION_REJECTED**　dbmlsync サーバによって接続が拒否されました。

  str1 は、サーバによって返された文字列を指します。この文字列には、接続試行が拒否された理由の詳細が含まれることがあります。
- **DBSC_ERR_TIMED _OUT**　サーバからの応答を待っている間にタイムアウトになりました。
- **DBSC_ERR_STILL_CONNECTED**　クラスがまだサーバに接続されているため、そのクラスに対して Fini を実行できませんでした。
- **DBSC_ERR_SYNC_NOT_CANCELED**　サーバが同期要求をキャンセルできませんでした。同期がすでに進行中であることが原因と思われます。
- **DBSC_ERR_INVALID_VALUE**　SetProperty メソッドに無効なプロパティ値が渡されました。
- **DBSC_ERR_INVALID_PROP_NAME**　指定したプロパティ名は無効です。

- **DBCS_ERR_VALUE_TOO_LONG** プロパティ値が長すぎます。プロパティは、DBCS_MAX_PROPERTY_LEN で指定されているバイト数よりも短くしてください。
- **DBSC_ERR_SERVER_SIDE_ERROR** サーバでエラーがありました。

  str1 は、サーバによって返された文字列を指します。この文字列には、エラーの詳細が含まれることがあります。
- **DBSC_ERR_CREATE_PROCESS_FAILED** 新しい dbmlsync サーバを起動できません。
- **DBSC_ERR_READ_FAILED** dbmlsync サーバからのデータの読み込み中にエラーが発生しました。
- **DBSC_ERR_WRITE_FAILED** dbmlsync サーバへのデータの送信中にエラーが発生しました。
- **DBSC_ERR_NO_SERVER_RESPONSE** 要求されたアクションを完了するために必要な応答をサーバから受信できませんでした。
- **DBSC_ERR_UID_OR_PWD_TOO_LONG** 指定した UID または PWD が長すぎます。
- **DBSC_ERR_UID_OR_PWD_NOT_VALID** 指定した UID または PWD が無効です。
- **DBSC_ERR_INVALID_PARAMETER** 関数に渡されたパラメータのいずれかが無効です。
- **DBSC_ERR_WAIT_FAILED** サーバが停止するのを待っている間にエラーが発生しました。
- **DBSC_SHUTDOWN_NOT_CALLED** ShutdownServer メソッドを呼び出す前に WaitForServerShutdown メソッドが呼び出されました。
- **DBSC_ERR_NO_SYNC_ACK** 同期要求がサーバに送信されましたが、受信確認が受信されませんでした。サーバが要求を受信したかどうかを判別できません。

  hdl1 は、送信された同期要求のハンドルです。サーバが要求を受信した場合は、このハンドルを使用すると、「GetEvent メソッド」 328 ページ を使用して取得された同期用イベントを識別できます。

str1、str2、val1、val2、hdl1 には失敗に関する詳細が含まれ、その意味は type 値によって異なります。ほとんどの type 値では、これらのフィールドに有用な情報はありません。例外は、前述のリストに示しています。

SetProperty メソッド

クラス・インスタンスに対して、動作のさまざまな面を変更するプロパティを設定できるようにします。

構文

bool **SetProperty(** const char *name, const char *value **)**

備考

プロパティ値への変更は、変更後に行われた同期要求にのみ反映されます。

サーバの path プロパティを設定して、StartServer メソッドが呼び出されたときにクライアントが *dbmlsync.exe* を起動するディレクトリを指定できます。このプロパティが設定されていない場合、path 環境変数を使用して *dbmlsync.exe* が検索されます。「StartServer メソッド」 323 ページを参照してください。

次のプロパティは、GetEvent メソッドによって返されるイベントのタイプを制御します。不要なイベントを無効にすることによって、パフォーマンスを向上できることがあります。イベント・タイプは、対応するプロパティを 1 に設定すると有効になり、0 に設定すると無効になります。「GetEvent メソッド」 328 ページを参照してください。

次に、使用可能なプロパティのリストと、各プロパティによって制御されるイベント・タイプを示します。

プロパティ名	制御されるイベント・タイプ	デフォルト
enable errors	DBSC_EVENTTYPE_ERROR_MSG	1
enable warnings	DBSC_EVENTTYPE_WARNING_MSG	1
enable info msgs	DBSC_EVENTTYPE_INFO_MSG	1
enable progress	DBSC_EVENTTYPE_PROGRESS_INDEX	0
enable progress text	DBSC_EVENTTYPE_PROGRESS_TEXT	0
enable title	DBSC_EVENTTYPE_TITLE	0
enable sync start	DBSC_EVENTTYPE_SYNC_START	1
enable sync done	DBSC_EVENTTYPE_SYNC_DONE	1

パラメータ

- **name** 設定するプロパティの名前。これは、上で定義されているプロパティ名のいずれかにしてください。
- **value** プロパティに設定する値。指定する文字列は、DBCS_MAX_PROPERTY_LEN バイト未満にしてください。

戻り値

プロパティが正常に設定された場合は true を返します。

プロパティが正常に設定されなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

GetProperty メソッド

プロパティの現在の値を取得します。

構文

bool **GetProperty(** const char *name, char *value **)**

パラメータ

- **name**　値を取得するプロパティの名前。有効なプロパティ名のリストについては、「SetProperty メソッド」 331 ページを参照してください。
- **value**　プロパティの値を格納する、DBSC_MAX_PROPERTY_LEN バイト以上のバッファ。

戻り値

プロパティが正常に取得された場合は true を返します。

プロパティを取得できなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 329 ページを参照してください。

DBSC_Event 構造体

DBSC_Event 構造体は、要求されている同期に関する情報を格納します。この構造体は、次のように定義されます。

```
typedef struct {
    DBSC_SyncHdl hdl;
    DBSC_EventType type;
    const char   *str1;
    const char  *str2;
    long int  val1;
    long int  val2;
    void   *data;
} DBSC_Event;
```

hdl フィールドは、構造体が情報を格納している同期要求を識別します。この値は、Sync メソッドによって返されるハンドルと一致します。

type フィールドは、レポートされるイベントのタイプを識別します。

残りのフィールドは、追加データを格納します。追加データの性質は、type フィールドの値によって異なります。次に、type が取り得る値のリストと、それぞれに関連付けられている残りのフィールドの意味を示します。

- **DBSC_EVENTTYPE_ERROR_MSG**　同期によってエラーが生成されました。str1 はエラーのテキストを指します。
- **DBSC_EVENTTYPE_WARNING_MSG**　同期によって警告が生成されました。str1 は警告のテキストを指します。

- **DBSC_EVENTTYPE_INFO_MSG**　同期によって情報メッセージが生成されました。str1 はメッセージのテキストを指します。
- **DBSC_EVENTTYPE_PROGRESS_INDEX**　進行状況バーを更新するための情報を提供します。val1 は、新しい進行状況値を格納します。完了した割合 (パーセント) を計算するには、val1 を 1000 で割ります。
- **DBSC_EVENTTYPE_PROGRESS_TEXT**　進行状況バーに関連付けられているテキストが更新されました。新しい値は、str1 が指します。
- **DBSC_EVENTTYPE_TITLE**　同期ウィンドウ／コントロールのタイトルが変更されました。新しいタイトルは、str1 が指します。
- **DBSC_EVENTTYPE_SYNC_START**　同期が開始しました。このイベントに関連付けられている追加情報はありません。
- **DBSC_EVENTTYPE_SYNC_DONE**　同期が完了しました。val1 は、同期からの終了コードを格納します。0 という値は、成功を示します。0 以外の値は、同期が失敗したことを示します。

.NET 用 dbmlsync API

この項では、DbmlsyncClient クラスの .NET 実装のメソッドについて説明します。

この後に示す例は、dbmlsync API の .NET バージョンを使用して同期を行い、出力イベントを受け取る典型的なアプリケーションを示したものです。この例では、わかりやすいようにエラー処理を省略しています。各 API 呼び出しの戻り値を確認する習慣をつけることをおすすめします。

dbmlsync .NET の例

```
using System;
using System.Collections.Generic;
using System.Text;
using iAnywhere.MobiLink.Client;

namespace ConsoleApplication6
{
   class Program
   {
      static void Main(string[] args)
      {
         DbmlsyncClient cli1;
         DBSC_StartType st1;
         DBSC_Event ev1;
         UInt32 syncHdl;

         cli1 = DbmlsyncClient.InstantiateClient();
         cli1.Init();

         // Setting the "server path" is usually required on Windows
         // Mobile/CE. In other environments the server path is usually
         // not required unless you SA install is not in your path or
         // you have multiple versions of the product installed
         cli1.SetProperty("server path", "d:¥¥sybase¥¥asa1100r¥¥bin32");

         cli1.StartServer(3426,
          "-c eng=cons;dbn=rem1;uid=dba;pwd=sql -ve+ -ot c:¥¥dbsync1.txt",
          5000, out st1);
         cli1.Connect(null, 3426, "dba", "sql");
         syncHdl = cli1.Sync("sp1", "");
         while (cli1.GetEvent(out ev1, 5000)
                 == DBSC_GetEventRet.DBSC_GETEVENT_OK)
         {
            if (ev1.hdl == syncHdl)
            {
               Console.WriteLine("Event Type : {0}", ev1.type);
               if (ev1.type == DBSC_EventType.DBSC_EVENTTYPE_INFO_MSG)
               {
                  Console.WriteLine("Info : {0}", ev1.str1);
               }
               if (ev1.type == DBSC_EventType.DBSC_EVENTTYPE_SYNC_DONE)
               {
                  break;
               }
            }
         }
         cli1.ShutdownServer(DBSC_ShutdownType.DBSC_SHUTDOWN_ON_EMPTY_QUEUE);
         cli1.WaitForServerShutdown(10000);
         cli1.Disconnect();
         cli1.Fini();
         Console.ReadLine();
```

```
        }
    }
}
```

API の .NET バージョンの DbmlsyncClient クラスのパブリック・メソッドを次に示します。

InstantiateClient メソッド

InstantiateClient メソッドを呼び出して、DbmlsyncClient クラスをインスタンス化します。

構文

static DbmlsyncClient **InstantiateClient()**

備考

このメソッドによって返されるオブジェクトを使用して、クラス内の残りのメソッドを呼び出すことができます。

戻り値

作成された新しい DbmlsyncClient インスタンスを返します。エラーが発生した場合、NULL が返されます。

Init メソッド

クラスのインスタンスを初期化します。

構文

Boolean **Init()**

備考

InstantiateClient メソッドを使用してクラスがインスタンス化された後に呼び出される最初のメソッドは、このメソッドにしてください。

戻り値

クラス・インスタンスが正常に初期化された場合は true を返します。

クラス・インスタンスが正常に初期化されなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。インスタンスが正常に初期化されるまで、他のメソッドを呼び出すことはできません。「GetErrorInfo メソッド」 343 ページを参照してください。

StartServer メソッド

このメソッドはまず、指定のポートで受信している dbmlsync サーバがあるかどうかをチェックします。サーバがある場合、メソッドは starttype パラメータを

DBSC_SS_ALREADY_RUNNING に設定し、その他の処理を行わずに戻ります。サーバが見つからない場合、cmdline 引数で指定されたオプションを使用して新しいサーバを起動し、そのサーバが要求を受け入れ始めるまで待ってから、戻ります。

注意
Windows Mobile デバイスでは通常、StartServer を正常に呼び出すには、先にサーバの path プロパティを設定する必要があります。ただし、次の場合はサーバの path プロパティを設定する必要はありません。

- アプリケーションが *dbmlsync.exe* と同じディレクトリにある。
- *dbmlsync.exe* が Windows ディレクトリにある。

構文

Boolean **StartServer(**Int32 port, String cmdline, UInt32 timeout, out DBSC_Starttype starttype **)**

パラメータ

- **port**　既存の dbmlsync サーバがないかどうかをチェックする TCP ポート。サーバは、新しく起動される場合、このポートで受信するように設定されます。
- **cmdline**　dbmlsync サーバを起動するための有効なコマンド・ライン。コマンド・ラインには、次のオプションのみを含めることができます。これらのオプションは、dbmlsync ユーティリティに対して持つ意味と同じ意味を持ちます。
  - -a、-c、-dl、-do、-ek、-ep、-k、-l、-o、-os、-ot、-p、-pc+、-pc-、-pd、-pp、-q、-qi、-qc、-sc、-sp、-uc、-ud、-ui、-um、-un、-ux、-v[cnoprsut]、-wc、-wh。「dbmlsync 構文」 139 ページを参照してください。

  -c オプションを指定してください。
- **timeout**　dbmlsync サーバが起動された後、要求を受け入れる準備が完了するまでの最大待ち時間 (ミリ秒単位)。待ち時間を制限しない場合は、DBSC_INFINITY を使用します。DBSC_INFINITY 定数はネームスペースではなく DbmlSyncClient クラス内で定義されているので、定数にはプレフィクスが必要です (例：timeout = DbmlSyncClient.DBSC_INFINITY;)。
- **starttype**　これは OUT パラメータです。StartServer が true を返した場合、終了時に starttype が次のいずれかの値に設定されます。
  - **DBSC_SS_STARTED**　新しい dbmlsync サーバが起動されたことを示します。
  - **DBSC_SS_ALREADY_RUNNING**　既存の dbmlsync サーバが見つかったため、新しいサーバが起動されなかったことを示します。

戻り値

サーバがすでに実行されている場合、および正常に起動された場合は true を返します。

サーバが正常に起動されなかった場合、およびタイムアウトまでにサーバが要求の処理を開始しなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、

失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

Connect メソッド

すでにこのコンピュータで実行されている dbmlsync サーバとの接続を開きます。

構文

Boolean **Connect(**String host, Int32 port, String uid, String pwd**)**

備考

渡されたデータベース・ユーザ ID とパスワードを使用して、このクライアントがデータベースの同期に必要なパーミッションを持っているかどうかが検証されます。同期が実行されるときは、dbmlsync サーバの起動時に -c オプションで指定したユーザ ID が使用されます。

パラメータ

- **host**　予約。NULL を使用します。
- **port**　サーバが受信している TCP ポート。StartServer メソッドを使用してサーバを起動したときに指定した port 値と同じ port 値を使用してください。
- **uid**　同期されるリモート・データベースに対する DBA 権限または REMOTE DBA 権限を持つ、有効なデータベース・ユーザ ID。
- **pwd**　uid で指定されたユーザのデータベース・パスワード。

戻り値

サーバへの接続が確立された場合は true を返します。

サーバへの接続を確立できなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

Disconnect メソッド

Connect メソッドを使用して作成された dbmlsync サーバとの接続を切断します。

構文

Boolean **Disconnect()**

備考

接続を使い終わったら、必ず Disconnect を呼び出してください。

戻り値

サーバへの接続が正常に切断された場合は true を返します。

サーバへの接続を切断できなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

Ping メソッド

dbmlsync サーバに ping 要求を送信して、接続が正常であるかどうか、およびサーバが活動状態であって要求に応答しているかどうかを確認します。

構文

Boolean **Ping(**UInt32 timeout**)**

備考

このメソッドを呼び出すには、サーバに接続されている必要があります。

パラメータ

- **timeout**　サーバが ping 要求に応答するのを待つ最大のミリ秒数。待ち時間を制限しない場合は、DBSC_INFINITY を使用します。DBSC_INFINITY 定数はネームスペースではなく DbmlSyncClient クラス内で定義されているので、定数にはプレフィクスが必要です(例：timeout = DbmlSyncClient.DBSC_INFINITY;)。

戻り値

ping 要求に対する応答をサーバから受信した場合は true を返します。

ping 要求に対する応答を受信しなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

Sync メソッド

同期を実行するよう、dbmlsync サーバに要求します。

構文

UInt32 **Sync(** string profile_name, string extra_opts **)**

備考

サーバに接続されてから、このメソッドを呼び出してください。

syncName と opts のうち少なくとも 1 つを NULL 以外の値にしてください。

パラメータ

- **profile_name**　同期のオプションを含む同期プロファイルの名前で、リモート・データベース内に定義されているもの。syncName が NULL の場合、プロファイルは使用されないため、

extra_opts パラメータに、同期に使用するすべてのオプションが指定されている必要があります。

- **extra_opts**　同期プロファイルのオプション文字列を定義するときと同じルールに従った形式の文字列。この文字列で指定したオプションは、profile_name で指定した同期プロファイルにすでにあるオプションに追加されます。プロファイルに文字列のオプションがすでに存在する場合は、プロファイルにすでに格納済みの値が文字列の値に置き換わります。profile_name が NULL の場合、extra_opts に、同期に使用するすべてのオプションが指定されている必要があります。

戻り値

この同期要求をユニークに識別する整数値を返します。返される値は、クライアントがサーバから切断されるまでの間のみ有効です。

エラーのために同期要求を作成できなかった場合は NULL_SYNCHDL を返します。その場合は、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

ShutdownServer メソッド

クライアントの接続先である dbmlsync サーバを停止します。

構文

Boolean **ShutdownServer(**DBSC_ShutdownType how**)**

備考

Shutdown メソッドはすぐに戻りますが、サーバが実際に停止するまでに遅延が生じることがあります。

WaitForServerShutdown メソッドを使用すると、サーバが実際に停止するまで待つことができます。「WaitForServerShutdown メソッド」 341 ページを参照してください。

パラメータ

- **how**　サーバを停止する緊急度を示します。次の値がサポートされます。
  - **DBSC_SHUTDOWN_ON_EMPTY_QUEUE**　サーバが未処理の同期要求を完了してから停止する必要があることを示します。サーバは、停止要求を受信すると、それよりも後の同期要求を受け入れません。
  - **DBSC_SHUTDOWN_CLEANLY**　サーバができるだけ早く正常に停止する必要があることを示します。未処理の同期要求は実行されません。実行中の同期は中断されることがあります。

戻り値

停止要求が正常にサーバに送信された場合は true を返します。

停止要求を送信できなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

WaitForServerShutdown メソッド

WaitForServerShutdown メソッドは、サーバが停止したときかタイムアウトになったときのどちらか早い方で戻ります。

構文

Boolean **WaitForServerShutdown(** UInt32 timeout **)**

備考

WaitForServerShutdown を呼び出すには、先に ShutdownServer メソッドを呼び出す必要があります。「ShutdownServer メソッド」 340 ページを参照してください。

パラメータ

- **timeout** サーバが停止するまでの最大待ち時間 (ミリ秒) を示します。待ち時間を制限しない場合は、DBSC_INFINITY を使用します。DBSC_INFINITY 定数はネームスペースではなく DbmlSyncClient クラス内で定義されているので、定数にはプレフィクスが必要です(例：timeout = DbmlSyncClient.DBSC_INFINITY;)。

戻り値

サーバが停止したためにメソッドが戻った場合は true を返します。

それ以外の場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

CancelSync メソッド

Sync メソッドを使用して以前に行われた同期要求を、クライアントがキャンセルできるようにします。

構文

Boolean **CancelSync(**UInt32 hdl**)**

備考

キャンセルできるのは、サービスが提供されるのを待っている同期要求のみです。サーバは、すでに開始した同期をキャンセルしません。

このメソッドを使用するには、サーバへの接続が確立されている必要があります。Sync メソッドを呼び出した後、クライアントがサーバから切断されている場合は、このメソッドを使用することはできません。

パラメータ

- **hdl**　同期が要求されたときに Sync メソッドによって返された同期ハンドル。

戻り値

同期要求が正常にキャンセルされた場合は true を返します。

同期要求がキャンセルされなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

GetEvent メソッド

dbmlsync サーバは、同期を実行しながら、同期の進行状況に関する情報を含む一連のイベントを生成します。これらのイベントは、サーバから DbmlsyncClient クラスに送信され、そこでキューイングされます。GetEvent メソッドを呼び出すと、キュー内で待機している次のイベントがある場合、そのイベントが返されます。

構文

DBSC_GetEventRet **GetEvent(**out DBSC_Event ev, UInt32 timeout**)**

備考

キュー内で待機しているイベントがない場合、このメソッドは、イベントが実行可能になるまで、または指定されたタイムアウトになるまで待ってから、戻ります。

同期用に生成されるイベントのタイプは、プロパティを使用して制御できます。「SetProperty メソッド」 345 ページを参照してください。

パラメータ

- **ev**　戻り値が DBSC_GETEVENT_OK である場合、ev に、取得されたイベントに関する情報が入力されます。「DBSC_Event 構造体」 333 ページを参照してください。
- **timeout**　すぐに返すことができるイベントがない場合に待つ最大のミリ秒数を指定します。待ち時間を制限しない場合は、DBSC_INFINITY を使用します。DBSC_INFINITY 定数はネームスペースではなく DbmlSyncClient クラス内で定義されているので、定数にはプレフィクスが必要です(例：timeout = DbmlSyncClient.DBSC_INFINITY;)。

戻り値

次のいずれかを返します。

戻り値	説明
DBSC_GETEVENT_OK	イベントが正常に取得されたことを示します。
DBSC_GETEVENT_TIMED_OUT	返すことができるイベントがないまま、タイムアウトになったことを示します。

戻り値	説明
DBSC_GETEVENT_FAILED	エラーが発生したためにイベントが返されなかったことを示します。DBSC_GETEVENT_FAILED が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

GetErrorInfo メソッド

インタフェース内の別のメソッドが失敗した直後にこのメソッドを使用して、失敗に関する追加情報を取得します。

構文

DBSC_ErrorInfo **GetErrorInfo()**

戻り値

失敗に関する情報を含む DBSC_ErrorInfo 構造体へのポインタを返します。この構造体の内容は、クラス・メソッドが次回呼び出されたときに上書きされることがあります。DBSC_ErrorInfo 構造体は、次のように定義されます。

```
public struct DBSC_ErrorInfo
 {
   public DBSC_ErrorType type;
   public String str1;
   public String str2;
   public Int32 val1;
   public Int32 val2;
   public UInt32 hdl1;
 }
```

備考

type フィールドは、失敗の理由を示す値を格納します。現在、type は次の値を取ります。

type の値	説明
DBSC_ERR_OK	エラーは発生しませんでした。
DBSC_ERR_NOT_INITIALIZED	クラスが、init メソッドを呼び出すことによって初期化されていません。
DBSC_ERR_ALREADY_INITIALIZED	すでに初期化されたクラスに対して、init メソッドが呼び出されました。
DBSC_ERR_NOT_CONNECTED	dbmlsync サーバへの接続がありません。
DBSC_ERR_CANT_RESOLVE_HOST	ホスト情報を解決できません。

type の値	説明
DBSC_ERR_CONNECT_FAILED	接続が失敗しました。
DBSC_ERR_INITIALIZING_TCP_LAYER	TCP レイヤの初期化中にエラーが発生しました。
DBSC_ERR_ALREADY_CONNECTED	すでに接続が確立されているため、接続が失敗しました。
DBSC_ERR_PROTOCOL_ERROR	これは内部エラーです。
DBSC_ERR_CONNECTION_REJECTED	dbmlsync サーバによって接続が拒否されました。
DBSC_ERR_TIMED _OUT	サーバからの応答を待っている間にタイムアウトになりました。
DBSC_ERR_STILL_CONNECTED	クラスがまだサーバに接続されているため、そのクラスに対して Fini を実行できませんでした。
DBSC_ERR_SYNC_NOT_CANCELED	サーバが同期要求をキャンセルできませんでした。同期がすでに進行中であることが原因と思われます。
DBSC_ERR_INVALID_VALUE	SetProperty メソッドに無効なプロパティ値が渡されました。
DBSC_ERR_INVALID_PROP_NAME	指定したプロパティ名は無効です。
DBCS_ERR_VALUE_TOO_LONG	プロパティ値が長すぎます。プロパティは、DBCS_MAX_PROPERTY_LEN で指定されているバイト数よりも短くしてください。
DBSC_ERR_SERVER_SIDE_ERROR	サーバでエラーがありました。
DBSC_ERR_CREATE_PROCESS_FAILED	新しい dbmlsync サーバを起動できません。
DBSC_ERR_READ_FAILED	dbmlsync サーバからのデータの読み込み中にエラーが発生しました。
DBSC_ERR_WRITE_FAILED	dbmlsync サーバへのデータの送信中にエラーが発生しました。
DBSC_ERR_NO_SERVER_RESPONSE	要求されたアクションを完了するために必要な応答をサーバから受信できませんでした。
DBSC_ERR_UID_OR_PWD_TOO_LONG	指定した UID または PWD が長すぎます。

type の値	説明
DBSC_ERR_UID_OR_PWD_NOT_VALID	指定した UID または PWD が無効です。
DBSC_ERR_INVALID_PARAMETER	関数に渡されたパラメータのいずれかが無効です。
DBSC_ERR_WAIT_FAILED	サーバが停止するのを待っている間にエラーが発生しました。
DBSC_SHUTDOWN_NOT_CALLED	ShutdownServer メソッドを呼び出す前に WaitForServerShutdown メソッドが呼び出されました。
DBSC_ERR_NO_SYNC_ACK	同期要求がサーバに送信されましたが、受信確認が受信されませんでした。サーバが要求を受信したかどうかを判別できません。
DBSC_ERR_CONNECTION_REJECTED	str1 は、サーバによって返された文字列を指します。この文字列には、接続試行が拒否された理由の詳細が含まれることがあります。
DBSC_ERR_NO_SYNC_ACK	hdl1 は、送信された同期要求のハンドルです。サーバが要求を受信した場合は、このハンドルを使用すると、GetEvent を使用して取得された同期用イベントを識別できます。
DBSC_ERR_SERVER_SIDE_ERROR	str1 は、サーバによって返された文字列を指します。この文字列には、エラーの詳細が含まれることがあります。

str1、str2、val1、val2、hdl1 には失敗に関する詳細が含まれ、その意味は type 値によって異なります。ほとんどの type 値では、これらのフィールドに有用な情報はありません。ただし、次の例外があります。

- DBSC_ERR_CONNECTION_REJECTED
- DBSC_ERR_NO_SYNC_ACK
- DBSC_ERR_SERVER_SIDE_ERROR

SetProperty メソッド

クラス・インスタンスに対して、動作のさまざまな面を変更するプロパティを設定できるようにします。

構文

Boolean **SetProperty(**String name, String Value**)**

備考

プロパティ値への変更は、変更後に行われた同期要求にのみ反映されます。

サーバの path プロパティを設定して、StartServer メソッドが呼び出されたときにクライアントが *dbmlsync.exe* を起動するディレクトリを指定できます。このプロパティが設定されていない場合、path 環境変数を使用して *dbmlsync.exe* が検索されます。「StartServer メソッド」 336 ページを参照してください。

次のプロパティは、「GetEvent メソッド」 342 ページによって返されるイベントのタイプを制御します。不要なイベントを無効にすることによって、パフォーマンスを向上できることがあります。イベント・タイプは、対応するプロパティを 1 に設定すると有効になり、0 に設定すると無効になります。次に、使用可能なプロパティのリストと、各プロパティによって制御されるイベント・タイプを示します。

プロパティ名	制御されるイベント・タイプ	デフォルト
enable errors	DBSC_EVENTTYPE_ERROR_MSG	1
enable warnings	DBSC_EVENTTYPE_WARNING_MSG	1
enable info msgs	DBSC_EVENTTYPE_INFO_MSG	1
enable progress	DBSC_EVENTTYPE_PROGRESS_INDEX	0
enable progress text	DBSC_EVENTTYPE_PROGRESS_TEXT	0
enable title	DBSC_EVENTTYPE_TITLE	0
enable sync start	DBSC_EVENTTYPE_SYNC_START	1
enable sync done	DBSC_EVENTTYPE_SYNC_DONE	1

パラメータ

- **name**　設定するプロパティの名前。これは、上で定義されているプロパティ名のいずれかにしてください。
- **value**　プロパティに設定する値。

戻り値

プロパティが正常に設定された場合は true を返します。

プロパティが正常に設定されなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

GetProperty メソッド

プロパティの現在の値を取得します。

構文

Boolean **GetProperty(**String name, out String Value**)**

パラメータ

- **name**　値を取得するプロパティの名前。有効なプロパティ名のリストについては、「SetProperty メソッド」 345 ページを参照してください。
- **value**　終了時に、プロパティの値がこの変数に格納されます。

戻り値

プロパティが正常に取得された場合は true を返します。

プロパティを取得できなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

Fini メソッド

クラスのこのインスタンスによって使用されているすべてのリソースを解放します。

構文

Boolean **Fini()**

備考

Fini メソッドを呼び出してから、クラスのインスタンスを削除してください。

インスタンスがサーバに接続されている場合は、Disconnect メソッドを呼び出してから、Fini メソッドを呼び出してください。

戻り値

メソッドが正常終了した場合は true を返します。

メソッドが正常終了しなかった場合は false を返します。false が返されたときは、GetErrorInfo メソッドを呼び出して、失敗に関する詳細な情報を取得できます。「GetErrorInfo メソッド」 343 ページを参照してください。

DBSC_Event 構造体

DBSC_Event 構造体は、要求されている同期に関する情報を格納します。この構造体は、次のように定義されます。

```
public struct DBSC_Event
 {
   public UInt32 hdl;
   public DBSC_EventType type;
   public String str1;
   public String str2;
   public Int32 val1;
   public Int32 val2;
 }
```

hdl フィールドは、構造体が情報を格納している同期要求を識別します。この値は、Sync メソッドによって返される値と一致します。

type フィールドは、レポートされるイベントのタイプを識別します。

残りのフィールドは、追加データを格納します。追加データの性質は、type フィールドの値によって異なります。次に、type が取り得る値のリストと、それぞれに関連付けられている残りのフィールドの意味を示します。

値	説明
DBSC_EVENTTYPE_ERROR_MSG	同期によってエラーが生成されました。str1 はエラーのテキストを格納します。
DBSC_EVENTTYPE_WARNING_MSG	同期によって警告が生成されました。str1 は警告のテキストを格納します。
DBSC_EVENTTYPE_INFO_MSG	同期によって情報メッセージが生成されました。str1 はメッセージのテキストを格納します。
DBSC_EVENTTYPE_PROGRESS_INDEX	進行状況バーを更新するための情報を提供します。val1 は、新しい進行状況値を格納します。完了した割合 (パーセント) を計算するには、val1 を 1000 で割ります。
DBSC_EVENTTYPE_PROGRESS_TEXT	進行状況バーに関連付けられているテキストが更新されました。str1 は、新しい値を格納します。
DBSC_EVENTTYPE_TITLE	同期ウィンドウ／コントロールのタイトルが変更されました。str1 は新しいタイトルを格納します。
DBSC_EVENTTYPE_SYNC_START	同期が開始しました。このイベントに関連付けられている追加情報はありません。
DBSC_EVENTTYPE_SYNC_DONE	同期が完了しました。val1 は、同期からの終了コードを格納します。0 という値は、成功を示します。0 以外の値は、同期が失敗したことを示します。

dbmlsync 統合コンポーネント (旧式)

目次

注意
dbmlsync 統合コンポーネントは推奨されなくなりました。代わりに、dbmlsync プログラミング・インタフェースを使用してください。「dbmlsync API」 319 ページを参照してください。

dbmlsync 統合コンポーネントの概要

注意
dbmlsync 統合コンポーネントは推奨されなくなりました。代わりに、dbmlsync プログラミング・インタフェースを使用してください。「dbmlsync API」 319 ページを参照してください。

dbmlsync 統合コンポーネントは、アプリケーションに同期を追加するのに使用できる ActiveX です。このコンポーネントでは、SQL Anywhere クライアントの動作を調整するための一連のプロパティ、イベント、メソッドを提供します。

dbmlsync 統合コンポーネントは、2 つの形式で使用できます。いずれも、同じプロパティ、イベント、メソッドを公開しています。

- ビジュアル・コンポーネント。標準の dbmlsync ユーザ・インタフェースをアプリケーションに簡単に統合できます。
- 非ビジュアル・コンポーネント。ユーザ・インタフェースを使用せずに、または独自に作成したカスタム・ユーザ・インタフェースを使用して、コンポーネントの機能にアクセスできます。

dbmlsync 統合コンポーネントを使用すると、アプリケーションで同期を開始し、同期の進行状況についての情報を受け取ってから、同期イベントに基づいた特別な処理を実装できます。

dbmlsync の DBTools インタフェース

dbmlsync 統合コンポーネントを使用する代わりに、dbmlsync 用の DBTools インタフェースを使用することもできます。

「データベース・ツール・インタフェース」 『SQL Anywhere サーバ - プログラミング』を参照してください。

サポートされるプラットフォーム

dbmlsync 統合コンポーネントは、ActiveX をサポートする Windows Mobile バージョンを含め、Mobile Link でサポートされているすべての Windows オペレーティング・システムで使用できます。

サポートされている開発環境には、Microsoft Visual Basic 6.0、eMbedded Visual Basic、Visual Studio があります。

サポートされるプラットフォームのリストについては、http://www.ianywhere.jp/developers/technotes/os_components_1101.html を参照してください。

dbmlsync 統合コンポーネントの設定

dbmlsync 統合コンポーネントは、さまざまなプログラミング環境で使用できる ActiveX です。このコンポーネントの設定方法については、ご使用のプログラミング環境のマニュアルを参照してください。

dbmlsync 統合コンポーネントのメソッド

次に、DbmlsyncCOM.Dbmlsync クラスによって実装されるメソッドを示します。

Run メソッド

dbmlsync コマンド・ライン・オプションを使用して、1 つまたは複数の同期を開始します。

構文

Run(ByVal *cmdLine* As String **)**
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

cmdLine dbmlsync オプションを指定する文字列です。

備考

オプションのリストは、「dbmlsync 構文」 139 ページを参照してください。

Run メソッドはすぐに返され、同期が完了するのを待機しません。DoneExecution イベントを使用して、同期が完了するタイミングを指定することができます。

cmdLine パラメータには、dbmlsync コマンド・ライン・ユーティリティとの同期を実行する場合に使用するのと同じオプションを含める必要があります。たとえば、次のコマンド・ラインと Run メソッドの呼び出しは同じです。

```
dbmlsync -c uid=DBA;pwd=sql
```

```
dbmlsync1.Run "-c uid=DBA;pwd=sql"
```

例

次の例は、remote1 というリモート・データベースの同期を開始します。

```
dbmlsync1.Run "-c eng=remote1;uid=DBA;pwd=sql"
```

Stop メソッド

アクティブな同期に終了するように要求します。

構文

Stop( )
Member of **DbmlsyncCOM.Dbmlsync**

備考

Stop メソッドは、アクティブな同期を終了する要求を発行します。このメソッドはすぐに返されます。

ビジュアルな dbmlsync 統合コンポーネントに組み込まれた停止ボタンは、このメソッドを自動的に呼び出します。

例

次の例は、dbmlsync 統合コンポーネントのインスタンスである dbmlsync1 によって実行されている同期を停止します。

```
dbmlsync1.Stop
```

dbmlsync 統合コンポーネントのプロパティ

dbmlsync 統合コンポーネントのプロパティを使用すると、コンポーネントの動作をカスタマイズしたり、実行中の同期の状態を調べたりできます。

Path プロパティ

dbmlsync.exe のロケーションを指定します。

構文

Public Property **Path( )** As String
Member of **DbmlsyncCOM.Dbmlsync**

備考

Windows の PATH 環境変数で指定されているディレクトリに *dbmlsync.exe* が置かれている場合は、このプロパティを設定する必要はありません。

例

次の例は、dbmlsync 統合コンポーネントのインスタンスのパスを設定します。

```
dbmlsync1.Path = "c:¥program files¥SQL Anywhere 11¥bin32"
```

UploadEventsEnabled プロパティ

UploadRow イベントを有効にします。

構文

Public Property **UploadEventsEnabled()** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

備考

UploadRow イベントを処理する場合は、このプロパティを True に設定してください。デフォルトは False で、UploadRow イベントは無効になっています。プロパティを True に設定すると、パフォーマンスが低下します。

「UploadRow イベント」 371 ページを参照してください。

例

次の例は、UploadEventsEnabled を True に設定します。

```
dbmlsync1.UploadEventsEnabled = True
```


DownloadEventsEnabled プロパティ

DownloadRow イベントを有効にします。

構文

Public Property **DownloadEventsEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

備考

DownloadRow イベントを処理する場合は、このプロパティを True に設定してください。デフォルトは False で、DownloadRow イベントは無効になっています。プロパティを True に設定すると、パフォーマンスが低下します。

「DownloadRow イベント」 363 ページを参照してください。

例

次の例は、DownloadEventsEnabled を True に設定します。

```
dbmlsync1.DownloadEventsEnabled = True
```

ErrorMessageEnabled プロパティ

MsgError 型のメッセージに対して Message イベントが呼び出されないようにします。

構文

Public Property **ErrorMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

備考

エラー情報を Message イベントで処理しない場合は、パフォーマンスを向上させるため、このプロパティを False に設定してください。デフォルトは True で、Message イベントをトリガする MsgError 型のメッセージが有効になっています。

「Message イベント」 366 ページを参照してください。

例

次の例は、ErrorMessageEnabled を False に設定します。

```
dbmlsync1.ErrorMessageEnabled = False
```

WarningMessageEnabled プロパティ

MsgWarning 型のメッセージに対して Message イベントが呼び出されないようにします。

構文

Public Property **WarningMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

備考

警告情報を Message イベントで処理しない場合は、パフォーマンスを向上させるため、このプロパティを False に設定してください。デフォルトは True で、Message イベントをトリガする MsgWarning 型のメッセージが有効になっています。

「Message イベント」 366 ページを参照してください。

例

次の例は、WarningMessageEnabled を False に設定します。

```
dbmlsync1.WarningMessageEnabled = False
```

InfoMessageEnabled プロパティ

MsgInfo 型のメッセージに対して Message イベントが呼び出されないようにします。

構文

Public Property **InfoMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

備考

通常の進行状況情報を Message イベントで処理しない場合は、パフォーマンスを向上させるため、このプロパティを False に設定してください。デフォルトは True で、Message イベントをトリガする MsgInfo 型のメッセージが有効になっています。

「Message イベント」 366 ページを参照してください。

例

次の例は、InfoMessageEnabled を False に設定します。

```
dbmlsync1.InfoMessageEnabled = False
```

DetailedInfoMessageEnabled プロパティ

MsgDetailedInfo 型のメッセージに対して Message イベントが呼び出されないようにします。

構文

Public Property **DetailedInfoMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

備考

詳細な進行状況情報を Message イベントで処理しない場合は、パフォーマンスを向上させるため、このプロパティを False に設定してください。デフォルトは True で、Message イベントをトリガする MsgDetailedInfo 型のメッセージが有効になっています。

「Message イベント」 366 ページを参照してください。

例

次の例は、DetailedInfoMessageEnabled を False に設定します。

```
dbmlsync1.DetailedInfoMessageEnabled = False
```

UseVB6Types プロパティ

Visual Basic 6 を使用している場合は、このプロパティを True に設定して、UploadRow イベントと DownloadRow イベントから返されるロー・データの処理を簡素化します。

構文

Public Property **DetailedInfoMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

備考

Visual Basic 6 では、符号なしの 32 ビット値とすべての 64 ビット値がサポートされていません。これらの型のデータは、IRowTransferData オブジェクトの ColumnValue プロパティから返されることがあります。UseVB6Types を True に設定すると、これらの型のデータは Visual Basic 6 でサポートされる別の型に変換され、処理が簡素化されます。Uint32 の値は double 型に変換され、64 ビット値は文字列に変換されます。

参照

- 「IRowTransferData インタフェース」 373 ページ
- 「UploadRow イベント」 371 ページ
- 「DownloadRow イベント」 363 ページ

例

次の例は、Visual Basic 6.0 で使用される dbmlsync 統合コンポーネントのインスタンスのデータ型の強制変換を有効にします。

```
dbmlsync1.UseVB6Types = True
```

ExitCode プロパティ

最新の Run メソッドの呼び出しによって開始された同期からの終了コードを返します。

構文

Public Property **ExitCode( )** As Integer
Member of **DbmlsyncCOM.Dbmlsync**

備考

ExitCode プロパティは、最後に呼び出された Run メソッドによって開始された同期の終了コードを返します。0 は同期が成功したことを示します。他の値は同期が失敗したことを示します。

> **注意**
> DoneExecution イベントがトリガされる前にこのプロパティの値を取得すると、終了コードが無効な値になることがあります。

例

次の例は、DoneExecution イベントがトリガされた時点での最新の同期からの終了コードを表示します。

```
Private Sub dbmlsync1_DoneExecution() Handles dbmlsync1.DoneExecution
    MsgBox(dbmlsync1.ExitCode)
End Sub
```

EventChannelSize プロパティ

メソッドの呼び出しの処理に使用される内部バッファのサイズを指定します。

構文

Public Property **EventChannelSize( )** As Integer
Member of **DbmlsyncCOM.Dbmlsync**

備考

ほとんどの場合、このプロパティを変更する必要はありません。

DispatchChannelSize プロパティ

イベント情報の処理に使用される内部バッファのサイズを指定します。

構文

Public Property **DispatchChannelSize( )** As Integer
Member of **DbmlsyncCOM.Dbmlsync**

備考

ほとんどの場合、このプロパティを変更する必要はありません。

dbmlsync 統合コンポーネントのイベント

イベントは、クライアント・アプリケーションが同期の進行状況の情報を受け取り、それに基づいてアクションを行うためのメカニズムを提供します。

BeginDownload イベント

BeginDownload イベントは、同期のダウンロード処理開始時にトリガされます。

構文

Public Event **BeginDownload( )**
Member of **DbmlsyncCOM.Dbmlsync**

備考

このイベントを使用して、同期のダウンロード処理開始時にカスタム・アクションを追加します。

例

次に示す Visual Basic .NET の例は、BeginDownload イベントがトリガされたときにメッセージを出力します。

```
Private Sub dbmlsync1_BeginDownload()
Handles dbmlsync1.BeginDownload

    MsgBox("Beginning Download")

End Sub
```

BeginLogScan イベント

BeginLogScan イベントは、アップロードをアセンブルするために dbmlsync がトランザクション・ログをスキャンする直前にトリガされます。このイベントは、スクリプト化されたアップロードでは起動されません。

構文

Public Event **BeginLogScan(** ByVal *rescanLog* As Boolean **)**
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

rescanLog　この同期でトランザクション・ログが初めてスキャンされる場合、この値は False、それ以外の場合は True。Mobile Link サーバと dbmlsync でスキャン開始位置の情報が異なっている場合、ログは 2 回スキャンされます。

備考

このイベントを使用して、アップロード用にトランザクション・ログがスキャンされる直前にカスタム・アクションを追加します。

例

次に示す Visual Basic .NET の例は、BeginLogScan イベントがトリガされたときにメッセージを出力します。

```
Private Sub dbmlsync1_BeginLogScan(
 ByVal rescanLog As Boolean
)
Handles dbmlsync1.BeginLogScan

    MsgBox("Begin Log Scan")

End Sub
```

BeginSynchronization イベント

BeginSynchronization イベントは、各同期の開始時にトリガされます。

構文

Public Event **BeginSynchronization(_**
  ByVal *userName* As String, _
  ByVal *pubNames* As String _
)
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

userName　同期対象となる Mobile Link ユーザ。

pubNames　同期されるパブリケーション。パブリケーションが複数の場合は、カンマで区切ったリストで指定します。

備考

このイベントを使用して、同期の開始時にカスタム・アクションを追加します。

例

次に示す Visual Basic .NET の例は、BeginSynchronization イベントがトリガされたときにメッセージを出力します。メッセージには、ユーザ名とパブリケーション名が出力されます。

```
Private Sub dbmlsync1_BeginSynchronization(
 ByVal userName As String,
 ByVal pubNames As String
)
Handles dbmlsync1.BeginSynchronization

    MsgBox("Beginning synchronization for: " + userName _
   + " publication: " + pubNames)

End Sub
```

BeginUpload イベント

BeginUpload イベントは、アップロード転送の直前にトリガされます。

構文

Public Event **BeginUpload( )**
Member of **DbmlsyncCOM.Dbmlsync**

備考

このイベントを使用して、Mobile Link サーバへのアップロード転送の直前にカスタム・アクションを追加します。

例

次に示す Visual Basic .NET の例は、BeginUpload イベントがトリガされたときにメッセージを出力します。

```
Private Sub dbmlsync1_BeginUpload()
Handles dbmlsync1.BeginUpload

    MsgBox("Begin Upload")

End Sub
```

ConnectMobilink イベント

ConnectMobilink イベントは、コンポーネントが Mobile Link サーバに接続する直前にトリガされます。

構文

Public Event **ConnectMobilink( )**
Member of **DbmlsyncCOM.Dbmlsync**

備考

このイベントを使用して、リモート・データベースが Mobile Link サーバに接続する直前にカスタム・アクションを追加します。この段階では、dbmlsync はアップロードの生成を完了しています。

ConnectMobiLink イベントは、BeginSynchronization イベントの後で発生します。

例

次に示す Visual Basic .NET の例は、ConnectMobilink イベントがトリガされたときにメッセージを出力します。

```
Private Sub dbmlsync1_ConnectMobilink()
Handles dbmlsync1.ConnectMobilink

    MsgBox("Connecting to the MobiLink server")

End Sub
```

DisconnectMobilink イベント

DisconnectMobilink イベントは、コンポーネントが Mobile Link サーバから切断した直後にトリガされます。

構文

Public Event **DisconnectMobilink( )**
Member of **DbmlsyncCOM.Dbmlsync**

備考

このイベントを使用して、リモート・データベースが Mobile Link サーバから切断した直後にカスタム・アクションを追加します。

例

次に示す Visual Basic .NET の例は、DisconnectMobilink イベントがトリガされたときにメッセージを出力します。

```
Private Sub dbmlsync1_DisconnectMobilink()
Handles dbmlsync1.DisconnectMobilink

    MsgBox("Disconnected from the MobiLink server")

End Sub
```

DoneExecution イベント

DoneExecution イベントは、Run メソッドの呼び出しによって開始された同期がすべて完了したときにトリガされます。

構文

Public Event **DoneExecution( )**
Member of **DbmlsyncCOM.Dbmlsync**

備考

このイベントを使用して、Run メソッドの呼び出しによって開始された同期がすべて完了したときにカスタム・アクションを追加します。

例

ExitCode プロパティを使用する次の Visual Basic .NET 例は、最後に呼び出された Run メソッドによって開始された同期からの終了コードを出力します。

```
Private Sub dbmlsync1_DoneExecution()
Handles dbmlsync1.DoneExecution

    MsgBox(dbmlsync1.ExitCode)

End Sub
```


DownloadRow イベント

DownloadRow イベントは、Mobile Link サーバからローがダウンロードされるときにトリガされます。

構文

Public Event **DownloadRow(**
ByVal *rowData* As DbmlsyncCOM.IRowTransferData
)
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

rowData　ダウンロードされるローの詳細が含まれている IRowTransferData オブジェクト。

IRowTransferData インタフェースの詳細については、「IRowTransferData インタフェース」 373 ページを参照してください。

備考

このイベントを使用して、Mobile Link サーバからダウンロードされるローを調べます。

DownloadRow イベントを有効にするには、DownloadEventsEnabled プロパティを使用します。

「DownloadEventsEnabled プロパティ」 355 ページを参照してください。

ローのダウンロード・イベントで削除操作が発生したときは、プライマリ・キー・カラムの値しか使用できません。

例

次に示す Visual Basic .NET の例は、DownloadRow イベントでローのすべてのカラムを反復処理します。この処理で、カラム値が NULL であるかどうかが判別され、カラム名と値が出力されます。

```
Private Sub dbmlsync1_DownloadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.DownloadRow

Dim liX As Integer
For liX = 0 To rowData.ColumnCount - 1
   If VarType(rowData.ColumnValue(liX)) <> VariantType.Null Then
      ' output the non-null column value
      MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
      ", " + CStr(rowData.ColumnValue(liX)))
   Else
      ' output 'NULL' for the column value
      MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
      ", " + "NULL")
   End If
Next liX

End Sub
```

EndDownload イベント

EndDownload イベントは、同期処理のダウンロード処理終了時にトリガされます。

構文

Public Event **EndDownload(**
  long *upsertRows*,
  long *deleteRows*
)
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

upsertRows　ダウンロード処理で更新または挿入されたローの数を示します。

deleteRows　ダウンロード処理で削除されたローの数を示します。

備考

このイベントを使用して、同期のダウンロード処理終了時にカスタム・アクションを追加します。

例

次に示す Visual Basic .NET の例は、EndDownload イベントがトリガされたときに、挿入、更新、削除されたローの数とメッセージを出力します。

```
Private Sub dbmlsync1_EndDownload(
 ByVal upsertRows As Integer,
 ByVal deleteRows As Integer
)
Handles dbmlsync1.EndDownload

   MsgBox("Download complete." + _
    CStr(upsertRows) + "Rows updated or inserted" + _
    CStr(deleteRows) + "Rows deleted")

End Sub
```

EndLogScan イベント

EndLogScan イベントは、アップロード用にトランザクション・ログがスキャンされた直後にトリガされます。このイベントは、スクリプト化されたアップロードでは起動されません。

構文

Public Event **EndLogScan( )**
Member of **DbmlsyncCOM.Dbmlsync**

備考

このイベントを使用して、アップロード用にトランザクション・ログがスキャンされた直後にカスタム・アクションを追加します。

例

次に示す Visual Basic .NET の例は、EndLogScan イベントがトリガされたときにメッセージを出力します。

```
Private Sub dbmlsync1_EndLogScan()
Handles dbmlsync1.EndLogScan

    MsgBox("Scan of transaction log complete...")

End Sub
```

EndSynchronization イベント

EndSynchronization イベントは、同期の完了時にトリガされます。

構文

Public Event **EndSynchronization(**
  ByVal *exitCode* As Integer,
  ByRef *restart* As Boolean
)
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

exitCode　0 (ゼロ) 以外の値に設定されている場合は、同期エラーが発生したことを示します。

restart　このイベントが呼び出されるときは、この値は False に設定されます。イベントによって値が True に変更された場合、dbmlsync は同期を再起動します。

備考

このイベントを使用して、同期の完了時にカスタム・アクションを追加します。

例

次に示す Visual Basic .NET の例は、EndSynchronization イベントを使用して、最大で 5 つの失敗した同期を再起動します。同期の再起動にすべて失敗すると、"All restart attempts failed" というメッセージと、終了コードが出力されます。同期が成功すると、"Synchronization succeeded" というメッセージと、終了コードが出力されます。

```
' Global variable for the number of restarts
Dim numberOfRestarts As Integer

Private Sub dbmlsync1_EndSynchronization(
 ByVal ExitCode As Integer,
 ByRef restart As Boolean
)
Handles dbmlsync1.EndSynchronization

    If numberOfRestarts < 5 Then
        MsgBox("Restart Number: " + CStr(numberOfRestarts + 1))
        If ExitCode <> 0 Then
            ' restart the failed synchronization
            restart = True
```

```
            numberOfRestarts = numberOfRestarts + 1
        Else
            ' the last synchronization succeeded
            MsgBox("Synchronization succeeded. " + _
              "Exit code: " + CStr(ExitCode))
        End If
    Else
        MsgBox("All restart attempts failed. " + _
            "Exit code: " + CStr(ExitCode))
    End If

End Sub
```

EndUpload イベント

EndUpload イベントは、Mobile Link サーバへのアップロード転送の直後にトリガされます。

構文

Public Event **EndUpload( )**
Member of **DbmlsyncCOM.Dbmlsync**

備考

このイベントを使用して、dbmlsync から Mobile Link サーバへのアップロード転送の直後にカスタム・アクションを追加します。

例

次に示す Visual Basic .NET の例は、EndUpload イベントがトリガされたときにメッセージを出力します。

```
Private Sub dbmlsync1_EndUpload()
Handles dbmlsync1.EndUpload

    MsgBox("End Upload")

End Sub
```

Message イベント

Message イベントは、dbmlsync が情報のログを取るときにトリガされます。

構文

Public Event **Message(_**
  ByVal *msgClass* As DbmlsyncCOM.MessageClass, _
  ByVal *msgID* As Integer, ByVal *msg* As String_
)
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

msgClass　メッセージの重大度を示します。次のいずれかの値を取ります。

- **MsgInfo**　同期の進行状況情報が含まれるメッセージです。
- **MsgDetailedInfo**　MsgInfo と似ていますが、さらに詳細が含まれています。
- **MsgWarning**　問題が発生する可能性があることを示すメッセージですが、正常な同期を妨げない問題です。
- **MsgError**　正常な同期を妨げない問題を示すメッセージです。

msgID　メッセージのユニークな識別子です。msgID が 0 の場合、メッセージにはユニークな識別子はありません。

msg　メッセージのテキストです。

備考

このイベントを使用して、dbmlsync がログを取った情報を受け取ります。特定のメッセージが生成されたときに特別な処理を実行する場合は、MsgID でメッセージを確認します。このようにすると、メッセージのテキストが変更されても、コードはそのまま機能します。

例

次に示す Visual Basic .NET の例は、dbmlsync がログを取ったメッセージをリストボックス・コントロールに追加します。

```
Private Sub dbmlsync1_Message(
 ByVal msgClass As DbmlsyncCOM.MessageClass,
 ByVal msgId As Integer, ByVal msg As String
)
Handles dbmlsync1.Message

    Select Case msgClass
        Case DbmlsyncCOM.MessageClass.MsgError
            lstMessages.Items.Add("Error: " + msg)
        Case DbmlsyncCOM.MessageClass.MsgWarning
            lstMessages.Items.Add("Warning: " + msg)
        Case DbmlsyncCOM.MessageClass.MsgInfo
            lstMessages.Items.Add("Info: " + msg)
        Case DbmlsyncCOM.MessageClass.MsgDetailedInfo
            lstMessages.Items.Add("DetInfo: " + msg)
    End Select

End Sub
```

例

次に示す Visual Basic .NET の例は、エラーを処理するように Message イベントを設定します。エラー・メッセージは、lstMessages という ListBox コントロールに追加されます。

```
Private Sub dbmlsync1_Message(ByVal msgClass As DbmlsyncCOM.MessageClass, ByVal msgId As
Integer, ByVal msg As String) Handles dbmlsync1.Message
  If msgClass = DbmlsyncCOM.MessageClass.MsgError Then
    lstMessages.Items.Add("Error: " + msgId.ToString() + " " + msg)
  End If
End Sub
```

可能性のあるエラーの ID 値を表示するには、dbmlsync 統合コンポーネントをテスト実行します。たとえば、dbmlsync が「Mobile Link サーバに接続できません。」というエラーを返すと、Message イベントは次のエントリを lstMessages に挿入します。

```
Error: 14173 Unable to connect to MobiLink server.
```

これで、「Mobile Link サーバに接続できません。」というエラーとエラー ID 14173 を関連付けることができます。次の例では、エラー 14173 が発生したときに必ず同期をリトライするように dbmlsync 統合コンポーネントを設定します。Message イベントは、エラー 14173 が発生すると、restartSynchronization という変数を設定し、numberOfRestarts という変数をリセットします。EndSynchronization イベントは、最大 5 回、同期をリトライします。

```
' variables for restarting synchronization
Dim numberOfRestarts As Integer = 0
Dim restartSynchronization As Integer = 0


Private Sub dbmlsync1_Message
(
 ByVal msgClass As DbmlsyncCOM.MessageClass,
 ByVal msgId As Integer, ByVal msg As String ) Handles dbmlsync1.Message

 If msgClass = DbmlsyncCOM.MessageClass.MsgError Then
    lstMessages.Items.Add("Error: " + msgId.ToString() + " " + msg)

    If msgId = 14173 Then
      restartSynchronization = 1
      numberOfRestarts = 0
    End If
 End If
End Sub

Private Sub dbmlsync1_EndSynchronization(ByVal ExitCode As Integer, _
   ByRef restart As Boolean _
   ) Handles dbmlsync1.EndSynchronization


   If restartSynchronization = 1 Then
     If numberOfRestarts < 5 Then
       restart = True
       numberOfRestarts = numberOfRestarts + 1
     End If
   End If
End Sub
```

ProgressIndex イベント

ProgressIndex イベントは、dbmlsync が進行状況バーを更新したときにトリガされます。

構文

Public Event **ProgressIndex(_**
  ByVal *index* As Integer, _
  ByVal *max* As Integer _
)
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

index　同期の進行状況を表す整数。

max　進行状況の最大値。完了した割合 (パーセント) は、index/max x 100 で計算できます。この値が 0 である場合、最大値はこのイベントが最後に呼び出されてから変更されていません。

備考

このイベントを使用して、進行状況バーのような、進行状況を示すインジケータを更新します。

例

次に示す Visual Basic .NET の例は、Index の値に基づいて、進行状況バー・コントロールを更新します。インデックスの最大値は、同期の開始時に設定されます。

```
Private Sub dbmlsync1_ProgressIndex(
 ByVal index As Integer,
 ByVal max As Integer
)
Handles dbmlsync1.ProgressIndex

    If max <> 0 Then
        ProgressBar1.Maximum = max
    End If
    ProgressBar1.Value = index

End Sub
```

ProgressMessage イベント

ProgressMessage イベントは、同期の進行状況情報が変更されたときにトリガされます。

構文

Public Event **ProgressMessage(** ByVal *msg* As String **)**
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

msg　新しい進行状況の文字列。

備考

このイベントを使用して、通常は dbmlsync の進行状況バーに表示される文字列を受け取ります。

例

次に示す Visual Basic .NET の例は、ProgressMessage イベントがトリガされたときに進行状況のラベルの値を設定します。

```
Private Sub dbmlsync1_ProgressMessage(
 ByVal msg As String
)
Handles dbmlsync1.ProgressMessage

    lblProgressMessage.Text = msg

End Sub
```

SetTitle イベント

SetTitle イベントは、ステータス情報が変更されたときにトリガされます。dbmlsync ユーティリティでは、この情報はタイトル・バーに表示されます。

構文

Public Event **SetTitle(** ByVal *title* **)** As String
)
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

title dbmlsync ウィンドウのタイトル・バー内のタイトル。

備考

このイベントを使用して、dbmlsync ウィンドウの値が変更されたときに、このウィンドウに通常表示されるタイトルを受け取ります。

例

次に示す Visual Basic .NET の例は、SetTitle イベントがトリガされたときに Windows フォームのタイトルを設定します。

```
Private Sub dbmlsync1_SetTitle(
 ByVal title As String
)
Handles dbmlsync1.SetTitle

    Me.Text = title

End Sub
```

UploadAck イベント

UploadAck イベントは、コンポーネントがアップロードの確認を Mobile Link サーバから受信した後にトリガされます。

構文

Public Event **UploadAck(_**
 ByVal *status* As DbmlsyncCOM.UploadAckStatus _
)
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

status アップロードが処理された後に、Mobile Link によってリモートに返されたステータスを示します。以下のいずれかの値になります。

- **StatCommitted** Mobile Link サーバがアップロードを受信し、コミットしたことを示します。

- **StatRetry**　アップロードの開始位置であるログ・オフセットの値が、Mobile Link サーバと dbmlsync で異なっていたことを示します。Mobile Link サーバは、アップロードをコミットしませんでした。コンポーネントは、Mobile Link サーバのログ・オフセットから開始して別のアップロードを送信します。
- **StatFailed**　Mobile Link サーバがアップロードをコミットしなかったことを示します。

備考

このイベントを使用して、dbmlsync がアップロードの確認を Mobile Link サーバから受信した後にカスタム・アクションを追加します。

例

次に示す Visual Basic .NET の例は、UploadAck イベントがトリガされたときに、アップロードが失敗していた場合にメッセージを出力します。

```
Private Sub dbmlsync1_UploadAck(ByVal status As DbmlsyncCOM.UploadAckStatus) Handles
dbmlsync1.UploadAck

    If status = DbmlsyncCOM.UploadAckStatus.StatFailed Then
        MsgBox("Upload Failed")
    End If

End Sub
```

UploadRow イベント

UploadRow イベントは、Mobile Link サーバにローがアップロードされるときにトリガされます。

構文

Public Event **UploadRow(**
  ByVal *rowData* As DbmlsyncCOM.IRowTransferData
)
Member of **DbmlsyncCOM.Dbmlsync**

パラメータ

rowData　アップロードされるローの詳細が含まれている IRowTransferData オブジェクト。

「IRowTransferData インタフェース」 373 ページを参照してください。

備考

このイベントを使用して、Mobile Link サーバにアップロードされるローを調べます。

UploadRow イベントを有効にするには、UploadEventsEnabled プロパティを使用します。
「UploadEventsEnabled プロパティ」 354 ページを参照してください。

例

次に示す Visual Basic .NET の例は、UploadRow イベントでローのすべてのカラムを反復処理します。この処理で、カラム値が NULL であるかどうかが判別され、カラム名と値が出力されます。

```
Private Sub dbmlsync1_UploadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.UploadRow

Dim liX As Integer
For liX = 0 To rowData.ColumnCount - 1
   If VarType(rowData.ColumnValue(liX)) <> VariantType.Null Then
      ' output the non-null column value
      MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
      ", " + CStr(rowData.ColumnValue(liX)))
   Else
      ' output 'NULL' for the column value
      MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
      ", " + "NULL")
   End If
Next liX

End Sub
```

WaitingForUploadAck イベント

WaitingForUploadAck イベントは、コンポーネントが Mobile Link サーバからのアップロード確認の待機を開始したときにトリガされます。

構文

Public Event **WaitingForUploadAck( )**
Member of **DbmlsyncCOM.Dbmlsync**

備考

このイベントを使用して、コンポーネントが Mobile Link サーバからのアップロード確認を待機しているときにカスタム・アクションを追加します。

例

次に示す Visual Basic .NET の例は、WaitingForUploadAck イベントがトリガされたときにメッセージを出力します。

```
Private Sub dbmlsync1_WaitingForUploadAck()
Handles dbmlsync1.WaitingForUploadAck

   MsgBox("Waiting for Upload Acknowledgement")

End Sub
```


IRowTransferData インタフェース

Public Interface **IRowTransferData**
Member of **DbmlsyncCOM**

UploadRow イベントと DownloadRow イベントは、パラメータとして DbmlsyncCOM.IRowTransferData オブジェクトを受け入れ、アップロードされたローとダウンロードされたローを調べます。このインタフェースは、テーブル名、ローの操作、カラム名などのローの詳細な情報を定義します。

RowOperation プロパティ

ローで実行される操作を指定します。

構文

Public Property **RowOperation( )** As DbmlsyncCOM.RowEventOp
Member of **DbmlsyncCOM.IRowTransferData**

備考

このプロパティの値は、次のいずれかです。

OpInsert ローが挿入されました。

OpUpdate ローが更新されました。

OpDelete ローが削除されました。

OpTruncate テーブルがトランケートされました (テーブルのすべてのローが削除されました)。RowOperation プロパティがこの値を保持しているときは、ColumnName プロパティと ColumnValue プロパティは無効な情報を返します。

「注意：」 DownloadRow イベントの場合、アップサート (更新または挿入) 操作に値 OpInsert が指定されます。

TableName プロパティ

アップロード操作またはダウンロード操作が発生するテーブルの名前です。

構文

Public Property **TableName( )** As String
Member of **DbmlsyncCOM.IRowTransferData**

備考

TableName プロパティは、アップロード操作またはダウンロード操作が発生するテーブルの名前を指定します。次の例は、UploadRow イベントでの TableName プロパティの使い方を示しています。

「UploadRow イベント」 371 ページを参照してください。

例

次に Visual Basic .NET の例を示します。

```
Private Sub dbmlsync1_UploadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.UploadRow

   MsgBox ("Table name:" + rowData.TableName)

End Sub
```

ColumnName プロパティ

アップロード操作またはダウンロード操作が発生するローのカラム名を取得します。

構文

Public Property **ColumnName(**ByVal *index* As Integer**)** As Object
Member of **DbmlsyncCOM.IRowTransferData**

パラメータ

index　取得するカラム名を指定するゼロベースの整数。インデックス値の範囲は、0 から、ColumnCount プロパティの値未満までです。

「ColumnCount プロパティ」 376 ページを参照してください。

備考

同じインデックスで ColumnValue プロパティを使用して、対応するカラム値を取得できます。

例

次に示す Visual Basic .NET の例は、UploadRow イベントでローのすべてのカラムを反復処理します。この処理で、カラム値が NULL であるかどうかが判別され、カラム名と値が出力されます。

「UploadRow イベント」 371 ページを参照してください。

```
Private Sub dbmlsync1_UploadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.UploadRow

Dim liX As Integer
For liX = 0 To rowData.ColumnCount - 1
   If VarType(rowData.ColumnValue(liX)) <> VariantType.Null Then
      ' output the non-null column value
      MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
      ", " + CStr(rowData.ColumnValue(liX)))
   Else
      ' output 'NULL' for the column value
      MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
      ", " + "NULL")
```



```
    End If
Next liX

End Sub
```

ColumnValue プロパティ

アップロード操作またはダウンロード操作が発生するカラムの値を取得します。

構文

Public Property **ColumnValue(** ByVal *index* As Integer **)** As Object
Member of **DbmlsyncCOM.IRowTransferData**

パラメータ

index　取得するカラム値を指定するゼロベースの整数。インデックス値の範囲は、0 から、ColumnCount プロパティの値未満までです。

「ColumnCount プロパティ」 376 ページを参照してください。

備考

更新操作が発生した場合、このプロパティで指定されるカラム値は更新が適用された後の値です。

同じインデックスで ColumnName プロパティを使用して、対応するカラム名を取得できます。

BLOB のカラム値を、このプロパティを通じて使用することはできません。BLOB のカラムが検出された場合、ColumnValue は文字列 "(blob)" です。

例

次に示す Visual Basic .NET の例は、UploadRow イベントでローのすべてのカラムを反復処理します。この処理で、カラム値が NULL であるかどうかが判別され、カラム名と値が出力されます。

「UploadRow イベント」 371 ページを参照してください。

```
Private Sub dbmlsync1_UploadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.UploadRow

Dim liX As Integer
For liX = 0 To rowData.ColumnCount - 1
    If VarType(rowData.ColumnValue(liX)) <> VariantType.Null Then
        ' output the non-null column value
        MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
        ", " + CStr(rowData.ColumnValue(liX)))
    Else
        ' output 'NULL' for the column value
        MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
        ", " + "NULL")
    End If
Next liX
```

```
End Sub
```

ColumnCount プロパティ

アップロード操作またはダウンロード操作が発生するローに含まれるカラム数です。

構文

Public Property **ColumnCount( )** As Integer
Member of **DbmlsyncCOM.IRowTransferData**

備考

ColumnCount プロパティは、アップロード操作またはダウンロード操作が発生するローのカラム数を指定します。次の例は、UploadRow イベントでの ColumnCount プロパティの使い方を示しています。

「UploadRow イベント」 371 ページを参照してください。

例

次に Visual Basic .NET の例を示します。

```
Private Sub dbmlsync1_UploadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.UploadRow

    MsgBox "Number of Columns:" + CStr(rowData.ColumnCount)

End Sub
```


dbmlsync の DBTools インタフェース

目次

dbmlsync の DBTools インタフェースの概要

データベース・ツール (DBTools) は、同期を含むデータベース管理をアプリケーションに統合するために使用できるライブラリです。データベース管理ユーティリティはすべて、DBTools によって構築されます。

「データベース・ツール・インタフェース」『SQL Anywhere サーバ - プログラミング』を参照してください。

dbmlsync 用の DBTools インタフェースを使用することで、Mobile Link 同期クライアント・アプリケーションに同期機能を統合できます。たとえば、このインタフェースを使用し、カスタム・ユーザ・インタフェースに dbmlsync の出力メッセージを表示できます。

dbmlsync 用の DBTools インタフェースは、Mobile Link 同期クライアントを設定および実行する次の要素から構成されます。

- **a_sync_db 構造体**　この構造体は、dbmlsync コマンド・ライン・オプションに対応する設定を保持します。これらの設定によって同期をカスタマイズできます。この構造体には、同期と進行状況情報を受け取るコールバック関数へのポインタも含まれます。

  「a_sync_db 構造体」『SQL Anywhere サーバ - プログラミング』を参照してください。

- **a_syncpub 構造体**　この構造体は、パブリケーション情報を保持します。同期用パブリケーションのリンク・リストを指定できます。

  「a_syncpub 構造体」『SQL Anywhere サーバ - プログラミング』を参照してください。

- **DBSynchronizeLog 関数**　この関数は同期処理を開始します。この関数のパラメータは、a_sync_db インスタンスへのポインタだけです。

  「DBSynchronizeLog 関数」『SQL Anywhere サーバ - プログラミング』を参照してください。

dbmlsync 統合コンポーネント

dbmlsync の DBTools インタフェースを使用する代わりに、dbmlsync API を使用することもできます。

「dbmlsync API」 319 ページを参照してください。

dbmlsync の DBTools インタフェースの設定

この項では、dbmlsync の DBTools インタフェースの基本的な使用手順を示します。

DBTools ライブラリの詳細については、「データベース・ツール・インタフェースの概要」『SQL Anywhere サーバ - プログラミング』を参照してください。

ご使用の開発環境でのインポート・ライブラリの使用についての詳細は、「データベース・ツール・インタフェースの使い方」『SQL Anywhere サーバ - プログラミング』を参照してください。

♦ C や C++ で作成された DBTools インタフェースを使用して dbmlsync の設定と起動を行うには、次の手順に従います。

1. DBTools ヘッダ・ファイルをインクルードします。

   DBTools ヘッダ・ファイル *dbtools.h* は、DBTools ライブラリのエントリ・ポイントをリストし、必要なデータ型を定義します。

   ```
   #include "dbtools.h"
   ```

2. DBTools インタフェースを起動します。

   - a_dbtools_info 構造体の宣言と初期化を行います。

     ```
     a_dbtools_info   info;
     short ret;
     ...
     // clear a_dbtools_info fields
     memset( &info, 0, sizeof( info ) );
     info.errorrtn = dbsyncErrorCallBack;
     ```

     dbsyncErrorCallBack はエラー・メッセージを処理する関数であり、この作業の手順 4 で定義します。

   - DBToolsInit 関数を使用して DBTools を初期化します。

     ```
     ret = DBToolsInit( &info );
     if( ret != 0 ) {
      printf("dbtools initialization failure ¥n");
     }
     ```

     DBTools の初期化の詳細については、次の項を参照してください。

     - 「データベース・ツール・インタフェースの使い方」『SQL Anywhere サーバ - プログラミング』
     - 「a_dbtools_info 構造体」『SQL Anywhere サーバ - プログラミング』
     - 「DBToolsInit 関数」『SQL Anywhere サーバ - プログラミング』

3. a_sync_db 構造体を初期化します。

   - a_sync_db インスタンスを宣言します。たとえば、次のように dbsync_info というインスタンスを宣言します。

     ```
     a_sync_db dbsync_info;
     ```

   - a_sync_db 構造体のフィールドをクリアします。

```
memset( &dbsync_info, 0, sizeof( dbsync_info ) );
```

- 必須の a_sync_db のフィールドを設定します。

  ```
  dbsync_info.version = DB_TOOLS_VERSION_NUMBER;
  dbsync_info.output_to_mobile_link = 1;
  dbsync_info.default_window_title
    = "dbmlsync dbtools sample";
  ```

- データベース接続文字列を設定します。

  ```
  dbsync_info.connectparms = "uid=DBA;pwd=sql";
  ```

  データベース接続パラメータの詳細については、「-c オプション」 151 ページを参照してください。

- 同期をカスタマイズするための他の a_sync_db のフィールドを設定します。

  ほとんどのフィールドは、dbmlsync コマンド・ライン・オプションに対応しています。この対応の詳細については、*dbtools.h* を参照してください。

  次の例では、冗長オペレーションが指定されています。

  ```
  dbsync_info.verbose_upload = 1;
  dbsync_info.verbose_option_info = 1;
  dbsync_info.verbose_row_data = 1;
  dbsync_info.verbose_row_cnts = 1;
  ```

a_sync_db のフィールドの詳細については、「a_sync_db 構造体」 『SQL Anywhere サーバ - プログラミング』を参照してください。

4. 同期中にフィードバックを受け取るコールバック関数を作成し、これらの関数を適切な a_sync_db のフィールドに割り当てます。

   次の関数は、標準出力ストリームを使用して dbmlsync のエラー、ログ、進行状況情報を表示します。

   DBTools のコールバック関数の詳細については、「コールバック関数の使い方」 『SQL Anywhere サーバ - プログラミング』を参照してください。

   - たとえば、生成されたエラー・メッセージを処理する dbsyncErrorCallBack という関数を作成します。

     ```
     extern short _callback dbsyncErrorCallBack( char *str )
     {
        if( str != NULL ) {
           printf( "Error Msg    %s¥n", str );
        }
        return 0;
     }
     ```

   - たとえば、生成された警告メッセージを処理する dbsyncWarningCallBack という関数を作成します。

     ```
     extern short _callback dbsyncWarningCallBack( char *str )
     {
        if( str != NULL ) {
           printf( "Warning Msg  %s¥n", str );
        }
        return 0;
     }
     ```


- たとえば、冗長情報メッセージを受け取る dbsyncLogCallBack という関数を作成します。この情報メッセージは、ウィンドウに表示する代わりにファイルに記録できます。

```
extern short _callback dbsyncLogCallBack( char *str )
{
   if( str != NULL ) {
      printf( "Log Msg      %s¥n", str );
   }
   return 0;
}
```

- たとえば、同期中に生成された情報メッセージを受け取る dbsyncMsgCallBack という関数を作成します。

```
extern short _callback dbsyncMsgCallBack( char *str )
{
   if( str != NULL ) {
      printf( "Display Msg  %s¥n", str );
   }
   return 0;
}
```

- たとえば、進行状況テキストを受け取る dbsyncProgressMessageCallBack という関数を作成します。dbmlsync ユーティリティでは、このテキストは進行状況バーの真上に表示されます。

```
extern short _callback dbsyncProgressMessageCallBack(
 char *str )
{
   if( str != NULL ) {
      printf( "ProgressText %s¥n", str );
   }
   return 0;
}
```

- たとえば、進行状況インジケータまたは進行状況バーを更新するために情報を受け取る dbsyncProgressIndexCallBack という関数を作成します。この関数は、次の 2 つのパラメータを受け取ります。
  - **index**　同期の現在の進行状況を表す整数。
  - **max**　進行状況の最大値。この値が 0 である場合、最大値はこのイベントが最後に呼び出されてから変更されていません。

```
extern short _callback dbsyncProgressIndexCallBack
(a_sql_uint32 index, a_sql_uint32 max )
{
   printf( "ProgressIndex    Index %d Max: %d¥n",
       index, max );
   return 0;
}
```

次に、このコールバックへの呼び出しの一般的な順序を示します。

```
// example calling sequence
dbsyncProgressIndexCallBack( 0, 100 );
dbsyncProgressIndexCallBack( 25, 0 );
dbsyncProgressIndexCallBack( 50, 0 );
dbsyncProgressIndexCallBack( 75, 0 );
dbsyncProgressIndexCallBack( 100, 0 );
```

この順序では、0% 完了、25% 完了、50% 完了、75% 完了、100% 完了に設定された進行状況バーが表示されます。

- たとえば、ステータス情報を受け取る dbsyncWindowTitleCallBack という関数を作成します。dbmlsync ユーティリティでは、この情報はタイトル・バーに表示されます。

```
extern short _callback dbsyncWindowTitleCallBack(
 char *title )
{
   printf( "Window Title     %s¥n", title );
   return 0;
}
```

- dbsyncMsgQueueCallBack 関数は、遅延またはスリープが必要な場合に呼び出します。この関数は、*dllapi.h* に定義されている次の値のいずれかを返す必要があります。
  - **MSGQ_SLEEP_THROUGH**　要求したミリ秒の間ルーチンがスリープしたことを示します。ほとんどの場合はこの値が返されるようにします。
  - **MSGQ_SHUTDOWN_REQUESTED**　できるだけ早く同期を終了したいことを示します。
  - **MSGQ_SYNC_REQUESTED**　要求したミリ秒に達しないうちにルーチンがスリープ状態を終了したことと、同期が現在進行中でない場合はただちに次の同期をとり始めることを示します。

```
extern short _callback dbsyncMsgQueueCallBack(
  a_sql_uint32 sleep_period_in_milliseconds )
{

 printf( "Sleep %d ms¥n", sleep_period_in_milliseconds );
 Sleep( sleep_period_in_milliseconds );
 return MSGQ_SLEEP_THROUGH;
}
```

- 適切な a_sync_db 同期構造体のフィールドにコールバック関数のポインタを割り当てます。

```
// set call back functions
dbsync_info.errorrtn    = dbsyncErrorCallBack;
dbsync_info.warningrtn  = dbsyncWarningCallBack;
dbsync_info.logrtn      = dbsyncLogCallBack;
dbsync_info.msgrtn      = dbsyncMsgCallBack;
dbsync_info.msgqueuertn = dbsyncMsgQueueCallBack;
dbsync_info.progress_index_rtn
    = dbsyncProgressIndexCallBack;
dbsync_info.progress_msg_rtn
    = dbsyncProgressMessageCallBack;
dbsync_info.set_window_title_rtn
    = dbsyncWindowTitleCallBack;
```

5. どのパブリケーションを同期するかを指定する a_syncpub 構造体のリンク・リストを作成します。

   リンク・リスト内の各ノードは、dbmlsync コマンド・ライン上の -n オプションの 1 つのインスタンスに対応します。

   - a_syncpub インスタンスを宣言します。たとえば、これを publication_info とします。

```
a_syncpub publication_info;
```


- a_syncpub フィールドを初期化し、同期するパブリケーションを指定します。

  たとえば、単一の同期セッションで template_p1 パブリケーションと template_p2 パブリケーションをまとめて識別するには、次のように指定します。

  ```
  publication_info.next = NULL; // linked list terminates
  publication_info.pub_name = "template_p1,template_p2";
  publication_info.ext_opt  = "sv=template_ver1";
  publication_info.alloced_by_dbsync = 0;
  ```

  これは、dbmlsync コマンド・ラインで -n template_p1,template_p2 と指定するのと同じです。

  ext_opt フィールドを使用して指定された関連スクリプト・バージョンは、dbmlsync -eu オプションと同じ機能を提供します。

  「-eu オプション」 162 ページを参照してください。

- a_sync_db インスタンスの upload_defs フィールドにパブリケーション構造体を割り当てます。

  ```
  dbsync_info.upload_defs   = &publication_info;
  ```

a_syncpub 構造体のリンク・リストを作成できます。リンク・リスト内の各 a_syncpub インスタンスは、dbmlsync コマンド・ライン上の -n オプションの 1 つの定義に相当します。

「-n オプション」 168 ページと「a_syncpub 構造体」『SQL Anywhere サーバ - プログラミング』を参照してください。

6. DBSynchronizeLog 関数を使用して dbmlsync を実行します。

   次に示すコード内の sync_ret_val には、成功した場合は戻り値 0、失敗した場合は 0 以外の値が格納されます。

   ```
   short sync_ret_val;
   printf("Running dbmlsync using dbtools interface...¥n");
   sync_ret_val = DBSynchronizeLog(&dbsync_info);
   printf("¥n Done... synchronization return value is: %I ¥n", sync_ret_val);
   ```

   手順 6 は、同じパラメータ値または異なるパラメータ値を指定して複数回繰り返すことができます。

7. DBTools インタフェースをシャットダウンします。

   DBToolsFini 関数は、DBTools リソースを開放します。

   ```
   DBToolsFini( &info );
   ```

   「DBToolsFini 関数」『SQL Anywhere サーバ - プログラミング』を参照してください。

スクリプト化されたアップロード

目次

スクリプト化されたアップロードの概要

スクリプト化されたアップロードは、SQL Anywhere リモート・データベースを使用する Mobile Link アプリケーションだけに適用されます。

警告
スクリプト化されたアップロードを実装した場合、dbmlsync では、アップロードするデータの判別にトランザクション・ログは使用されません。その結果、スクリプトがすべての変更を取得しなかった場合は、リモート・データベース上のデータが失われることがあります。このため、ほとんどのアプリケーションの同期方法としては、ログベースの同期をおすすめします。

ほとんどの Mobile Link アプリケーションでは、データベースのトランザクション・ログによってアップロードが判別されるので、最後のアップロード以降にリモート・データベースに加えられた変更が同期されます。これは、ほとんどのアプリケーションでは適切な設計であり、リモート上のデータが失われないようにしています。

しかし、まれに、トランザクション・ログを無視して、アップロードを定義する必要がある場合があります。スクリプト化されたアップロードを使用すると、アップロードするデータを正確に定義できます。スクリプト化されたアップロードを使用する場合は、リモート・データベースのトランザクション・ログを保持する必要はありません。トランザクション・ログはかなりの領域を占有するため、小型デバイスでは考慮する必要があります。ただし、トランザクション・ログは、データベースのバックアップとリカバリには非常に重要であり、データベースのパフォーマンスの向上に役立ちます。

スクリプト化されたアップロードを実装するには、特殊なパブリケーションを作成し、そのパブリケーションで、作成したストアド・プロシージャの名前を指定します。ストアド・プロシージャは、統合データベースで挿入、更新、または削除するローが含まれる結果セットを返すことによって、アップロードを定義します。

「注意：」スクリプト化されたアップロードとアップロード・スクリプトを混同しないように注意してください。アップロード・スクリプトは、アップロードによりどのような処理を行うかを Mobile Link サーバに指示するための、統合データベース上の Mobile Link イベント・スクリプトです。スクリプト化されたアップロードを使用する場合は、統合データベースにアップロードを適用するためにアップロード・スクリプトを作成し、ダウンロードするデータを指定するためにダウンロード・スクリプトを作成する必要があります。

シナリオ

スクリプト化されたアップロードが役立つシナリオを以下に示します。

- リモート・データベースは記憶領域が限定されているデバイスで実行中で、トランザクション・ログを格納するのに十分な領域がない場合
- すべてのリモート・データベースからすべてのデータをアップロードして、新しい統合データベースを作成する場合
- 統合データベースにアップロードされる変更を特定するカスタム論理を作成する場合

警告

スクリプト化されたアップロードを実装する前に、この章全体をお読みください。次の点に特に注意してください。

- スクリプト化されたアップロードを正しく設定しないと、データが失われます。
- スクリプト化されたアップロードを実装する場合は、dbmlsync が通常処理するデータを維持または参照する必要があります。これには、データの更新前と更新後のイメージや、同期の進行状況が含まれます。
- スクリプト化されたアップロードを介して同期している間、リモート・データベース上のテーブルをロックする必要があります。ログベースの同期を行う場合は、ロックする必要はありません。
- スクリプト化されたアップロードを使用してトランザクション単位のアップロードを実装するのは非常に困難です。

スクリプト化されたアップロードの設定

次の手順では、Mobile Link 同期が設定済みであることを前提に、スクリプト化されたアップロードの設定に必要なタスクの概要について説明します。

♦ スクリプト化されたアップロードの設定の概要

1. アップロードするローを識別するストアド・プロシージャを作成します。テーブルごとに 3 つのストアド・プロシージャ (アップロード、挿入、削除用に 1 つずつ) を定義できます。

   「スクリプト化されたアップロードのストアド・プロシージャの定義」 396 ページを参照してください。

2. WITH SCRIPTED UPLOAD というキーワードが含まれ、ストアド・プロシージャの名前を指定するパブリケーションを作成します。

   「スクリプト化されたアップロードのパブリケーションの作成」 400 ページを参照してください。

スクリプト化されたアップロードを使用するときは、dbmlsync の LockTables 拡張オプションにデフォルト設定を使用することをおすすめします。

スクリプト化されたアップロードの多くの問題は、LockTables にデフォルト設定を使用することで防ぐことができます。この設定により、dbmlsync は、すべての同期テーブルのロックを取得してから、アップロードを構築できます。これにより、スクリプトがアップロードを構築中に、他の接続が同期テーブルを変更するのを防ぐことができます。また、この設定を行うと、スクリプトがアップロードを構築中に、同期テーブルに影響するコミットされていないトランザクションをなくすことができます。

クイック・スタートのためのその他の資料

- 「スクリプト化されたアップロードの例」 401 ページ

スクリプト化されたアップロードの設計上の考慮事項

1 ローあたり 1 つの操作

アップロードには、1 つのローに対して複数の操作 (挿入、更新、または削除) を含めることはできません。ただし、複数の操作を 1 つのアップロード操作にまとめることはできます。たとえば、ローを挿入してから更新する場合は、2 つの操作を、最後の値を 1 回挿入する操作に置き換えることができます。

操作の順序

アップロードを統合データベースに適用すると、挿入と更新操作の後に削除操作が適用されます。特定のテーブル内での操作順序について他の想定をすることはできません。

競合の解決

同期と同期の間に複数のデータベースでローが更新されると、競合が発生します。アップロード内の各更新操作には、更新中のローの更新前イメージが含まれているので、Mobile Link サーバは競合を検出することができます。更新前イメージは、最後にアップロードまたはダウンロードに成功したローの全カラムの値です。アップロードが提供されたときに、更新前イメージが統合データベースの値と一致しないと、Mobile Link サーバは競合を検出します。

アプリケーションで競合を検出する必要があり、スクリプト化されたアップロードを使用している場合は、リモート・データベースで、最後にアップロードまたはダウンロードに成功した各ローの値を追跡する必要があります。これにより、正しい更新前イメージをアップロードできます。

更新前イメージのデータを維持する 1 つの方法は、同期テーブルと同一の更新前イメージ・テーブルを作成することです。次に、更新を実行するたびに更新前イメージ・テーブルを移植するトリガを同期テーブルに作成します。アップロードに成功したら、更新前イメージ・テーブルのローを削除できます。

競合解決の実装の例については、「スクリプト化されたアップロードの例」 401 ページを参照してください。

競合を処理しない

競合の検出を処理する必要がない場合は、更新前イメージを追跡しなければ、アプリケーションを大幅に簡素化できます。代わりに、挿入操作として更新をアップロードします。次に、ローが存在しない場合は挿入し、存在する場合はローを更新する upload_insert スクリプトを統合データベースに作成します。SQL Anywhere 統合データベースを使用している場合、この操作は、upload_insert スクリプトの INSERT 文にある ON EXISTING 句を使用して実行できます。

「INSERT 文」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

競合を処理せず、複数のリモート・データベースで同じローが変更されると、最後に同期したリモート・データベースによって、それまでの変更が上書きされます。

強制的な競合解決の処理

削除操作では、アップロードされたローのプライマリ・キーが正しいことが重要です。しかし、ほとんどの場合、非プライマリ・キー・カラムの値が、統合データベースの値と一致しているか

どうかは問題ではありません。非プライマリ・キー・カラムが重要なのは、Mobile Link サーバで強制的な競合モードが使用されるときだけです。この場合、カラムのすべての値が、統合データベースの upload_old_row_insert スクリプトに渡されます。このスクリプトを実装した方法によっては、非プライマリ・キー・カラムを正しい値にする必要があります。

「強制的な競合解決」『Mobile Link - サーバ管理』を参照してください。

ロッキング

スクリプト化されたアップロードの多くの問題は、dbmlsync 拡張オプション LockTables にデフォルト設定を使用することで防ぐことができます。この設定により、dbmlsync は、すべての同期テーブルの排他ロックを取得してから、アップロードを構築できます。これにより、スクリプトがアップロードを構築中に、他の接続が同期テーブルを変更するのを防ぐことができます。また、この設定を行うと、スクリプトがアップロードを構築中に、同期テーブルに影響するコミットされていないトランザクションをなくすことができます。

テーブルのロックをオフにする必要がある場合は、「テーブルをロックしない場合のスクリプト化されたアップロード」 392 ページを参照してください。

冗長なアップロード

多くの場合、リモート・データベースで各操作をアップロードするのは、一度だけです。この操作を支援するため、Mobile Link は各サブスクリプションの進行状況値を維持します。進行状況値は、デフォルトで、dbmlsync が正常な最終アップロードを構築し始めた時間です。この進行状況値は、sp_hook_dbmlsync_set_upload_end_progress フックを使用して異なる値で上書きできます。

「sp_hook_dbmlsync_set_upload_end_progress」 307 ページを参照してください。

アップロード・プロシージャの 1 つが呼び出されるたびに、#hook_dict テーブルを介して値が渡されます。渡される値としては、'start progress' や 'end progress' があります。これらの値は、リモート・データベースへの変更が構築中のアップロードに含まれるようにする時間を定義します。'start progress' 前に発生した操作は、すでにアップロードされています。'end progress' 後に発生する操作は、次の同期中にアップロードされます。

不明なアップロード・ステータス

スクリプト化されたアップロードの実装でよくある間違いは、sp_hook_dbmlsync_upload_end または sp_hook_dbmlsync_end フックを使用して、アップロードが正常に統合データベースに適用されたかどうだけを通知できるストアド・プロシージャを作成してしまうことです。この方法は信頼性に欠けます。

たとえば、次の例では、各ローの 1 ビットを使用して挿入を処理し、ローをアップロードする必要があるかどうかを追跡しようとしています。ビットは、ローが挿入されると設定され、アップロードが正常にコミットされると sp_hook_dbmlsync_upload_end フックで解除されます。

```
//
// DO NOT DO THIS!
//
CREATE TABLE t1 (
   pk    integer primary key,
   val      varchar( 256 ),
   to_upload   bit DEFAULT 1
);
```



```
CREATE PROCEDURE t1_ins()
RESULT( pk integer, val varchar(256) )
BEGIN
   SELECT pk, val
   FROM t1
   WHERE to_upload = 1;
END;

CREATE PROCEDURE sp_hook_dbmlsync_upload_end()
BEGIN
   DECLARE    upload_status   varchar(256);

   SELECT value
   INTO upload_status
   FROM #hook_dict
   WHERE name = 'upload status';

   if upload_status = 'committed' THEN
      UPDATE t1 SET to_upload = 0;
   END IF
END;

    CREATE PUBLICATION p1 WITH SCRIPTED UPLOAD (
       TABLE t1 USING ( PROCEDURE t1_ins FOR UPLOAD INSERT )
    );
```

この方法は、ほとんどの場合に機能します。ハードウェアまたはソフトウェアの障害が発生し、アップロードが送信された後、サーバで受信確認される前に、dbmlsync が停止されると、失敗します。この場合、アップロードは統合データベースに適用されますが、sp_hook_dbmlsync_upload_end フックは呼び出されず、to_upload ビットは解除されません。その結果、次回の同期では、アップロード済みのローに挿入がアップロードされます。この場合、通常は統合データベースで重複プライマリ・キー・エラーが発生し、同期が失敗します。

そのほかには、Mobile Link サーバとの通信が、アップロードが送信された後、受信確認される前に失われた場合に、問題が発生します。この場合、dbmlsync は、アップロードが正常に適用されたかどうかを確認できません。dbmlsync は sp_hook_dbmlsync_upload_end フックを呼び出して、アップロード・ステータスを不明に設定します。フックが作成されると、to_upload ビットが解除されるのを防ぐことができます。サーバによってアップロードが適用されなかった場合、問題は発生しません。ただし、アップロードが適用されると、前述と同じ問題が発生します。いずれの場合も、問題を手動で解決するまで、影響を受けたリモート・データベースを再び同期することはできません。

ダウンロード時のデータ損失の防止

スクリプト化されたアップロードを使用すると、リモート・データベースでアップロードが必要なデータが、統合データベースからダウンロードされたデータで上書きされる可能性があります。この場合、リモート・データベースでの変更内容が失われます。アップロード・プロシージャで構築するアップロードに、sp_hook_dbmlsync_set_upload_end_progress hook フックの呼び出し前にリモート・データベースでコミットされた変更内容がすべて含まれる場合は、dbmlsync によってデータ損失が防止されます。

この規則に従わなかった場合にデータがどのように失われるかを次の例に示します。

時間	
1:05:00	統合データベースとリモート・データベースの両方にあるロー R が、リモート・データベースで新しい値 R1 に更新され、変更がコミットされます。
1:06:00	統合データベースでロー R が新しい値 R2 に更新され、変更がコミットされます。
1:07:00	同期が発生します。アップロード・スクリプトは、1:00:00 より前にコミットされた操作だけが含まれるように記述されています。アップロードの構築前に発生したすべての操作がアップロードされないので、これは規則に従っていません。ロー R への変更は、1:00:00 を過ぎてから発生したのでアップロードに含まれません。サーバから受信されるダウンロードにはロー R2 が含まれます。ダウンロードが適用されると、リモート・データベースのロー R1 がロー R2 に置き換わります。リモート・データベースでの更新内容は失われます。

dbmlsync は、複数のメカニズムを使用して、sp_hook_dbmlsync_set_upload_end_progress フックの呼び出し時にコミットされていなかった変更または sp_hook_dbmlsync_set_upload_end_progress フックの呼び出し後にコミットされた変更がダウンロード内容で上書きされないようになっています。

フックの呼び出し前にコミットされた変更は保護されないので、ダウンロードの適用時に上書きが可能です。ただし、ダウンロード構築前に送信されるアップロードに変更内容が含まれれば、変更内容は Mobile Link サーバに送信されるので、サーバ側のスクリプトで、統合データベース内のデータとの間で競合を解決してから、ダウンロードを構築できます。

テーブルをロックしない場合のスクリプト化されたアップロード

デフォルトでは、dbmlsync によって同期対象のテーブルがロックされてからアップロード・スクリプトが呼び出され、このロックはダウンロードがコミットされるまで保持されます。拡張オプション LockTables をオフに設定すると、テーブルのロックを無効にできます。

可能な場合は、テーブルをロックするデフォルトの動作を使用することをおすすめします。テーブルをロックしないでスクリプト化されたアップロードを行うと、考慮する必要がある問題が増え、実行可能な正しい解決法を作成するのが難しくなります。これは、データベースの同時実行性と同期の概念をよく理解している上級ユーザだけが行ってください。

テーブルをロックしない場合の独立性レベルの使用

テーブルのロックがオフのときは、アップロードのストアド・プロシージャを実行する独立性レベルが非常に重要です。独立性レベルによって、コミットされていないトランザクションの処理方法が決まるからです。テーブルのロックがオンのときは、テーブルのロックによって、アップロードの構築時に、同期対象のテーブルに対するコミットされていない変更が防止されるので、独立性レベルは問題ではありません。

アップロードのストアド・プロシージャで独立性レベルを明示的に変更しなかった場合、ストアド・プロシージャは、dbmlsync のコマンド・ラインで指定したデータベース・ユーザのデフォルトの独立性レベルで実行されます。

データベースのデフォルトの独立性レベルは 0 ですが、テーブルをロックしないでスクリプト化されたアップロードを行うときは、アップロードのプロシージャを独立性レベル 0 で実行しないことをおすすめします。テーブルをロックしないでスクリプト化されたアップロードを実行するときに独立性レベル 0 を使用すると、コミットされていない変更がアップロードされ、次の問題が発生する可能性があります。

- コミットされていない変更がロールバックされると、間違ったデータが統合データベースに送信されます。
- コミットされていないトランザクションが完了していない場合は、トランザクションの一部だけがアップロードされ、統合データベースの整合性が失われる可能性があります。

使用できる独立性レベルは 1、2、3、またはスナップショットです。これらの独立性レベルでは、コミットされていないトランザクションはアップロードされません。

独立性レベル 1、2、または 3 を使用した場合、テーブルに対してコミットされていない変更があると、アップロードのストアド・プロシージャがブロックする可能性があります。アップロードのストアド・プロシージャは、dbmlsync が Mobile Link サーバに接続している間に呼び出されるので、サーバ接続が占有される可能性があります。独立性レベル 1 を使用した場合、SELECT 文で READPAST テーブル・ヒント句を使用することでブロックを防止できる場合があります。

スナップショット・アイソレーションを使用すると、ブロックと、コミットされていない変更の読み込みの両方を防止できます。

コミットされていない変更の損失

テーブルをロックしない場合は、同期の発生時にコミットされていない操作を処理するメカニズムが必要です。なぜこれが問題であるかを理解するために、次に例を示します。

スクリプト化されたアップロードでテーブルを同期するとします。わかりやすくするために、挿入だけをアップロードするとします。テーブルには insert_time というカラムがあります。このカラムは、各ローが挿入された時刻を示すタイムスタンプです。

各アップロードは、テーブル内で insert_time が最後の正常なアップロードと、現在のアップロードの構築を開始した時刻 (sp_hook_dbmlsync_set_upload_end_progress フックが呼び出された時刻) の間にあるすべてのコミットされたローを選択して構築します。次の処理が行われるとします。

時間	
1:00:00	正常な同期が発生します。
1:04:00	ロー R がテーブルに挿入されますが、コミットされません。R の insert_time カラムが 1:04:00 に設定されます。
1:05:00	同期が発生します。insert_time が 1:00:00 と 1:05:00 の間にあるローがアップロードされます。ロー R はコミットされていないのでアップロードされません。同期の進行状況が 1:05:00 に設定されます。
1:07:00	1:04:00 に挿入されたローがコミットされます。R の insert_time カラムは 1:04:00 のままです。

時間	
1:10:00	同期が発生します。insert_time が 1:05:00 と 1:10:00 の間にあるローがアップロードされます。ロー R は、insert_time がこの範囲内にないのでアップロードされません。つまり、ロー R はアップロードされることはありません。

一般に、同期の前に発生し、同期の後にコミットされた操作は、このように失われる可能性があります。

コミットされていないトランザクションの処理

コミットされていないトランザクションを処理する方法として最も簡単なのは、sp_hook_dbmlsync_set_upload_end_progress フックを使用して、各同期の終了進行状況を、フックの呼び出し時にコミットされていない最も古いトランザクションの開始時刻に設定する方法です。この時刻は、次のように sa_transactions システム・プロシージャを使用して確認できます。

```
SELECT min( start_time )
FROM sa_transactions()
```

この場合、アップロードのストアド・プロシージャでは、sa_transactions を使用して sp_hook_dbmlsync_set_upload_end_progress フックで計算され、#hook_dict テーブルを使用して渡される終了進行状況を無視する必要があります。ストアド・プロシージャでは、単に開始進行状況後に発生したコミットされた操作をすべてアップロードします。このようにすると、変更内容をアップロードする必要があるローがダウンロード内容で上書きされません。また、コミットされていないトランザクションがあっても、操作が適時にアップロードされます。

この解決法では、操作は失われませんが、一部の操作が複数回アップロードされる可能性があります。サーバ側のスクリプトは、複数回アップロードされる操作を処理するように記述する必要があります。この設定でローが複数回アップロードされる例を次に示します。

時間	
1:00:00	正常な同期が発生します。
2:00:00	ロー R1 が挿入されますが、コミットされません。
2:10:00	ロー R2 が挿入され、コミットされます。
3:00:00	同期が発生します。1:00 と 3:00 の間に発生した操作がアップロードされます。ロー R2 はアップロードされます。コミットされていない最も古いトランザクションの開始時刻が 2:00 なので、進行状況は 2:00 に設定されます。
4:00:00	ロー R1 がコミットされます。
5:00:00	同期が発生します。2:00 と 5:00 の間に発生した操作がアップロードされ、進行状況が 5:00 に設定されます。アップロードには R1 と R2 の両方のローが含まれます。いずれもタイムスタンプがアップロード範囲内にあるからです。したがって、R2 は 2 回アップロードされます。

統合データベースが SQL Anywhere の場合は、統合データベースの upload_insert スクリプトで INSERT ... ON EXISTING UPDATE 文を使用することで、複数回アップロードされる挿入操作を処理できます。

その他の統合データベースについては、upload_insert スクリプトから呼び出すストアド・プロシージャで似たような論理を実装できます。挿入するローとプライマリ・キーが同じであるローが統合データベースにあるかどうかを確認するだけです。ローがある場合は更新し、ない場合は新しいローを挿入します。

サーバ側に競合を検出または解決する論理がある場合は、削除操作と更新操作が複数回アップロードされることが問題になります。サーバ側で競合の検出と解決のスクリプトを記述する場合は、スクリプトで複数回のアップロードを処理できる必要があります。

統合データベースでプライマリ・キーの値を再利用できる場合は、削除操作が複数回アップロードされることが重大な問題になります。次の一連のイベントを考えてみます。

1. プライマリ・キーが 100 のロー R がリモート・データベースに挿入され、統合データベースにアップロードされます。
2. ロー R がリモート・データベースで削除され、削除操作がアップロードされます。
3. プライマリ・キーが 100 の新しいロー R' が統合データベースに挿入されます。
4. 手順 2 のロー R の削除操作がリモート・データベースからもう一度アップロードされます。これにより、ロー R' が統合データベースから間違って削除される可能性があります。

参照

- 「sa_transactions システム・プロシージャ」『SQL Anywhere サーバ - SQL リファレンス』
- 「独立性レベルの設定」『SQL Anywhere サーバ - SQL の使用法』

スクリプト化されたアップロードのストアド・プロシージャの定義

スクリプト化されたアップロードを実装するには、統合データベースで挿入、更新、または削除するローが含まれる結果セットを返すことによってアップロードを定義する、ストアド・プロシージャを作成します。

ストアド・プロシージャが呼び出されると、#hook_dict というテンポラリ・テーブルが作成されます。このテーブルには、name と value という 2 つのカラムがあります。このテーブルを使用すると、名前と値の組み合わせをストアド・プロシージャに渡すことができます。ストアド・プロシージャは、このテーブルから有用な情報を取得することができます。

次の名前と値の組み合わせが定義されています。

名前	値	説明
start progress	タイムスタンプ文字列	リモート・データベースに対するすべての変更がアップロードされるまでの時間。アップロードが反映されるのは、この時間以降に発生した操作だけです。
raw start progress	64 ビット符号なし整数	符号なし整数で示された開始進行状況
end progress	タイムスタンプ文字列	アップロード期間の終了。アップロードが反映されるのは、この時間の前に発生した操作だけです。
raw end progress	64 ビット符号なし整数	符号なし整数で示された終了進行状況
generating download exclusion list	true|false	同期がダウンロード専用またはファイルベースの場合は true。この場合、アップロードは送信されず、ダウンロードがスクリプト化されたアップロードのストアド・プロシージャで選択したローに影響しない場合、ダウンロードは適用されません(これにより、アップロードの必要がある、リモートで追加された変更が、ダウンロードによって上書きされないようにすることができます)。
publication_*n*	パブリケーション名	同期されているパブリケーション (*n* は整数)。*n* の番号は 0 から始まります。
script version	バージョン名	同期に使用される Mobile Link スクリプト・バージョン

名前	値	説明
MobiLink user	Mobile Link ユーザ名	同期対象となる Mobile Link ユーザ

「#hook_dict テーブル」 251 ページを参照してください。

スクリプト化されたアップロードのカスタム進行状況値

スクリプト化されたアップロード・プロシージャに渡された開始進行状況値と終了進行状況値は、デフォルトで、タイムスタンプを表します。終了進行状況値は、デフォルトで、dbmlsync がアップロードを構築し始めた時間です。同期の開始進行状況値は、その同期のサブスクリプションを最後に正常にアップロードしたときに使用した終了進行状況値です。ほとんどの実装には、このデフォルトの動作が適しています。

まれに、別の動作が必要な場合は、sp_hook_dbmlsync_set_upload_end_progress フックが使用されます。このフックを使用すると、アップロードに使用する終了進行状況値を設定できます。終了進行状況値には、開始進行状況値より大きい値を設定する必要があります。開始進行状況値を変更することはできません。

sp_hook_dbmlsync_set_upload_end_progress フックでは、終了進行状況値をタイムスタンプまたは符号なし整数として指定できます。アップロード・ストアド・プロシージャに対しては、いずれの形式の値も使用できます。便宜上、sa_convert_ml_progress_to_timestamp と sa_convert_timestamp_to_ml_progress 関数を使用して、2 形式間で進行状況値を変換することができます。

次の項を参照してください。

- 「sp_hook_dbmlsync_set_upload_end_progress」 307 ページ
- 「sa_convert_ml_progress_to_timestamp システム・プロシージャ」 『SQL Anywhere サーバ - SQL リファレンス』
- 「sa_convert_timestamp_to_ml_progress システム・プロシージャ」 『SQL Anywhere サーバ - SQL リファレンス』

挿入用ストアド・プロシージャの定義

挿入用のストアド・プロシージャは、CREATE PUBLICATION 文で定義したように、CREATE TABLE 文で宣言されたカラムと同じ順序で、アップロードするすべてのカラムを含む結果セットを返す必要があります。

カラムの順序

t1 と呼ばれるテーブル内にあるカラムの作成順序を確認するには、次のクエリを使用します。

```
SELECT column_name
FROM SYSTAB JOIN SYSTABCOL
  WHERE table_name = 't1'
ORDER BY column_id
```

例

挿入用のストアド・プロシージャの定義方法の詳細については、「スクリプト化されたアップロードの例」 401 ページを参照してください。

次の例では、t1 というテーブルと p1 というパブリケーションを作成します。パブリケーションは、WITH SCRIPTED UPLOAD を指定し、ストアド・プロシージャ t1_insert を挿入プロシージャとして登録します。t1_insert ストアド・プロシージャの定義の結果セットには、CREATE PUBLICATION 文にリストされたすべてのカラムが含まれますが、順序は、CREATE TABLE 文でカラムが宣言された順です。

```
CREATE TABLE t1(
  //The column ordering is taken from here
  pk integer primary key,
  c1 char( 30),
  c2,   float,
  c3 double );

CREATE PROCEDURE t1_insert ()
RESULT( pk integer, c1 char(30), c3 double )
begin
  ...
end

CREATE PUBLICATION WITH SCRIPTED UPLOAD p1(
  // Order of columns here is ignored
  TABLE t1( c3, pk, c1 ) USING (
    PROCEDURE t1_insert FOR UPLOAD INSERT
  )
)
```

削除用ストアド・プロシージャの定義

削除用のストアド・プロシージャは、CREATE PUBLICATION 文で定義したように、CREATE TABLE 文で宣言されたカラムと同じ順序で、アップロードするすべてのカラムを含む結果セットを返す必要があります。

カラムの順序

t1 と呼ばれるテーブル内にあるカラムの作成順序を確認するには、次のクエリを使用します。

```
SELECT column.name
FROM SYSTAB JOIN SYSTABCOL
  WHERE table_name = 't1'
ORDER BY column_id
```

例

削除用のストアド・プロシージャの定義方法の詳細については、「スクリプト化されたアップロードの例」 401 ページを参照してください。

次の例では、t1 というテーブルと p1 というパブリケーションを作成します。パブリケーションは、WITH SCRIPTED UPLOAD を指定し、ストアド・プロシージャ t1_delete を削除プロシージャとして登録します。t1_delete ストアド・プロシージャの定義の結果セットには、CREATE PUBLICATION 文にリストされたすべてのカラムが含まれますが、順序は、CREATE TABLE 文でカラムが宣言された順です。


```
CREATE TABLE t1(
  //The column ordering is taken from here
  pk integer primary key,
  c1 char( 30),
  c2 float,
  c3 double );

CREATE PROCEDURE t1_delete ()
RESULT( pk integer, c1 char(30), c3 double )
begin
  ...
end

CREATE PUBLICATION p1 WITH SCRIPTED UPLOAD (
  // Order of columns here is ignored
  TABLE t1( c3, pk, c1 ) USING (
    PROCEDURE t1_delete FOR UPLOAD DELETE
  )
)
```

更新用ストアド・プロシージャの定義

更新用のストアド・プロシージャは、次に示す 2 つの値セットを含む結果セットを返す必要があります。

- 最初の値セットは更新前イメージを指定します (Mobile Link サーバから最後に受信または正常にアップロードされたローの中の値)。
- 2 つ目の値セットは更新後イメージを指定します (統合データベースで更新される必要があるローの中の値)。

つまり、更新用のストアド・プロシージャは、挿入または削除用のストアド・プロシージャの 2 倍のカラムを持つ結果セットを返す必要があります。

例

更新用のストアド・プロシージャの定義方法の詳細については、「スクリプト化されたアップロードの例」 401 ページを参照してください。

次の例では、t1 というテーブルと p1 というパブリケーションを作成します。パブリケーションは、WITH SCRIPTED UPLOAD を指定し、ストアド・プロシージャ t1_update を更新プロシージャとして登録します。パブリケーションは、同期させる 3 つのカラム (pk、c1、c3) を指定します。更新プロシージャは、6 つのカラムを持つ結果セットを返します。最初の 3 つのカラムには、pk,、c1、c3 のカラムの更新前イメージが含まれています。2 つ目の 3 つのカラムには、同じカラムの更新後イメージが含まれています。どちらの場合も、カラムの順序は、CREATE PUBLICATION 文の順序ではなく、テーブルが作成されたときの順序です。

```
CREATE TABLE t1(
  //Column ordering is taken from here
  pk integer primary key,
  c1 char( 30),
  c2 float,
  c3 double );

CREATE PROCEDURE t1_update ()
```

```
RESULT( preimage_pk integer, preimage_c1 char(30), preimage_c3 double,
postimage_pk integer, postimage_c1 char(30), postimage_c3 double  )
BEGIN
  ...
END

CREATE PUBLICATION p1 WITH SCRIPTED UPLOAD (
      // Order of columns here is ignored
      TABLE t1( c3, pk, c1 ) USING (
  PROCEDURE t1_update FOR UPLOAD UPDATE
  )
)
```

スクリプト化されたアップロードのパブリケーションの作成

スクリプト化されたアップロード・パブリケーションを作成するには、キーワード WITH SCRIPTED UPLOAD を使用して、USING 句でストアド・プロシージャを指定します。

スクリプト化されたアップロード・パブリケーションのテーブルに対してストアド・プロシージャを定義しないと、テーブルには操作はアップロードされません。ALTER PUBLICATION を使用して、通常のパブリケーションをスクリプト化されたアップロード・パブリケーションに変更することはできません。

例

次のパブリケーションはストアド・プロシージャを使用して、t1 と t2 という 2 つのテーブルのデータをアップロードします。テーブル t1 には、挿入、削除、更新がアップロードされます。テーブル t2 には、挿入のみがアップロードされます。

```
CREATE PUBLICATION pub WITH SCRIPTED UPLOAD (
    TABLE t1 (col1, col2, col3) USING (
      PROCEDURE my.t1_ui FOR UPLOAD INSERT,
      PROCEDURE my.t1_ud FOR UPLOAD DELETE,
      PROCEDURE my.t1_uu FOR UPLOAD UPDATE
    ),
    TABLE t2 USING (
      PROCEDURE my.t2_ui FOR UPLOAD INSERT
    )
  )
```

参照

- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「ALTER PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』

スクリプト化されたアップロードの例

この例は、競合を検出するスクリプト化されたアップロードの設定方法を示しています。例では、スクリプト化されたアップロードで必要な統合データベースとリモート・データベース、ストアド・プロシージャ、パブリケーション、サブスクリプションを作成します。この例は、参考にするだけでもかまいませんし、テキストをコピー・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。

統合データベースの作成

サンプル・ファイルを保持するディレクトリを作成します。ここでは、名前を *scripteduploa*d とします。コマンド・プロンプトを開き、そのディレクトリに移動します。

(この例では、ファイル名を指定し、ファイルが現在のディレクトリにあるものと想定しています。実際のアプリケーションでは、ファイルのフル・パスを指定してください。)

次のコマンドを実行して、統合データベースを作成します。

```
dbinit consol.db
```

次のコマンドを実行して、統合データベースの ODBC データ・ソースを定義します。

```
dbdsn -w dsn_consol -y -c "uid=DBA;pwd=sql;dbf=consol.db;eng=consol"
```

データベースを Mobile Link 統合データベースとして使用するには、Mobile Link で使用するシステム・テーブル、ビュー、ストアド・プロシージャを追加する設定スクリプトを実行する必要があります。次のコマンドを実行し、統合データベースとして consol.db を設定します。

```
dbisql -c "dsn=dsn_consol" %sqlany11%¥MobiLink¥setup¥syncsa.sql
```

Interactive SQL を開き、dsn_consol DSN を使用して consol.db に接続します。次の SQL 文を実行します。実行すると、統合データベースで employee テーブルが作成され、値をテーブルが挿入され、必要な同期スクリプトが作成されます。

```
CREATE TABLE employee (
  id      unsigned integer primary key,
  name    varchar( 256),
  salary  numeric( 9, 2 )
);

INSERT INTO employee VALUES( 100, 'smith', 225000 );
COMMIT;

CALL ml_add_table_script( 'default', 'employee', 'upload_insert',
     'INSERT INTO employee ( id, name, salary ) VALUES ( ?, ?, ? )' );

CALL ml_add_table_script( 'default', 'employee', 'upload_update',
     'UPDATE employee SET name = ?, salary = ? WHERE id = ?' );

CALL ml_add_table_script( 'default', 'employee', 'upload_delete',
     'DELETE FROM employee WHERE id = ?' );

CALL ml_add_table_script( 'default', 'employee', 'download_cursor',
     'SELECT * from employee' );
```

リモート・データベースの作成

サンプル・ディレクトリのコマンド・プロンプトで、次のコマンドを実行して、リモート・データベースを作成します。

```
dbinit remote.db
```

次のコマンドを実行して、ODBC データ・ソースを定義します。

```
dbdsn -w dsn_remote -y -c "uid=dba;pwd=sql;dbf=remote.db;eng=remote"
```

Interactive SQL で、dsn_remote DSN を使用して remote.db に接続します。次の文のセットを実行して、リモート・データベースでオブジェクトを作成します。

まず、同期させるテーブルを作成します。insert_time と delete_time カラムは同期されませんが、アップロードするローを指定するためにアップロード・ストアド・プロシージャで使用される情報が含まれます。

```
CREATE TABLE employee (
    id          unsigned integer primary key,
    name        varchar( 256),
    salary      numeric( 9, 2 ),
    insert_time   timestamp default '1900-01-01'
);
```

次に、ストアド・プロシージャと、アップロードを処理するその他の処理を定義する必要があります。更新、挿入、削除ごとに別々に定義します。

挿入の処理

まず、ローを挿入するときに、各ローに insert_time を設定するトリガを作成します。このタイムスタンプは、最後の同期以降にローが挿入されたかどうかを調べるのに使用されます。統合データベースからダウンロードされた挿入を dbmlsync が適用するときは、このトリガは起動しません。これは、この例の後半で、FireTriggers 拡張オプションがオフに設定されるからです。ダウンロードによって挿入されたローは、employee テーブルが作成されたときに定義されたデフォルト値 1900-01-01 を insert_time として取得します。ローが新しい挿入として処理され、次の同期中にアップロードされるのを防ぐために、この値は、開始進行状況値よりも前に設定する必要があります。

```
CREATE TRIGGER emp_ins AFTER INSERT ON employee
REFERENCING NEW AS newrow
FOR EACH ROW
BEGIN
   UPDATE employee SET insert_time = CURRENT TIMESTAMP
   WHERE id = newrow.id
END;
```

次に、アップロード用に挿入されたすべてのローを結果セットとして返すプロシージャを作成します。このプロシージャは、最後に正常にアップロードされてから (insert_time に基づいて) 挿入された後、続いて削除されなかったたすべてのローを返します。最後の正常なアップロードの時間は、#hook_dict テーブルの開始進行状況値から判別します。この例では、dbmlsync 拡張オプション LockTables にデフォルト設定を使用して、同期対象のテーブルをロックします。したがって、終了進行状況後に挿入されたローを除外する必要がありません。テーブルのロックによって、アップロードの構築中に、操作が終了進行状況後に発生することが防止されます。

```
CREATE PROCEDURE employee_insert()
RESULT( id  unsigned integer,
```



```
        name varchar( 256 ),
        salary numeric( 9,2 )
    )
BEGIN
    DECLARE start_time timestamp;
    SELECT value
    INTO start_time
    FROM #hook_dict
    WHERE name = 'start progress as timestamp';

    // Upload as inserts all rows inserted after the start_time
    // that were not subsequently deleted
    SELECT id, name, salary
    FROM employee e
    WHERE insert_time > start_time AND
      NOT EXISTS( SELECT id FROM employee_delete ed  WHERE ed.id = e.id );

END;
```

更新の処理

アップロードを処理するには、アップロード構築中、開始進行状況値に基づいて正しい更新前イメージを使用する必要があります。

まず、更新されたローの更新前イメージを保持するテーブルを作成します。スクリプト化されたアップロードを生成するときには、更新前イメージが使用されます。

```
CREATE TABLE employee_preimages (
  id          unsigned integer NOT NULL,
  name        varchar( 256),
  salary      numeric( 9, 2 ),
  img_time    timestamp default CURRENT TIMESTAMP,
  primary key( id, img_time )
);
```

次に、各ローが更新されるときに、更新前イメージを保存するトリガを作成します。挿入トリガと同様、このトリガもダウンロードでは起動しません。

このトリガは、ローが更新されるたびに更新前イメージを保存します (ただし、2 つの更新が非常に近く、タイムスタンプが同じ場合を除く)。これは一見、無駄に見えるので、ローの更新前イメージがテーブルにない場合のみ保存し、sp_hook_dbmlsync_upload_end フックに依存して、更新前イメージがアップロードされた後に削除しがちです。

しかし、sp_hook_dbmlsync_upload_end フックは、この目的では確実ではありません。アップロードを送信した後で受信確認される前に、ハードウェアまたはソフトウェアの障害により dbmlsync が停止した場合、フックが呼び出されない可能性があります。その結果、ローが正常にアップロードされても、ローは更新前イメージ テーブルから削除されません。また、通信障害が発生すると、dbmlsync はサーバからアップロードの受信確認を受信できない場合があります。この場合、フックに渡されるアップロード・ステータスは「不明」になります。このステータスでは、フックは、更新前イメージ・テーブルがクリーンアップされたのかそのままなのかを判別できません。複数の更新前イメージを保存すると、アップロードが構築されるときに、開始進行状況値に基づいて正しい更新前イメージが常に選択されます。

```
CREATE TRIGGER emp_upd AFTER UPDATE OF name,salary ON employee
  REFERENCING OLD AS oldrow
  FOR EACH ROW
BEGIN
  INSERT INTO employee_preimages ON EXISTING SKIP VALUES(
```

```
    oldrow.id, oldrow.name, oldrow.salary, CURRENT TIMESTAMP );
END;
```

次に、更新を処理するアップロード・プロシージャを作成します。このストアド・プロシージャは、他のスクリプトの 2 倍のカラムを持つ結果セットを 1 つ返します。これには、更新前イメージ (Mobile Link サーバから最後に受信したローと、正常にアップロードされたローの値) と更新後イメージ (統合データベースに入力される値) が含まれます。

更新前イメージは、start_progress 後に記録された employee_preimages の一番最初の値セットです。この例では、削除されてから再度挿入された既存のローは、正しく処理されません。より完全なソリューションでは、これらのローは更新としてアップロードされます。

```
CREATE PROCEDURE employee_update()
RESULT(
      preimage_id  unsigned integer,
      preimage_name varchar( 256),
      preimage_salary numeric( 9,2 ),
      postimage_id  unsigned integer,
      postimage_name varchar( 256),
      postimage_salary numeric( 9,2 )
    )
BEGIN
   DECLARE start_time timestamp;

   SELECT value
   INTO start_time
   FROM #hook_dict
   WHERE name = 'start progress as timestamp';

   // Upload as an update all rows that have been updated since
   // start_time that were not newly inserted or deleted.
   SELECT ep.id, ep.name, ep.salary, e.id, e.name, e.salary
   FROM employee e JOIN employee_preimages ep
      ON ( e.id = ep.id )
   // Do not select rows inserted since the start time. These should be
   // uploaded as inserts.
   WHERE insert_time <= start_time
     // Do not upload deleted rows.
     AND NOT EXISTS( SELECT id FROM employee_delete ed  WHERE ed.id = e.id )
     // Select the earliest pre-image after the start time.
     AND ep.img_time = ( SELECT MIN( img_time )
          FROM employee_preimages
          WHERE id = ep.id
          AND img_time > start_time );
END;
```

削除の処理

まず、削除されたローのリストを維持するテーブルを作成します。

```
CREATE TABLE employee_delete (
   id          unsigned integer  primary key NOT NULL,
   name        varchar( 256 ),
   salary      numeric( 9, 2 ),
   delete_time  timestamp
);
```

次に、employee テーブルからローが削除されるときに employee_delete テーブルを移植するトリガを作成します。後で dbmlsync 拡張オプション FireTriggers を false に設定するので、このトリ

ガは、ダウンロード中は呼び出されません。このトリガは、削除されたローは再度挿入されることはないことを前提としています。したがって、複数回削除されるローは処理されません。

```
CREATE TRIGGER emp_del AFTER DELETE ON employee
REFERENCING OLD AS delrow
FOR EACH ROW
BEGIN
    INSERT INTO employee_delete
VALUES( delrow.id, delrow.name, delrow.salary, CURRENT TIMESTAMP );
END;
```

次の SQL 文は、削除を処理するアップロード・プロシージャを作成します。ストアド・プロシージャは、統合データベースで削除するローが含まれる結果セットを返します。ストアド・プロシージャは employee_preimages テーブルを使用するので、ローが更新された後に削除されると、削除用にアップロードされるイメージは、最後に正常にダウンロードまたはアップロードされたイメージになります。

```
CREATE PROCEDURE employee_delete()
RESULT( id  unsigned integer,
        name varchar( 256),
        salary numeric( 9,2 )
      )
BEGIN
   DECLARE start_time timestamp;

   SELECT value
   INTO start_time
   FROM #hook_dict
   WHERE name = 'start progress as timestamp';

  // Upload as a delete all rows that were deleted after the
  // start_time that were not inserted after the start_time.
  // If a row was updated before it was deleted, then the row
  // to be deleted is the pre-image of the update.
   SELECT IF ep.id IS NULL THEN ed.id ELSE ep.id ENDIF,
         IF ep.id IS NULL THEN ed.name ELSE ep.name ENDIF,
         IF ep.id IS NULL THEN ed.salary ELSE ep.salary ENDIF
   FROM employee_delete ed LEFT OUTER JOIN employee_preimages ep
        ON( ed.id = ep.id AND ep.img_time > start_time )
   WHERE
     // Only upload deletes that occurred since the last sync.
     ed.delete_time > start_time
     // Don't upload a delete for rows that were inserted since
     // the last upload and then deleted.
   AND NOT EXISTS (
     SELECT id
        FROM employee e
        WHERE e.id = ep.id AND e.insert_time > start_time )
   // Select the earliest preimage after the start time.
   AND ( ep.id IS NULL OR ep.img_time = (SELECT MIN( img_time )
                                FROM employee_preimages
                                WHERE id = ep.id
                                 AND img_time > start_time ) );
END;
```

更新前イメージ・テーブルのクリーンアップ

次に、アップロードが成功したときに employee_preimage と employee_delete テーブルをクリーンアップする upload_end フックを作成します。この例では、同期中にテーブルがロックされるように、dbmlsync 拡張オプション LockTables にデフォルト設定を使用します。したがって、テー

ブルのローに対して、end_progress 後に発生した操作が実行される心配がありません。ロックにより、このような操作が発生するのを防ぐことができます。

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_end()
BEGIN
   DECLARE val   varchar(256);

   SELECT value
   INTO val
   FROM #hook_dict
   WHERE name = 'upload status';

   IF val = 'committed' THEN
     DELETE FROM employee_delete;
     DELETE FROM employee_preimages;
   END IF;
END;
```

パブリケーション、Mobile Link ユーザ、サブスクリプションの作成

pub1 と呼ばれるパブリケーションでは、スクリプト化されたアップロードの構文 (WITH SCRIPTED UPLOAD) が使用されます。このパブリケーションによって、employee テーブルのアーティクルが作成され、スクリプト化されたアップロード用に作成したばかりの 3 つのストアド・プロシージャが登録されます。u1 という Mobile Link ユーザと、v1 と pub1 の間のサブスクリプションが作成されます。拡張オプション FireTriggers はオフに設定されるので、ダウンロードが適用されているときはリモート・データベースでトリガが起動されません。これにより、次の同期中に、ダウンロードされた変更がアップロードされるのを防ぐことができます。

```
CREATE PUBLICATION pub1 WITH SCRIPTED UPLOAD (
TABLE employee( id, name, salary ) USING (
  PROCEDURE employee_insert FOR UPLOAD INSERT,
  PROCEDURE employee_update FOR UPLOAD UPDATE,
  PROCEDURE employee_delete FOR UPLOAD DELETE,
    )
)

CREATE SYNCHRONIZATION USER u1;

CREATE SYNCHRONIZATION SUBSCRIPTION TO pub1 FOR u1
TYPE 'tcpip'
ADDRESS 'host=localhost'
OPTION FireTriggers='off';
```

スクリプト化されたアップロードの実行

リモート・データベースに接続し、スクリプト化されたアップロードを使用して、同期するデータを挿入します。たとえば、Interactive SQL のリモート・データベースに対して次の SQL 文を実行します。

```
INSERT INTO employee(id, name, salary) VALUES( 7, 'black', 700 );
INSERT INTO employee(id, name, salary) VALUES( 8, 'anderson', 800 );
INSERT INTO employee(id, name, salary) VALUES( 9, 'dilon', 900 );
INSERT INTO employee(id, name, salary) VALUES( 10, 'dwit', 1000 );
INSERT INTO employee(id, name, salary) VALUES( 11, 'dwit', 1100 );
COMMIT;
```

コマンド・プロンプトで、Mobile Link サーバを起動します。

```
mlsrv11 -c "dsn=dsn_consol" -o mlserver.mls -v+ -dl -zu+
```

dbmlsync を使用して同期を開始します。

```
dbmlsync -c "dsn=dsn_remote" -k -uo -o remote.mlc -v+
```

これで、挿入がアップロードされたことを確認できます。

例のクリーンアップ

例を完了後コンピュータをクリーンアップするには、次の手順を実行します。

```
mlstop -h -w
dbstop -y -c eng=consol
dbstop -y -c eng=remote

dberase -y consol.db
dberase -y remote.db

del remote.mlc mlserver.mls
```


用語解説

用語解説

Adaptive Server Anywhere (ASA)

SQL Anywhere Studio のリレーショナル・データベース・サーバ・コンポーネントであり、主に、モバイル環境と埋め込み環境、または小規模および中規模のビジネス用のサーバとして使用されます。バージョン 10.0.0 で、Adaptive Server Anywhere は SQL Anywhere サーバに、SQL Anywhere Studio は SQL Anywhere にそれぞれ名前が変更されました。

参照：「SQL Anywhere」 416 ページ。

Carrier

Mobile Link システム・テーブルまたは Notifier プロパティ・ファイルに保存される Mobile Link オブジェクトで、システム起動同期で使用される通信業者に関する情報が含まれます。

参照：「サーバ起動同期」 421 ページ。

DB 領域

データ用の領域をさらに作成する追加のデータベース・ファイルです。1 つのデータベースは 13 個までのファイルに保管されます (初期ファイル 1 つと 12 の DB 領域)。各テーブルは、そのインデックスとともに、単一のデータベース・ファイルに含まれている必要があります。CREATE DBSPACE という SQL コマンドで、新しいファイルをデータベースに追加できます。

参照：「データベース・ファイル」 425 ページ。

DBA 権限

ユーザに、データベース内の管理作業を許可するレベルのパーミッションです。DBA ユーザにはデフォルトで DBA 権限が与えられています。

参照：「データベース管理者 (DBA)」 425 ページ。

EBF

Express Bug Fix の略です。Express Bug Fix は、1 つ以上のバグ・フィックスが含まれる、ソフトウェアのサブセットです。これらのバグ・フィックスは、更新のリリース・ノートにリストされます。バグ・フィックス更新を適用できるのは、同じバージョン番号を持つインストール済みのソフトウェアに対してだけです。このソフトウェアについては、ある程度のテストが行われているとはいえ、完全なテストが行われたわけではありません。自分自身でソフトウェアの妥当性を確かめるまでは、アプリケーションとともにこれらのファイルを配布しないでください。

Embedded SQL

C プログラム用のプログラミング・インタフェースです。SQL Anywhere の Embedded SQL は ANSI と IBM 規格に準拠して実装されています。

FILE

SQL Remote のレプリケーションでは、レプリケーション・メッセージのやりとりのために共有ファイルを使うメッセージ・システムのことです。これは特定のメッセージ送信システムに頼らずにテストやインストールを行うのに便利です。

参照：「レプリケーション」 433 ページ。

grant オプション

他のユーザにパーミッションを許可できるレベルのパーミッションです。

iAnywhere JDBC ドライバ

iAnywhere JDBC ドライバでは、pure Java である jConnect JDBC ドライバに比べて何らかの有利なパフォーマンスや機能を備えた JDBC ドライバが提供されます。ただし、このドライバは pure Java ソリューションではありません。iAnywhere JDBC ドライバは一般に推奨されるドライバです。

参照：

- 「JDBC」 413 ページ
- 「jConnect」 413 ページ

InfoMaker

レポート作成とデータ管理用のツールです。洗練されたフォーム、レポート、グラフ、クロスタブ、テーブルを作成できます。また、これらを基本的な構成要素とするアプリケーションも作成できます。

Interactive SQL

データベース内のデータの変更や問い合わせ、データベース構造の修正ができる、SQL Anywhere のアプリケーションです。Interactive SQL では、SQL 文を入力するためのウィンドウ枠が表示されます。また、クエリの進捗情報や結果セットを返すウィンドウ枠も表示されます。

JAR ファイル

Java アーカイブ・ファイルです。Java のアプリケーションで使用される 1 つ以上のパッケージの集合からなる圧縮ファイルのフォーマットです。Java プログラムをインストールしたり実行したりするのに必要なリソースが 1 つの圧縮ファイルにすべて収められています。

Java クラス

Java のコードの主要な構造単位です。これはプロシージャや変数の集まりで、すべてがある一定のカテゴリに関連しているためグループ化されたものです。

jConnect

JavaSoft JDBC 標準を Java で実装したものです。これにより、Java 開発者は多層／異機種環境でもネイティブなデータベース・アクセスができます。iAnywhere JDBC ドライバは一般に推奨されるドライバです。

参照：

- 「JDBC」 413 ページ
- 「iAnywhere JDBC ドライバ」 412 ページ

JDBC

Java Database Connectivity の略です。Java アプリケーションからリレーショナル・データにアクセスすることを可能にする SQL 言語プログラミング・インタフェースです。推奨 JDBC ドライバは、iAnywhere JDBC ドライバです。

参照：

- 「jConnect」 413 ページ
- 「iAnywhere JDBC ドライバ」 412 ページ

Listener

Mobile Link サーバ起動同期に使用される、dblsn という名前のプログラムです。Listener はリモート・デバイスにインストールされ、Push 通知を受け取ったときにデバイス上でアクションが開始されるように設定されます。

参照：「サーバ起動同期」 421 ページ。

LTM

LTM (Log Transfer Manager) は、Replication Agent とも呼ばれます。Replication Server と併用することで、LTM はデータベース・トランザクション・ログを読み込み、コミットされた変更を Sybase Replication Server に送信します。

参照：「Replication Server」 416 ページ。

Mobile Link

Ultra Light と SQL Anywhere のリモート・データベースを統合データベースと同期させるために設計された、セッションベース同期テクノロジです。

参照：

- 「統合データベース」 440 ページ
- 「同期」 440 ページ
- 「Ultra Light」 417 ページ

Mobile Link クライアント

2 種類の Mobile Link クライアントがあります。SQL Anywhere リモート・データベース用の Mobile Link クライアントは、dbmlsync コマンド・ライン・ユーティリティです。Ultra Light リモート・データベース用の Mobile Link クライアントは、Ultra Light ランタイム・ライブラリに組み込まれています。

Mobile Link サーバ

Mobile Link 同期を実行する、mlsrv11 という名前のコンピュータ・プログラムです。

Mobile Link システム・テーブル

Mobile Link の同期に必要なシステム・テーブルです。Mobile Link 設定スクリプトによって、Mobile Link 統合データベースにインストールされます。

Mobile Link モニタ

Mobile Link の同期をモニタするためのグラフィカル・ツールです。

Mobile Link ユーザ

Mobile Link ユーザは、Mobile Link サーバに接続するのに使用されます。Mobile Link ユーザをリモート・データベースに作成し、統合データベースに登録します。Mobile Link ユーザ名はデータベース・ユーザ名から完全に独立しています。

Notifier

Mobile Link サーバ起動同期に使用されるプログラムです。Notifier は Mobile Link サーバに統合されており、統合データベースに Push 要求がないか確認し、Push 通知を送信します。

参照：

- 「サーバ起動同期」 421 ページ
- 「Listener」 413 ページ

ODBC

Open Database Connectivity の略です。データベース管理システムに対する Windows の標準的なインタフェースです。ODBC は、SQL Anywhere がサポートするインタフェースの 1 つです。

ODBC アドミニストレータ

Windows オペレーティング・システムに付属している Microsoft のプログラムです。ODBC データ・ソースの設定に使用します。

ODBC データ・ソース

ユーザが ODBC からアクセスするデータと、そのデータにアクセスするために必要な情報の仕様です。

PDB

Palm のデータベース・ファイルです。

PowerDesigner

データベース・モデリング・アプリケーションです。これを使用すると、データベースやデータ・ウェアハウスの設計に対する構造的なアプローチが可能となります。SQL Anywhere には、PowerDesigner の Physical Data Model コンポーネントが付属します。

PowerJ

Java アプリケーション開発に使用する Sybase 製品です。

Push 通知

QAnywhere では、メッセージ転送を開始するよう QAnywhere クライアントに対して指示するために、サーバから QAnywhere クライアントに配信される特殊なメッセージです。Mobile Link サーバ起動同期では、Push 要求データや内部情報を含むデバイスに Notifer から配信される特殊なメッセージです。

参照：

- 「QAnywhere」 415 ページ
- 「サーバ起動同期」 421 ページ

Push 要求

Mobile Link サーバ起動同期において、Push 通知をデバイスに送信する必要があるかどうかを判断するために Notifier が確認する、結果セット内の値のローです。

参照：「サーバ起動同期」 421 ページ。

QAnywhere

アプリケーション間メッセージング (モバイル・デバイス間メッセージングやモバイル・デバイスとエンタープライズの間のメッセージングなど) を使用すると、モバイル・デバイスや無線デバイスで動作しているカスタム・プログラムと、集中管理されているサーバ・アプリケーションとの間で通信できます。

QAnywhere Agent

QAnywhere では、クライアント・デバイス上で動作する独立のプロセスのことです。クライアント・メッセージ・ストアをモニタリングし、メッセージを転送するタイミングを決定します。

REMOTE DBA 権限

SQL Remote では、Message Agent (dbremote) で必要なパーミッションのレベルを指します。Mobile Link では、SQL Anywhere 同期クライアント (dbmlsync) で必要なパーミッションのレベルを指します。Message Agent (dbremote) または同期クライアントがこの権限のあるユーザとして接続した場合、DBA のフル・アクセス権が与えられます。Message Agent (dbremote) または同期クライアント (dbmlsync) から接続しない場合、このユーザ ID にはパーミッションは追加されません。

参照：「DBA 権限」 411 ページ。

Replication Agent

参照：「LTM」 413 ページ。

Replication Server

SQL Anywhere と Adaptive Server Enterprise で動作する、Sybase による接続ベースのレプリケーション・テクノロジです。Replication Server は、少数のデータベース間でほぼリアルタイムのレプリケーションを行うことを目的に設計されています。

参照：「LTM」 413 ページ。

SQL

リレーショナル・データベースとの通信に使用される言語です。SQL は ANSI により標準が定義されており、その最新版は SQL-2003 です。SQL は、公認されてはいませんが、Structured Query Language の略です。

SQL Anywhere

SQLAnywhere のリレーショナル・データベース・サーバ・コンポーネントであり、主に、モバイル環境と埋め込み環境、または小規模および中規模のビジネス用のサーバとして使用されます。SQL Anywhere は、SQL Anywhere RDBMS、Ultra Light RDBMS、Mobile Link 同期ソフトウェア、その他のコンポーネントを含むパッケージの名前でもあります。

SQL Remote

統合データベースとリモート・データベース間で双方向レプリケーションを行うための、メッセージベースのデータ・レプリケーション・テクノロジです。統合データベースとリモート・データベースは、SQL Anywhere である必要があります。

SQL ベースの同期

Mobile Link では、Mobile Link イベントを使用して、テーブル・データを Mobile Link でサポートされている統合データベースに同期する方法のことです。SQL ベースの同期では、SQL を直接使用したり、Java と .NET 用の Mobile Link サーバ API を使用して SQL を返すことができます。

SQL 文

DBMS に命令を渡すために設計された、SQL キーワードを含む文字列です。

参照：

- 「スキーマ」 423 ページ
- 「SQL」 416 ページ
- 「データベース管理システム (DBMS)」 425 ページ

Sybase Central

SQL Anywhere データベースのさまざまな設定、プロパティ、ユーティリティを使用できる、グラフィカル・ユーザ・インタフェースを持つデータベース管理ツールです。Mobile Link などの他の iAnywhere 製品を管理する場合にも使用できます。

SYS

システム・オブジェクトの大半を所有する特別なユーザです。一般のユーザは SYS でログインできません。

Ultra Light

小型デバイス、モバイル・デバイス、埋め込みデバイス用に最適化されたデータベースです。対象となるプラットフォームとして、携帯電話、ポケットベル、パーソナル・オーガナイザなどが挙げられます。

Ultra Light ランタイム

組み込みの Mobile Link 同期クライアントを含む、インプロセス・リレーショナル・データベース管理システムです。Ultra Light ランタイムは、Ultra Light の各プログラミング・インタフェースで使用されるライブラリと、Ultra Light エンジンの両方に含まれます。

Windows

Windows Vista、Windows XP、Windows 200x などの、Microsoft Windows オペレーティング・システムのファミリのことです。

Windows CE

「Windows Mobile」 417 ページを参照してください。

Windows Mobile

Microsoft がモバイル・デバイス用に開発したオペレーティング・システムのファミリです。

アーティクル

Mobile Link または SQL Remote では、テーブル全体もしくはテーブル内のカラムとローのサブセットを表すデータベース・オブジェクトを指します。アーティクルの集合がパブリケーションです。

参照：

- 「レプリケーション」 433 ページ
- 「パブリケーション」 428 ページ

アップロード

同期中に、リモート・データベースから統合データベースにデータが転送される段階です。

アトミックなトランザクション

完全に処理されるかまったく処理されないことが保証される 1 つのトランザクションです。エラーによってアトミックなトランザクションの一部が処理されなかった場合は、データベースが一貫性のない状態になるのを防ぐために、トランザクションがロールバックされます。

アンロード

データベースをアンロードすると、データベースの構造かデータ、またはその両方がテキスト・ファイルにエクスポートされます (構造は SQL コマンド・ファイルに、データはカンマ区切りの ASCII ファイルにエクスポートされます)。データベースのアンロードには、アンロード・ユーティリティを使用します。

また、UNLOAD 文を使って、データから抜粋した部分だけをアンロードできます。

イベント・モデル

Mobile Link では、同期を構成する、begin_synchronization や download_cursor などの一連のイベントのことです。イベントは、スクリプトがイベント用に作成されると呼び出されます。

インクリメンタル・バックアップ

トランザクション・ログ専用のバックアップです。通常、フル・バックアップとフル・バックアップの間に使用します。

参照：「トランザクション・ログ」 427 ページ。

インデックス

ベース・テーブルにある 1 つ以上のカラムに関連付けられた、キーとポインタのソートされたセットです。テーブルの 1 つ以上のカラムにインデックスが設定されていると、パフォーマンスが向上します。

ウィンドウ

分析関数の実行対象となるローのグループです。ウィンドウには、ウィンドウ定義内のグループ化指定に従って分割されたデータの、1 つ、複数、またはすべてのローが含まれます。ウィンドウは、入力の現在のローについて計算を実行する必要があるローの数や範囲を含むように移動します。ウィンドウ構成の主な利点は、追加のクエリを実行しなくても、結果をグループ化して分析する機会が増えることです。

エージェント ID

参照：「クライアント・メッセージ・ストア ID」 420 ページ。

エンコード

文字コードとも呼ばれます。エンコードは、文字セットの各文字が情報の 1 つまたは複数のバイトにマッピングされる方法のことで、一般的に 16 進数で表現されます。UTF-8 はエンコードの例です。

参照：

- 「文字セット」 441 ページ
- 「コード・ページ」 421 ページ
- 「照合」 438 ページ

オブジェクト・ツリー

Sybase Central では、データベース・オブジェクトの階層を指します。オブジェクト・ツリーの最上位には、現在使用しているバージョンの Sybase Central がサポートするすべての製品が表示されます。それぞれの製品を拡張表示すると、オブジェクトの下位ツリーが表示されます。

参照：「Sybase Central」 417 ページ。

カーソル

結果セットへの関連付けに名前を付けたもので、プログラミング・インタフェースからローにアクセスしたり更新したりするときに使用します。SQL Anywhere では、カーソルはクエリ結果内で前方や後方への移動をサポートします。カーソルは、カーソル結果セット (通常 SELECT 文で定義される) とカーソル位置の 2 つの部分から構成されます。

参照：

- 「カーソル結果セット」 420 ページ
- 「カーソル位置」 419 ページ

カーソル位置

カーソル結果セット内の 1 つのローを指すポインタ。

参照：

- 「カーソル」 419 ページ
- 「カーソル結果セット」 420 ページ

カーソル結果セット

カーソルに関連付けられたクエリから生成されるローのセットです。

参照：

- 「カーソル」 419 ページ
- 「カーソル位置」 419 ページ

クエリ

データベースのデータにアクセスしたり、そのデータを操作したりする SQL 文や SQL 文のグループです。

参照：「SQL」 416 ページ。

クライアント／サーバ

あるアプリケーション (クライアント) が別のアプリケーション (サーバ) に対して情報を送受信するソフトウェア・アーキテクチャのことです。通常この 2 種類のアプリケーションは、ネットワークに接続された異なるコンピュータ上で実行されます。

クライアント・メッセージ・ストア

QAnywhere では、メッセージを保管するリモート・デバイスにある SQL Anywhere データベースのことです。

クライアント・メッセージ・ストア ID

QAnywhere では、Mobile Link リモート ID のことです。これによって、クライアント・メッセージ・ストアがユニークに識別されます。

グローバル・テンポラリ・テーブル

明示的に削除されるまでデータ定義がすべてのユーザに表示されるテンポラリ・テーブルです。グローバル・テンポラリ・テーブルを使用すると、各ユーザが、1 つのテーブルのまったく同じインスタンスを開くことができます。デフォルトでは、コミット時にローが削除され、接続終了時にもローが削除されます。

参照：

- 「テンポラリ・テーブル」 426 ページ
- 「ローカル・テンポラリ・テーブル」 434 ページ

ゲートウェイ

Mobile Link システム・テーブルまたは Notifier プロパティ・ファイルに保存される Mobile Link オブジェクトで、システム起動同期用のメッセージの送信方法に関する情報が含まれます。

参照：「サーバ起動同期」 421 ページ。

コード・ページ

コード・ページは、文字セットの文字を数値表示 (通常 0 ～ 255 の整数) にマッピングするエンコードです。Windows Code Page 1252 などのコード・ページがあります。このマニュアルの目的上、コード・ページとエンコードは同じ意味で使用されます。

参照：

- 「文字セット」 441 ページ
- 「エンコード」 419 ページ
- 「照合」 438 ページ

コマンド・ファイル

SQL 文で構成されたテキスト・ファイルです。コマンド・ファイルは手動で作成できますが、データベース・ユーティリティによって自動的に作成することもできます。たとえば、dbunload ユーティリティを使うと、指定されたデータベースの再構築に必要な SQL 文で構成されたコマンド・ファイルを作成できます。

サーバ・メッセージ・ストア

QAnywhere では、サーバ上のリレーショナル・データベースです。このデータベースは、メッセージを、クライアント・メッセージ・ストアまたは JMS システムに転送されるまで一時的に格納します。メッセージは、サーバ・メッセージ・ストアを介して、クライアント間で交換されます。

サーバ管理要求

XML 形式の QAnywhere メッセージです。サーバ・メッセージ・ストアを管理したり、QAnywhere アプリケーションをモニタリングするために QAnywhere システム・キューに送信されます。

サーバ起動同期

Mobile Link サーバから Mobile Link 同期を開始する方法です。

サービス

Windows オペレーティング・システムで、アプリケーションを実行するユーザ ID がログオンしていないときにアプリケーションを実行する方法です。

サブクエリ

別の SELECT 文、INSERT 文、UPDATE 文、DELETE 文、または別のサブクエリの中にネストされた SELECT 文です。

相関とネストの 2 種類のサブクエリがあります。

サブスクリプション

Mobile Link 同期では、パブリケーションと Mobile Link ユーザ間のクライアント・データベース内のリンクであり、そのパブリケーションが記述したデータの同期を可能にします。

SQL Remote レプリケーションでは、パブリケーションとリモート・ユーザ間のリンクのことで、これによりリモート・ユーザはそのパブリケーションの更新内容を統合データベースとの間で交換できます。

参照：

- 「パブリケーション」 428 ページ
- 「Mobile Link ユーザ」 414 ページ

システム・オブジェクト

SYS または dbo が所有するデータベース・オブジェクトです。

システム・テーブル

SYS または dbo が所有するテーブルです。メタデータが格納されています。システム・テーブル (データ辞書テーブルとしても知られています) はデータベース・サーバが作成し管理します。

システム・ビュー

すべてのデータベースに含まれているビューです。システム・テーブル内に格納されている情報をわかりやすいフォーマットで示します。

ジョイン

指定されたカラムの値を比較することによって 2 つ以上のテーブルにあるローをリンクする、リレーショナル・システムでの基本的な操作です。

ジョイン・タイプ

SQL Anywhere では、クロス・ジョイン、キー・ジョイン、ナチュラル・ジョイン、ON 句を使ったジョインの 4 種類のジョインが使用されます。

参照：「ジョイン」 422 ページ。

ジョイン条件

ジョインの結果に影響を及ぼす制限です。ジョイン条件は、JOIN の直後に ON 句か WHERE 句を挿入して指定します。ナチュラル・ジョインとキー・ジョインについては、SQL Anywhere がジョイン条件を生成します。

参照：

- 「ジョイン」 422 ページ
- 「生成されたジョイン条件」 439 ページ

スキーマ

テーブル、カラム、インデックス、それらの関係などを含んだデータベース構造です。

スクリプト

Mobile Link では、Mobile Link のイベントを処理するために記述されたコードです。スクリプトは、業務上の要求に適合するように、データ交換をプログラム的に制御します。

参照：「イベント・モデル」 418 ページ。

スクリプト・バージョン

Mobile Link では、同期を作成するために同時に適用される、一連の同期スクリプトです。

スクリプトベースのアップロード

Mobile Link では、ログ・ファイルを使用した方法の代わりとなる、アップロード処理のカスタマイズ方法です。

ストアド・プロシージャ

ストアド・プロシージャは、データベースに保存され、データベース・サーバに対する一連の操作やクエリを実行するために使用される SQL 命令のグループです。

スナップショット・アイソレーション

読み込み要求を発行するトランザクション用のデータのコミットされたバージョンを返す、独立性レベルの種類です。SQL Anywhere では、スナップショット、文のスナップショット、読み込み専用文のスナップショットの 3 つのスナップショットの独立性レベルがあります。スナップショット・アイソレーションが使用されている場合、読み込み処理は書き込み処理をブロックしません。

参照：「独立性レベル」 441 ページ。

セキュア機能

データベース・サーバが起動されたときに、そのデータベース・サーバで実行されているデータベースでは使用できないように -sf オプションによって指定される機能です。

セッション・ベースの同期

統合データベースとリモート・データベースの両方でデータ表現の一貫性が保たれる同期です。Mobile Link はセッション・ベースです。

ダイレクト・ロー・ハンドリング

Mobile Link では、テーブル・データを Mobile Link でサポートされている統合データベース以外のソースに同期する方法のことです。アップロードとダウンロードの両方をダイレクト・ロー・ハンドリングで実装できます。

参照：

- 「統合データベース」 440 ページ
- 「SQL ベースの同期」 417 ページ

ダウンロード

同期中に、統合データベースからリモート・データベースにデータが転送される段階です。

チェックサム

データベース・ページを使用して記録されたデータベース・ページのビット数の合計です。チェックサムを使用すると、データベース管理システムは、ページがディスクに書き込まれるときに数が一致しているかを確認することで、ページの整合性を検証できます。数が一致した場合は、ページが正常に書き込まれたとみなされます。

チェックポイント

データベースに加えたすべての変更内容がデータベース・ファイルに保存されるポイントです。通常、コミットされた変更内容はトランザクション・ログだけに保存されます。

データ・キューブ

同じ結果を違う方法でグループ化およびソートされた内容を各次元に反映した、多次元の結果セットです。データ・キューブは、セルフジョイン・クエリと相関サブクエリを必要とするデータの複雑な情報を提供します。データ・キューブは OLAP 機能の一部です。

データベース

プライマリ・キーと外部キーによって関連付けられているテーブルの集合です。これらのテーブルでデータベース内の情報が保管されます。また、テーブルとキーによってデータベースの構造が定義されます。データベース管理システムでこの情報にアクセスします。

参照：

- 「外部キー」 435 ページ
- 「プライマリ・キー」 430 ページ
- 「データベース管理システム (DBMS)」 425 ページ
- 「リレーショナル・データベース管理システム (RDBMS)」 433 ページ

データベース・オブジェクト

情報を保管したり受け取ったりするデータベース・コンポーネントです。テーブル、インデックス、ビュー、プロシージャ、トリガはデータベース・オブジェクトです。

データベース・サーバ

データベース内にある情報へのすべてのアクセスを規制するコンピュータ・プログラムです。SQL Anywhere には、ネットワーク・サーバとパーソナル・サーバの 2 種類のサーバがあります。

データベース・ファイル

データベースは 1 つまたは複数のデータベース・ファイルに保持されます。まず、初期ファイルがあり、それに続くファイルは DB 領域と呼ばれます。各テーブルは、それに関連付けられているインデックスとともに、単一のデータベース・ファイルに含まれている必要があります。

参照：「DB 領域」 411 ページ。

データベース管理システム (DBMS)

データベースを作成したり使用したりするためのプログラムの集合です。

参照：「リレーショナル・データベース管理システム (RDBMS)」 433 ページ。

データベース管理者 (DBA)

データベースの管理に必要なパーミッションを持つユーザです。DBA は、データベース・スキーマのあらゆる変更や、ユーザやグループの管理に対して、全般的な責任を負います。データベース管理者のロールはデータベース内に自動的に作成されます。その場合、ユーザ ID は DBA であり、パスワードは sql です。

データベース所有者 (dbo)

SYS が所有しないシステム・オブジェクトを所有する特別なユーザです。

参照：

- 「データベース管理者 (DBA)」 425 ページ
- 「SYS」 417 ページ

データベース接続

クライアント・アプリケーションとデータベース間の通信チャネルです。接続を確立するためには有効なユーザ ID とパスワードが必要です。接続中に実行できるアクションは、そのユーザ ID に付与された権限によって決まります。

データベース名

サーバがデータベースをロードするとき、そのデータベースに指定する名前です。デフォルトのデータベース名は、初期データベース・ファイルのルート名です。

参照：「データベース・ファイル」 425 ページ。

データ型

CHAR や NUMERIC などのデータのフォーマットです。ANSI SQL 規格では、サイズ、文字セット、照合に関する制限もデータ型に組み込めます。

参照：「ドメイン」 426 ページ。

データ操作言語 (DML)

データベース内のデータの操作に使う SQL 文のサブセットです。DML 文は、データベース内のデータを検索、挿入、更新、削除します。

データ定義言語 (DDL)

データベース内のデータの構造を定義するときに使う SQL 文のサブセットです。DDL 文は、テーブルやユーザなどのデータベース・オブジェクトを作成、変更、削除できます。

デッドロック

先へ進めない場所に一連のトランザクションが到達する状態です。

デバイス・トラッキング

Mobile Link サーバ起動同期において、デバイスを特定する Mobile Link のユーザ名を使用して、メッセージのアドレスを指定できる機能です。

参照：「サーバ起動同期」 421 ページ。

テンポラリ・テーブル

データを一時的に保管するために作成されるテーブルです。グローバルとローカルの 2 種類があります。

参照：

- 「ローカル・テンポラリ・テーブル」 434 ページ
- 「グローバル・テンポラリ・テーブル」 420 ページ

ドメイン

適切な位置に精度や小数点以下の桁数を含み、さらにオプションとしてデフォルト値や CHECK 条件などを含んでいる、組み込みデータ型のエイリアスです。ドメインには、通貨データ型のように SQL Anywhere が事前に定義したものもあります。ユーザ定義データ型とも呼ばれます。

参照：「データ型」 426 ページ。

トランザクション

作業の論理単位を構成する一連の SQL 文です。1 つのトランザクションは完全に処理されるかまったく処理されないかのどちらかです。SQL Anywhere は、ロック機能のあるトランザクション処理をサポートしているので、複数のトランザクションが同時にデータベースにアクセスしてもデータを壊すことがありません。トランザクションは、データに加えた変更を永久なものにする COMMIT 文か、トランザクション中に加えられたすべての変更を元に戻す ROLLBACK 文のいずれかで終了します。

トランザクション・ログ

データベースに対するすべての変更内容が、変更された順に格納されるファイルです。パフォーマンスを向上させ、データベース・ファイルが破損した場合でもデータをリカバリできます。

トランザクション・ログ・ミラー

オプションで設定できる、トランザクション・ログ・ファイルの完全なコピーのことで、トランザクション・ログと同時に管理されます。データベースの変更がトランザクション・ログへ書き込まれると、トランザクション・ログ・ミラーにも同じ内容が書き込まれます。

ミラー・ファイルは、トランザクション・ログとは別のデバイスに置いてください。一方のデバイスに障害が発生しても、もう一方のログにリカバリのためのデータが確保されます。

参照：「トランザクション・ログ」 427 ページ。

トランザクション単位の整合性

Mobile Link で、同期システム全体でのトランザクションの管理を保証します。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。

トリガ

データを修正するクエリをユーザが実行すると、自動的に実行されるストアド・プロシージャの特別な形式です。

参照：

- 「ロー・レベルのトリガ」 434 ページ
- 「文レベルのトリガ」 441 ページ
- 「整合性」 438 ページ

ネットワーク・サーバ

共通ネットワークを共有するコンピュータからの接続を受け入れるデータベース・サーバです。

参照：「パーソナル・サーバ」 428 ページ。

ネットワーク・プロトコル

TCP/IP や HTTP などの通信の種類です。

パーソナル・サーバ

クライアント・アプリケーションが実行されているコンピュータと同じマシンで実行されているデータベース・サーバです。パーソナル・データベース・サーバは、単一のコンピュータ上で単一のユーザが使用しますが、そのユーザからの複数の同時接続をサポートできます。

パッケージ

Java では、それぞれが互いに関連のあるクラスの集合を指します。

ハッシュ

ハッシュは、インデックスのエントリをキーに変換する、インデックスの最適化のことです。インデックスのハッシュの目的は、必要なだけの実際のロー・データをロー ID に含めることで、インデックスされた値を特定するためのローの検索、ロード、アンパックという負荷の高い処理を避けることです。

パフォーマンス統計値

データベース・システムのパフォーマンスを反映する値です。たとえば、CURRREAD 統計値は、データベース・サーバが要求したファイル読み込みのうち、現在まだ完了していないものの数を表します。

パブリケーション

Mobile Link または SQL Remote では、同期されるデータを識別するデータベース・オブジェクトのことです。Mobile Link では、クライアント上にのみ存在します。1 つのパブリケーションは複数のアーティクルから構成されています。SQL Remote ユーザは、パブリケーションに対してサブスクリプションを作成することによって、パブリケーションを受信できます。Mobile Link ユーザは、パブリケーションに対して同期サブスクリプションを作成することによって、パブリケーションを同期できます。

参照：

- 「レプリケーション」 433 ページ
- 「アーティクル」 418 ページ
- 「パブリケーションの更新」 428 ページ

パブリケーションの更新

SQL Remote レプリケーションでは、単一のデータベース内の 1 つまたは複数のパブリケーションに対して加えられた変更のリストを指します。パブリケーションの更新は、レプリケーション・メッセージの一部として定期的にリモート・データベースへ送られます。

参照：

- 「レプリケーション」 433 ページ
- 「パブリケーション」 428 ページ

パブリッシャ

SQL Remote レプリケーションでは、レプリケートできる他のデータベースとレプリケーション・メッセージを交換できるデータベースの単一ユーザを指します。

参照：「レプリケーション」 433 ページ。

ビジネス・ルール

実世界の要求に基づくガイドラインです。通常ビジネス・ルールは、検査制約、ユーザ定義データ型、適切なトランザクションの使用により実装されます。

参照：

- 「制約」 438 ページ
- 「ユーザ定義データ型」 432 ページ

ヒストグラム

ヒストグラムは、カラム統計のもっとも重要なコンポーネントであり、データ分散を表します。SQL Anywhere は、ヒストグラムを維持して、カラムの値の分散に関する統計情報をオプティマイザに提供します。

ビット配列

ビット配列は、一連のビットを効率的に保管するのに使用される配列データ構造の種類です。ビット配列は文字列に似てますが、使用される要素は文字ではなく 0 (ゼロ) と 1 になります。ビット配列は、一般的にブール値の文字列を保持するのに使用されます。

ビュー

データベースにオブジェクトとして格納される SELECT 文です。ビューを使用すると、ユーザは 1 つまたは複数のテーブルのローやカラムのサブセットを参照できます。ユーザが特定のテーブルやテーブルの組み合わせのビューを使うたびに、テーブルに保持されているデータから再計算されます。ビューは、セキュリティの目的に有用です。またデータベース情報の表示を調整して、データへのアクセスが簡単になるようにする場合も役立ちます。

ファイルベースのダウンロード

Mobile Link では、ダウンロードがファイルとして配布されるデータの同期方法であり、同期変更のオフライン配布を可能にします。

ファイル定義データベース

Mobile Link では、ダウンロード・ファイルの作成に使用される SQL Anywhere データベースのことです。

参照：「ファイルベースのダウンロード」 429 ページ。

フェールオーバ

アクティブなサーバ、システム、またはネットワークで障害や予定外の停止が発生したときに、冗長な (スタンバイ) サーバ、システム、またはネットワークに切り替えることです。フェールオーバは自動的に発生します。

プライマリ・キー

テーブル内のすべてのローをユニークに識別する値を持つカラムまたはカラムのリストです。

参照：「外部キー」 435 ページ。

プライマリ・キー制約

プライマリ・キーのカラムに対する一意性制約です。テーブルにはプライマリ・キー制約を 1 つしか設定できません。

参照：

- 「制約」 438 ページ
- 「検査制約」 437 ページ
- 「外部キー制約」 436 ページ
- 「一意性制約」 435 ページ
- 「整合性」 438 ページ

プライマリ・テーブル

外部キー関係でプライマリ・キーを含むテーブルです。

プラグイン・モジュール

Sybase Central で、製品にアクセスしたり管理したりする方法です。プラグインは、通常、インストールすると Sybase Central にもインストールされ、自動的に登録されます。プラグインは、多くの場合、Sybase Central のメイン・ウィンドウに最上位のコンテナとして、その製品名 (たとえば SQL Anywhere) で表示されます。

参照：「Sybase Central」 417 ページ。

フル・バックアップ

データベース全体をバックアップすることです。オプションでトランザクション・ログのバックアップも可能です。フル・バックアップには、データベース内のすべての情報が含まれており、システム障害やメディア障害が発生した場合の保護として機能します。

参照：「インクリメンタル・バックアップ」 418 ページ。

プロキシ・テーブル

メタデータを含むローカル・テーブルです。リモート・データベース・サーバのテーブルに、ローカル・テーブルであるかのようにアクセスするときに使用します。

参照：「メタデータ」 431 ページ。

ベース・テーブル

データを格納する永久テーブルです。テーブルは、テンポラリ・テーブルやビューと区別するために、「ベース・テーブル」と呼ばれることがあります。

参照：

- 「テンポラリ・テーブル」 426 ページ
- 「ビュー」 429 ページ

ポーリング

Mobile Link サーバ起動同期において、Mobile Link Listerner などのライト・ウェイト・ポーラが Notifier から Push 通知を要求する方法です。

参照：「サーバ起動同期」 421 ページ。

ポリシー

QAnywhere では、メッセージ転送の発生時期を指定する方法のことです。

マテリアライズド・ビュー

計算され、ディスクに保存されたビューのことです。マテリアライズド・ビューは、ビュー (クエリ指定を使用して定義される) とテーブル (ほとんどのテーブルの操作をそのテーブル上で実行できる) の両方の特性を持ちます。

参照：

- 「ベース・テーブル」 431 ページ
- 「ビュー」 429 ページ

ミラー・ログ

参照：「トランザクション・ログ・ミラー」 427 ページ。

メタデータ

データについて説明したデータです。メタデータは、他のデータの特質と内容について記述しています。

参照：「スキーマ」 423 ページ。

メッセージ・システム

SQL Remote のレプリケーションでは、統合データベースとリモート・データベースの間でのメッセージのやりとりに使用するプロトコルのことです。SQL Anywhere では、FILE、FTP、SMTP のメッセージ・システムがサポートされています。

参照：

- 「レプリケーション」 433 ページ
- 「FILE」 412 ページ

メッセージ・ストア

QAnywhere では、メッセージを格納するクライアントおよびサーバ・デバイスのデータベースのことです。

参照：

- 「クライアント・メッセージ・ストア」 420 ページ
- 「サーバ・メッセージ・ストア」 421 ページ

メッセージ・タイプ

SQL Remote のレプリケーションでは、リモート・ユーザと統合データベースのパブリッシャとの通信方法を指定するデータベース・オブジェクトのことを指します。統合データベースには、複数のメッセージ・タイプが定義されていることがあります。これによって、リモート・ユーザはさまざまなメッセージ・システムを使って統合データベースと通信できることになります。

参照：

- 「レプリケーション」 433 ページ
- 「統合データベース」 440 ページ

メッセージ・ログ

データベース・サーバや Mobile Link サーバなどのアプリケーションからのメッセージを格納できるログです。この情報は、メッセージ・ウィンドウに表示されたり、ファイルに記録されたりすることもあります。メッセージ・ログには、情報メッセージ、エラー、警告、MESSAGE 文からのメッセージが含まれます。

メンテナンス・リリース

メンテナンス・リリースは、同じメジャー・バージョン番号を持つ旧バージョンのインストール済みソフトウェアをアップグレードするための完全なソフトウェア・セットです (バージョン番号のフォーマットは、*メジャー.マイナー.パッチ.ビルド*です)。バグ・フィックスとその他の変更については、アップグレードのリリース・ノートにリストされます。

ユーザ定義データ型

参照：「ドメイン」 426 ページ。

ライト・ウェイト・ポーラ

Mobile Link サーバ起動同期において、Mobile Link サーバからの Push 通知をポーリングするデバイス・アプリケーションです。

参照：「サーバ起動同期」 421 ページ。

リダイレクタ

クライアントと Mobile Link サーバ間で要求と応答をルート指定する Web サーバ・プラグインです。このプラグインによって、負荷分散メカニズムとフェールオーバ・メカニズムも実装されます。

リファレンス・データベース

Mobile Link では、Ultra Light クライアントの開発に使用される SQL Anywhere データベースです。開発中は、1 つの SQL Anywhere データベースをリファレンス・データベースとしても統合データベースとしても使用できます。他の製品によって作成されたデータベースは、リファレンス・データベースとして使用できません。

リモート ID

SQL Anywhere と Ultra Light データベース内のユニークな識別子で、Mobile Link によって使用されます。リモート ID は NULL に初期設定されていますが、データベースの最初の同期時に GUID に設定されます。

リモート・データベース

Mobile Link または SQL Remote では、統合データベースとデータを交換するデータベースを指します。リモート・データベースは、統合データベース内のすべてまたは一部のデータを共有できます。

参照：

- 「同期」 440 ページ
- 「統合データベース」 440 ページ

リレーショナル・データベース管理システム (RDBMS)

関連するテーブルの形式でデータを格納するデータベース管理システムです。

参照：「データベース管理システム (DBMS)」 425 ページ。

レプリケーション

物理的に異なるデータベース間でデータを共有することです。Sybase では、Mobile Link、SQL Remote、Replication Server の 3 種類のレプリケーション・テクノロジを提供しています。

レプリケーション・メッセージ

SQL Remote または Replication Server では、パブリッシュするデータベースとサブスクリプションを作成するデータベース間で送信される通信内容を指します。メッセージにはデータを含み、レプリケーション・システムで必要なパススルー文、情報があります。

参照：

- 「レプリケーション」 433 ページ
- 「パブリケーションの更新」 428 ページ

レプリケーションの頻度

SQL Remote レプリケーションでは、リモート・ユーザに対する設定の 1 つで、パブリッシャの Message Agent がレプリケーション・メッセージを他のリモート・ユーザに送信する頻度を定義します。

参照：「レプリケーション」 433 ページ。

ロー・レベルのトリガ

変更されているローごとに一回実行するトリガです。

参照：

- 「トリガ」 427 ページ
- 「文レベルのトリガ」 441 ページ

ローカル・テンポラリ・テーブル

複合文を実行する間だけ存在したり、接続が終了するまで存在したりするテンポラリ・テーブルです。データのセットを 1 回だけロードする必要がある場合にローカル・テンポラリ・テーブルが便利です。デフォルトでは、COMMIT を実行するとローが削除されます。

参照：

- 「テンポラリ・テーブル」 426 ページ
- 「グローバル・テンポラリ・テーブル」 420 ページ

ロール

概念データベース・モデルで、ある視点からの関係を説明する動詞またはフレーズを指します。各関係は 2 つのロールを使用して表すことができます。"contains (A は B を含む)" や "is a member of (B は A のメンバ)" などのロールがあります。

ロールバック・ログ

コミットされていない各トランザクションの最中に行われた変更のレコードです。ROLLBACK 要求やシステム障害が発生した場合、コミットされていないトランザクションはデータベースから破棄され、データベースは前の状態に戻ります。各トランザクションにはそれぞれロールバック・ログが作成されます。このログは、トランザクションが完了すると削除されます。

参照：「トランザクション」 427 ページ。

ロール名

外部キーの名前です。この外部キーがロール名と呼ばれるのは、外部テーブルとプライマリ・テーブル間の関係に名前を指定するためです。デフォルトでは、テーブル名がロール名になります。ただし、別の外部キーがそのテーブル名を使用している場合、デフォルトのロール名はテーブル名に 3 桁のユニークな数字を付けたものになります。ロール名は独自に作成することもできます。

参照：「外部キー」 435 ページ。

ログ・ファイル

SQL Anywhere によって管理されているトランザクションのログです。ログ・ファイルを使用すると、システム障害やメディア障害が発生してもデータベースを回復させることができます。また、データベースのパフォーマンスを向上させたり、SQL Remote を使用してデータをレプリケートしたりする場合にも使用できます。

参照：

- 「トランザクション・ログ」 427 ページ
- 「トランザクション・ログ・ミラー」 427 ページ
- 「フル・バックアップ」 430 ページ

ロック

複数のトランザクションを同時に実行しているときにデータの整合性を保護する同時制御メカニズムです。SQL Anywhere では、2 つの接続によって同じデータが同時に変更されないようにするために、また変更処理の最中に他の接続によってデータが読み込まれないようにするために、自動的にロックが適用されます。

ロックの制御は、独立性レベルを設定して行います。

参照：

- 「独立性レベル」 441 ページ
- 「同時性 (同時実行性)」 441 ページ
- 「整合性」 438 ページ

ワーク・テーブル

クエリの最適化の最中に中間結果を保管する内部保管領域です。

一意性制約

NULL 以外のすべての値が重複しないことを要求するカラムまたはカラムのセットに対する制限です。テーブルには複数の一意性制約を指定できます。

参照：

- 「外部キー制約」 436 ページ
- 「プライマリ・キー制約」 430 ページ
- 「制約」 438 ページ

解析ツリー

クエリを代数で表現したものです。

外部キー

別のテーブルにあるプライマリ・キーの値を複製する、テーブルの 1 つ以上のカラムです。テーブル間の関係は、外部キーによって確立されます。

参照：

- 「プライマリ・キー」 430 ページ
- 「外部テーブル」 436 ページ

外部キー制約

カラムまたはカラムのセットに対する制約で、テーブルのデータが別のテーブルのデータとどのように関係しているかを指定するものです。カラムのセットに外部キー制約を加えると、それらのカラムが外部キーになります。

参照：

- 「制約」 438 ページ
- 「検査制約」 437 ページ
- 「プライマリ・キー制約」 430 ページ
- 「一意性制約」 435 ページ

外部ジョイン

テーブル内のすべてのローを保護するジョインです。SQL Anywhere では、左外部ジョイン、右外部ジョイン、全外部ジョインがサポートされています。左外部ジョインは JOIN 演算子の左側にあるテーブルのローを保護し、右側にあるテーブルのローがジョイン条件を満たさない場合には NULL を返します。全外部ジョインは両方のテーブルに含まれるすべてのローを保護します。

参照：

- 「ジョイン」 422 ページ
- 「内部ジョイン」 441 ページ

外部テーブル

外部キーを持つテーブルです。

参照：「外部キー」 435 ページ。

外部ログイン

リモート・サーバとの通信に使用される代替のログイン名とパスワードです。デフォルトでは、SQL Anywhere は、クライアントに代わってリモート・サーバに接続するときは、常にそのクライアントの名前とパスワードを使用します。外部ログインを作成することによって、このデフォルトを上書きできます。外部ログインは、リモート・サーバと通信するときに使用する代替のログイン名とパスワードです。

競合

リソースについて対立する動作のことです。たとえば、データベース用語では、複数のユーザがデータベースの同じローを編集しようとした場合、そのローの編集権についての競合が発生します。

競合解決

Mobile Link では、競合解決は 2 人のユーザが別々のリモート・データベースの同じローを変更した場合にどう処理するかを指定するロジックのことです。

検査制約

指定された条件をカラムやカラムのセットに課す制約です。

参照：

- 「制約」 438 ページ
- 「外部キー制約」 436 ページ
- 「プライマリ・キー制約」 430 ページ
- 「一意性制約」 435 ページ

検証

データベース、テーブル、またはインデックスについて、特定のタイプのファイル破損をテストすることです。

作成者 ID

Ultra Light の Palm OS アプリケーションでは、アプリケーションが作成されたときに割り当てられる ID のことです。

参照元オブジェクト

テーブルなどのデータベースの別のオブジェクトをオブジェクト定義が直接参照する、ビューなどのオブジェクトです。

参照：「外部キー」 435 ページ。

参照整合性

データの整合性、特に異なるテーブルのプライマリ・キー値と外部キー値との関係を管理する規則を厳守することです。参照整合性を備えるには、それぞれの外部キーの値が、参照テーブルにあるローのプライマリ・キー値に対応するようにします。

参照：

- 「プライマリ・キー」 430 ページ
- 「外部キー」 435 ページ

参照先オブジェクト

ビューなどの別のオブジェクトの定義で直接参照される、テーブルなどのオブジェクトです。

参照：「プライマリ・キー」 430 ページ。

識別子

テーブルやカラムなどのデータベース・オブジェクトを参照するときに使う文字列です。A ～ Z、a ～ z、0 ～ 9、アンダースコア (_)、アットマーク (@)、シャープ記号 (#)、ドル記号 ($) のうち、任意の文字を識別子として使用できます。

述部

条件式です。オプションで論理演算子 AND や OR と組み合わせて、WHERE 句または HAVING 句に条件のセットを作成します。SQL では、unknown と評価される述部が false と解釈されます。

照合

データベース内のテキストのプロパティを定義する文字セットとソート順の組み合わせのことです。SQL Anywhere データベースでは、サーバを実行しているオペレーティング・システムと言語によって、デフォルトの照合が決まります。たとえば、英語版 Windows システムのデフォルトの照合は 1252LATIN1 です。照合は、照合順とも呼ばれ、文字列の比較とソートに使用します。

参照：

- 「文字セット」 441 ページ
- 「コード・ページ」 421 ページ
- 「エンコード」 419 ページ

世代番号

Mobile Link では、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のことです。

参照：「ファイルベースのダウンロード」 429 ページ。

制約

テーブルやカラムなど、特定のデータベース・オブジェクトに含まれた値に関する制約です。たとえば、一意性制約があるカラム内の値は、すべて異なっている必要があります。テーブルに、そのテーブルの情報と他のテーブルのデータがどのように関係しているのかを指定する外部キー制約が設定されていることもあります。

参照：

- 「検査制約」 437 ページ
- 「外部キー制約」 436 ページ
- 「プライマリ・キー制約」 430 ページ
- 「一意性制約」 435 ページ

整合性

データが適切かつ正確であり、データベースの関係構造が保たれていることを保証する規則を厳守することです。

参照：「参照整合性」 437 ページ。

正規化

データベース・スキーマを改善することです。リレーショナル・データベース理論に基づく規則に従って、冗長性を排除したり、編成を改良します。

正規表現

正規表現は、文字列内で検索するパターンを定義する、一連の文字、ワイルドカード、演算子です。

生成されたジョイン条件

自動的に生成される、ジョインの結果に対する制限です。キーとナチュラルの 2 種類があります。キー・ジョインは、KEY JOIN を指定したとき、またはキーワード JOIN を指定したが、CROSS、NATURAL、または ON を使用しなかった場合に生成されます。キー・ジョインの場合、生成されたジョイン条件はテーブル間の外部キー関係に基づいています。ナチュラル・ジョインは NATURAL JOIN を指定したときに生成され、生成されたジョイン条件は、2 つのテーブルの共通のカラム名に基づきます。

参照：

- 「ジョイン」 422 ページ
- 「ジョイン条件」 423 ページ

接続 ID

クライアント・アプリケーションとデータベース間の特定の接続に付けられるユニークな識別番号です。現在の接続 ID を確認するには、次の SQL 文を使用します。

```
SELECT CONNECTION_PROPERTY( 'Number' );
```

接続プロファイル

ユーザ名、パスワード、サーバ名などの、データベースに接続するために必要なパラメータのセットです。便宜的に保管され使用されます。

接続起動同期

Mobile Link のサーバ起動同期の 1 つの形式で、接続が変更されたときに同期が開始されます。

参照：「サーバ起動同期」 421 ページ。

相関名

クエリの FROM 句内で使用されるテーブルやビューの名前です。テーブルやビューの元の名前か、FROM 句で定義した代替名のいずれかになります。

抽出

SQL Remote レプリケーションでは、統合データベースから適切な構造とデータをアンロードする動作を指します。この情報は、リモート・データベースを初期化するときに使用されます。

参照：「レプリケーション」 433 ページ。

通信ストリーム

Mobile Link では、Mobile Link クライアントと Mobile Link サーバ間での通信にネットワーク・プロトコルが使用されます。

転送ルール

QAnywhere では、メッセージの転送を発生させる時期、転送するメッセージ、メッセージを削除する時期を決定する論理のことです。

統合データベース

分散データベース環境で、データのマスタ・コピーを格納するデータベースです。競合や不一致が発生した場合、データのプライマリ・コピーは統合データベースにあるとみなされます。

参照：

- 「同期」 440 ページ
- 「レプリケーション」 433 ページ

統合化ログイン

オペレーティング・システムへのログイン、ネットワークへのログイン、データベースへの接続に、同一のユーザ ID とパスワードを使用するログイン機能の 1 つです。

動的 SQL

実行される前に作成したプログラムによって生成される SQL です。Ultra Light の動的 SQL は、占有容量の小さいデバイス用に設計された変形型です。

同期

Mobile Link テクノロジを使用してデータベース間でデータをレプリケートする処理です。

SQL Remote では、同期はデータの初期セットを使ってリモート・データベースを初期化する処理を表すために特に使用されます。

参照：

- 「Mobile Link」 413 ページ
- 「SQL Remote」 416 ページ

同時性 (同時実行性)

互いに独立し、場合によっては競合する可能性のある 2 つ以上の処理を同時に実行することです。SQL Anywhere では、自動的にロックを使用して各トランザクションを独立させ、同時に稼働するそれぞれのアプリケーションが一貫したデータのセットを参照できるようにします。

参照：

- 「トランザクション」 427 ページ
- 「独立性レベル」 441 ページ

独立性レベル

あるトランザクションの操作が、同時に処理されている別のトランザクションの操作からどの程度参照できるかを示します。独立性レベルには 0 から 3 までの 4 つのレベルがあります。最も高い独立性レベルには 3 が設定されます。デフォルトでは、レベルは 0 に設定されています。SQL Anywhere では、スナップショット、文のスナップショット、読み込み専用文のスナップショットの 3 つのスナップショットの独立性レベルがあります。

参照：「スナップショット・アイソレーション」 423 ページ。

内部ジョイン

2 つのテーブルがジョイン条件を満たす場合だけ、結果セットにローが表示されるジョインです。内部ジョインがデフォルトです。

参照：

- 「ジョイン」 422 ページ
- 「外部ジョイン」 436 ページ

物理インデックス

インデックスがディスクに保存されるときの実際のインデックス構造です。

文レベルのトリガ

トリガ付きの文の処理が完了した後に実行されるトリガです。

参照：

- 「トリガ」 427 ページ
- 「ロー・レベルのトリガ」 434 ページ

文字セット

文字セットは記号、文字、数字、スペースなどから成ります。"ISO-8859-1" は文字セットの例です。Latin1 とも呼ばれます。

参照：

- 「コード・ページ」 421 ページ
- 「エンコード」 419 ページ
- 「照合」 438 ページ

文字列リテラル

文字列リテラルとは、一重引用符 (') で囲まれ、シーケンスで並べられた文字のことです。

論理インデックス

物理インデックスへの参照 (ポインタ) です。ディスクに保存される論理インデックス用のインデックス構造はありません。

索引

記号

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Z

あ

い

う

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

ね

は

ひ

ふ

へ

ほ

ま

れ

ろ

わ

