SYBASE iAnywhere

Mobile Link
サーバ起動同期

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 のサーバ起動同期について説明します。サーバ起動同期とは、Mobile Link サーバから同期の開始またはリモート・デバイス上でのアクションの実行を可能にする機能です。

対象読者

このマニュアルは、サーバ起動同期の設定を行う Mobile Link ユーザを対象としています。

始める前に

Mobile Link の詳細については、Mobile Link - クイック・スタート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。

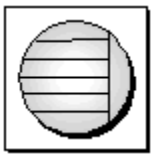
- **環境変数**　マニュアルでは、環境変数設定が引用されます。Windows のシェルでは、環境変数は構文 *%ENVVAR%* を使用して指定されます。UNIX のシェルでは、環境変数は構文 *$ENVVAR* または *${ENVVAR}* を使用して指定されます。

グラフィック・アイコン

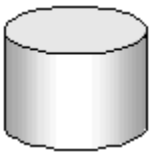
このマニュアルでは、次のアイコンを使用します。

- 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 サーバ起動同期の概要

目次

サーバ起動同期の概要

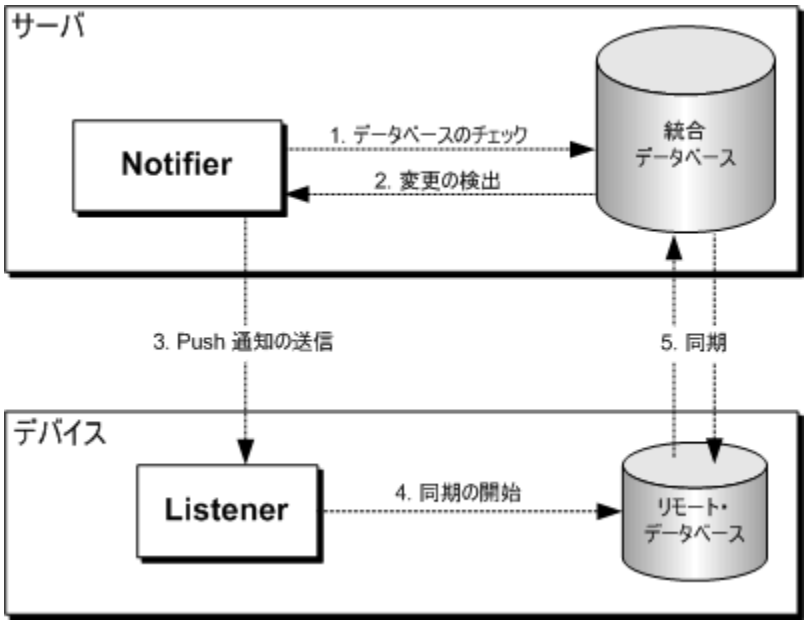
Mobile Link サーバ起動同期を使用すると、統合データベースから同期を開始できます。リモート・データベースに Push 通知を送信し、リモート・データベースから統合データベースを更新できます。この Mobile Link コンポーネントには、統合データベースで発生した変更の検出による同期の開始、Push 通知を送信するデバイスの選択、その Push 通知に対するデバイスの応答方法の決定を行うためのプログラム可能なオプションが用意されています。

例

トラック運送会社がモバイル・デバイスを自社の運転手に支給するとします。各デバイスではデータベースが実行されており、このデータベースには経路と配達場所が格納されています。道路が渋滞しているという情報をある運転手が送信すると、このレポートは統合データベースに送信されます。Notifier というサーバ側 Mobile Link コンポーネントによってレポートが検出され、渋滞の影響を受ける経路にいる他の運転手に Push 通知が送信されます。この Push 通知の結果、リモート・データベースが同期されるため、運転手は別の経路を使用できます。

サーバ起動同期のプロセス

次の図では、Notifier が統合データベースをチェックし、変更があるかどうかを確認しています。Notifier によって Push 通知がデバイスに送信され、その結果、リモート・データベースが統合データベースと同期されます。



サーバ起動同期プロセスでは、次の手順が実行されます。

1. Notifier はビジネス・ロジックに基づいたクエリを使用して統合データベースをチェックし、リモート・データベースとの同期が必要な変更があるかどうかを確認します。
2. 変更を検出すると、Notifier はデバイスに Push 通知を送信する準備を行います。

3. Notifier は Push 通知を送信します。Push 通知は、Device Tracker、UDP、SMTP、または SYNC ゲートウェイを使用して送信できます。
4. Listener は、件名、内容、または送信元をメッセージ・フィルタと照らし合わせて比較します。
5. フィルタ条件が満たされると、アクションが開始されます。たとえば、標準的な実装では、アクションによって Mobile Link クライアントを実行したり、Ultra Light アプリケーションを起動したりできます。

サーバ起動同期のコンポーネント

Mobile Link サーバ起動同期では、次のコンポーネントが必要です。

- **Push 要求** Push 要求は結果セット内の値のローで、デバイスに Push 通知を送信するよう Notifier に指示します。Push 要求によって、サーバ起動同期が実行されます。Notifier などの、あらゆるデータベース・アプリケーションで Push 要求を作成できます。たとえば、価格が変更されたときにアクティブになるデータベース・トリガを使用して、Push 要求を作成できます。「Push 要求」 10 ページを参照してください。
- **Mobile Link Notifier** Notifier は、Mobile Link サーバに統合されたプログラムです。統合データベースを頻繁にチェックして、Push 要求があるかどうかを確認します。Notifier が Push 要求をチェックする頻度を制御するには、Notifier のプロパティを指定します。Push 要求をチェックするビジネス・ロジックと、通知対象のデバイスを決定するビジネス・ロジックを指定する必要があります。Notifier が Push 要求を検出すると、デバイスに Push 通知が送信されます。「Notifier」 16 ページを参照してください。
- **Mobile Link Listener** Listener は、デバイス上で実行される 1 つのプログラムです。Notifier から Push 通知を受信すると、メッセージ・ハンドラを使用してメッセージをフィルタし、アクションを開始します。一般的なアプリケーションのアクションは同期の呼び出しですが、アプリケーションから他のアクションも実行できます。選択したサーバ・ソースからの Push 通知や特定の内容が含まれる Push 通知に対して、異なる動作をするように Listener を設定できます。

  Windows デバイスでは、Listener はコマンド・ライン・オプションを使用して設定する実行プログラムです。Push 通知を受信するには、デバイスの電源がオンになっていて、Listener が動作中である必要があります。「Windows デバイス用の Listener ユーティリティ」 64 ページを参照してください。

  Palm デバイス用の Mobile Link Listener を使用するには、Windows デスクトップで Listener 設定ユーティリティを実行して設定ファイルを作成し、このファイルをデバイスにコピーしてから Listener を実行します。「Palm デバイス用ユーティリティ」 87 ページを参照してください。
- **ライトウェイト・ポーラ** ライトウェイト・ポーラは、指定された時間間隔で 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。ライトウェイト・ポーラを使用すると、ゲートウェイを設定する代替手段となります。また、サーバへの永続的な接続を必要とせず、バッテリの寿命を伸ばすことができるため、ライトウェイト・ポーラを使用することをおすすめします。

  Listener は、Listener コマンド・ライン・オプションを使用して設定できるライトウェイト・ポーラです。別の方法として、ライトウェイト・ポーリング API を使用して、独自のライトウェイト・ポーラを作成できます。

  次の項を参照してください。

  ○「ライトウェイト・ポーリング・オプションの設定」 23 ページ
  ○「ライトウェイト・ポーリング API」 26 ページ
- **ゲートウェイ (ライトウェイト・ポーラの代替手段)** ゲートウェイでは、デバイスに Push 通知を送信するための Notifier インタフェースを提供します。ゲートウェイは、ライトウェイト・

ポーラの代替となる手段です。デバイス・トラッキング・ゲートウェイ、SYNC ゲートウェイ、UDP ゲートウェイ、または SMTP ゲートウェイを使用して、メッセージを送信できます。「ゲートウェイと Carrier」 32 ページを参照してください。

サーバ起動同期の配備に関する考慮事項

サーバ起動同期アプリケーションを配備する前に、次の点について検討してください。

UDP ゲートウェイを使用する場合のデバイスの制限事項

- デバイスの IP アドレスには、Mobile Link サーバから直接アクセスできる必要があります。
- Windows デバイスの IP アドレスに Mobile Link サーバから直接アクセスできない場合、UDP 通知の IP 追跡は機能しません。

デバイス・トラッキングの制限事項

Palm デバイス用の Listener と Adaptive Server Anywhere 9.0.0 以前の Listener では、デバイス・トラッキングをサポートしていません。これらの Listener でデバイス・トラッキングを使用するには、デバイス・トラッキングを手動で設定する必要があります。「デバイス・トラッキングのサポートの追加」 33 ページを参照してください。

ライトウェイト・ポーリングの制限事項

ライトウェイト・ポーリングは Palm デバイス用の Listener ではサポートされていません。

サポートされるデバイス・プラットフォーム

Mobile Link Listener は、Windows、Windows Mobile、Palm OS のデバイスのみでサポートされています。

サーバ起動同期のクイック・スタート・ガイド

この手順を完了する前に、通常の同期用の Mobile Link を設定する必要があります。Mobile Link - クイック・スタートを参照してください。

1. Mobile Link サーバで、Push 要求を格納するための統合データベースを準備します。「Push 要求の要件」 10 ページを参照してください。
2. Mobile Link サーバで、Push 要求を作成および管理する Notifier イベントを設定します。「Notifier イベントおよびプロパティの設定」 17 ページを参照してください。
3. デバイスで、ライトウェイト・ポーラを設定します。「ライトウェイト・ポーラ」 26 ページを参照してください。

   ライトウェイト・ポーラを使用しない場合は、Mobile Link サーバでサポートされているゲートウェイを設定します。SMTP ゲートウェイを使用する場合は、Carrier も設定する必要があります。「ゲートウェイと Carrier」 32 ページを参照してください。
4. デバイスで、メッセージをフィルタしてアクションを実行する Listener を設定します。「メッセージ・ハンドラ」 19 ページを参照してください。

Sybase Central を使用してサーバ起動同期を設定する方法については、「モデル・モードでのサーバ起動同期の設定」『Mobile Link - クイック・スタート』を参照してください。

その他のリソース

- サンプル・アプリケーションが *samples-dir¥MobiLink¥* ディレクトリにインストールされています。*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。サーバ起動同期に関連するすべてのアプリケーションは、**SIS_** プレフィクスの付いたディレクトリに配置されています。
- 「追加情報およびリソースの検索」『このヘルプの使い方』を参照してください。

サーバ起動同期の設定

目次

Push 要求

Push 要求は、Notifier が、Push 通知をデバイスに送信する必要があるかどうかを確認する場合にチェックする、結果セット内の値のローです。Notifier は Push 通知内に Push 要求を配置して、Push 通知を送信します。典型的なサーバ起動同期設定では、Push 要求にメッセージの内容とターゲット・デバイスの情報が含まれます。Push 通知を送信するには、まず、Notifier で Push 要求を検出できるよう Notifier イベントを設定する必要があります。

Push 要求の要件

Push 要求の要件は、Mobile Link サーバがデバイスとの通信に使用している方法によって異なります。すべての Push 要求に、subject カラムと content カラムが必要となります。ライトウェイト・ポーリングを使用して Push 通知をポーリングする場合は、poll key カラムを作成して Push 通知を識別します。ゲートウェイを使用して Push 通知を送信する場合は、gateway カラムと address カラムを作成します。

システムに Push 要求カラムがある場合は、カラムを作成する必要はありません。Push 要求の要件が満たされると、Push 要求を使用できます。「Push 要求の使用」 12 ページを参照してください。

ライトウェイト・ポーラを使用する場合の Push 要求の要件 (推奨)

ライトウェイト・ポーラを使用して Push 通知をポーリングする場合は、次のカラムを作成してください。

カラム	型	説明
ポーリング・キー	VARCHAR	ライトウェイト・ポーラを識別するために使用するキー。各ライトウェイト・ポーラはユニークなキーを送信して、Mobile Link サーバ上で自身を識別します。
このマニュアルの内容	VARCHAR	メッセージの件名の行です。
content	VARCHAR	メッセージの内容。

ゲートウェイを使用する場合の Push 要求の要件 (推奨)

特に指定がないかぎり、ゲートウェイを使用して Push 通知を送信する場合は、次のカラムを作成してください。

カラム	型	説明
[要求 ID]	INTEGER	オプション。Push 要求のユニークな ID。 一部の Notifier イベントでは、このカラム名が必須です。「Notifier イベント」 43 ページを参照してください。

カラム	型	説明
ゲートウェイ	VARCHAR	メッセージの送信先ゲートウェイの名前。
このマニュアルの内容	VARCHAR	メッセージの件名の行です。
content	VARCHAR	メッセージの内容。
address	VARCHAR	デバイスの送信先アドレス。
再送間隔	VARCHAR	オプション。メッセージが再送される時間間隔。 resend interval は、信頼性の低いネットワークで UDP ゲートウェイを使用する場合に便利です。Notifier は、Push 要求に関連付けられたすべての属性が変更されないことを前提としています。要求を最初にポーリングした後、後続の更新は無視されます。次のポーリング時刻の前に Push 通知を送信する必要がある場合、Notifier は次のポーリング間隔を自動的に調整します。Push 要求の送信を停止するには、request_cursor イベントで同期論理を使用します。対象 Listener から受信確認が届くと、次の再送は停止されます。「request_cursor イベント」 47 ページを参照してください。
存続期間	VARCHAR	オプション。再送の有効期限が切れるまでの時間です。

例

次の例では、SQL Anywhere 統合データベースのテーブルで必要なカラムを作成して、ライトウェイト・ポーリングを使用する場合の Push 要求の要件を満たします。

```
CREATE TABLE PushRequest (
   req_id INTEGER DEFAULT AUTOINCREMENT PRIMARY KEY,
   poll_key VARCHAR(128),
   subject VARCHAR(128),
   content VARCHAR(128)
)
```

このようなテーブルの作成が必要になるのは、Push 要求のカラムが他の場所で使用できない場合のみです。Push 要求のカラムは、複数のテーブル間、既存の複数のテーブル、または 1 つのビューに作成できます。

参照

- 「Notifier イベントおよびプロパティの設定」 17 ページ
- 「request_cursor イベント」 47 ページ

Push 要求の使用

Push 要求の生成

Push 要求を生成するには、サーバ起動同期に必要な Push 要求のカラムが対象となるデータベースに含まれている必要があります。また、単一のデータベース・クエリを使用して値を取得できる必要もあります。Push 要求は、Push 要求のカラムを選択する request_cursor イベントでデータベース・クエリを指定すると、自動的に生成されます。Push 要求の要件の詳細については、「Push 要求の要件」 10 ページを参照してください。

例

統合データベース内に、ライトウェイト・ポーラ用の Push 要求を含むテーブル PushRequest が含まれています。リモート・デバイスは、Mobile Link サーバで unique_device_ID として識別されます。このサーバでは、次の SQL スクリプトを使用して同期を要求します。

```
INSERT INTO PushRequest (poll_key, subject, content) VALUES ('unique_device_ID', 'synchronize', 'ASAP');
```

このスクリプトを使用して値を挿入しても、Push 要求は生成されません。値は、request_cursor イベントでデータベース・クエリを使用して選択してください。Push 要求を生成するには、サーバで次の request_cursor イベント・スクリプトを使用します。

```
SELECT poll_key, subject, content FROM PushRequest;
```

unique_device_ID デバイスによって Push 通知のサーバがポーリングされると、Push 要求が生成されます。Push 通知の件名は **synchronize** で、内容は **ASAP** です。

Push 要求の制限事項

次の表は、Push 要求の制限事項をカラムごとに示します。

カラム	型	制限
[要求 ID]	INTEGER	この値は、ユニークなプライマリ・キーにしてください。
ポーリング・キー	VARCHAR	ライトウェイト・ポーラの使用時にのみ必要です。 ポーリング・キーには制限がありません。

カラム	型	制限
ゲートウェイ	VARCHAR	ゲートウェイを使用する場合にのみ必要です。 この値には、有効なゲートウェイの名前を設定してください。独自のカスタム・ゲートウェイ名を指定するか、または次の事前に設定されたゲートウェイ名のいずれかを選択します。 ● **Default-DeviceTracker** ● **Default-SMTP** ● **Default-SYNC** ● **Default-UDP** 「ライトウェイト・ポーラの代わりにゲートウェイを使用する」 32 ページを参照してください。
このマニュアルの内容	VARCHAR	この値の設定では、英数字以外の文字を使用しないでください。中カッコ、カレット、二重引用符、一重引用符、角カッコは内部用に予約されているため、subject カラムで使用しないでください。
content	VARCHAR	メッセージの内容には制限がありません。
address	VARCHAR	ゲートウェイを使用する場合にのみ必要です。 UDP ゲートウェイの場合、この値は IP アドレスまたはホスト名にしてください。次のフォーマットのポート番号のサフィックスがサポートされています。 ● *IP-address*:*port-number* ● *hostname*:*port-number* SMTP ゲートウェイの場合、この値は電子メール・アドレスにしてください。 SYNC ゲートウェイとデバイス・トラッキング・ゲートウェイの場合、この値は Listener -t+ オプションで定義されている受信者名にしてください。「-t オプション」 75 ページを参照してください。
再送間隔	VARCHAR	デフォルトでは、この値は分単位で測定されます。秒、分、時間それぞれの単位として **S**、**M**、**H** を指定できます。また、単位を組み合わせることもできます。たとえば、1H 30M 10S は、メッセージを 1 時間ごと、30 分ごと、10 秒ごとに再送するよう Notifier に通知します。 この値が NULL または未指定の場合、デフォルトでは一度だけ送信され、再送は行われません。

カラム	型	制限
存続期間	VARCHAR	デフォルトでは、この値は分単位で測定されます。秒、分、時間それぞれの単位として **S**、**M**、**H** を指定できます。また、単位を組み合わせることもできます。たとえば、3H 30M 10S は、メッセージの最初の送信の 3 時間後、30 分後、10 秒後に再送を停止するよう Notifier に通知します。 この値が NULL または未指定の場合、デフォルトでは一度だけ送信され、再送は行われません。

Push 要求の検出と Push 通知の送信

Notifier では、request_cursor イベントを頻繁に起動することによって、Push 要求を検出します。デフォルトでは、このイベントにはスクリプトが指定されていません。Notifier が Push 要求を検出できるよう、request_cursor イベントを指定してください。典型的なアプリケーションでは、request_cursor イベント・スクリプトは SELECT 文です。「request_cursor イベント」 47 ページを参照してください。

次の例では、ml_add_property システム・プロシージャを使用して、Simple という名前のカスタム Notifier 用の request_cursor イベント・スクリプトを作成します。SELECT 文は、テーブル PushRequest から Push 要求を検出するよう Notifier に通知します。

```
CALL ml_add_property('SIS', 'Notifier(Simple)', 'request_cursor',
   'SELECT poll_key, subject, content FROM PushRequest'
);
```

注意
カラムは、Push 要求で指定されている順序と同じ順序で選択してください。「Push 要求の要件」 10 ページを参照してください。

Notifier イベントの設定については、「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

Push 要求の削除

Push 通知がビジネス規則を満たし、この規則に従って送信された後に通知されたデバイスの情報が更新されない場合、Notifier は通知を再送します。Push 要求が満たされたら、Notifier で古い Push 要求が検出されないようにする必要があります。同期目的で Push 通知が送信された場合は、同期スクリプトを使用して Push 要求を削除できます。

request_delete イベントを使用して要求 ID ごとに Push 要求を削除できますが、Push 要求に request ID カラムが含まれている必要があります。また、配信確認を有効にしてください。

注意
配信確認は Palm デバイスでは使用できませんが、独自の配信確認メカニズムを実装して Push 要求を削除できます。たとえば、独自の同期論理を使用すると、特定の同期が発生した場合に Push 要求を削除できます。配信確認の無効化の詳細については、「Windows 用の Listener キーワード」 77 ページを参照してください。

次の項を参照してください。

- 「Push 要求の要件」 10 ページ
- 「request_delete イベント」 48 ページ
- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ

Notifier

Notifier は Mobile Link サーバに統合されたプログラムで、統合データベースで Push 要求を頻繁にチェックします。Push 要求が検出されると、デバイスに Push 通知を送信します。また、Notifier は一連のイベントを実行して、データのモニタ、Push 要求の管理、配信確認の処理、エラーの処理を行うスクリプトを作成できるようにします。

Mobile Link サーバを最初にロードすると、Notifier が起動します。Mobile Link サーバの単一インスタンス内で複数の Notifier を実行できます。複数の Notifier の使用方法の例については、*samples-dir¥MobiLink¥SIS_MultipleNotifier* にあるサンプル・アプリケーションを参照してください。*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。

データベースへの接続が失われると、Notifier は再びアクセスできるまで接続のリカバリを試みます。リカバリ後、Notifier は引き続き同じ設定で動作します。

Mobile Link サーバ・ファームでの Notifier

11.0 以前の Mobile Link では、Mobile Link サーバ・ファームでサーバ起動同期を使用すると、冗長な Push 通知が発生し、追加の同期が行われ、Mobile Link サーバ・ファームの統合データベースに負荷がかかることがありました。現行バージョンでは、ファーム内のすべての Mobile Link サーバで Notifier を実行できるようになりました。これらの Notifier によって、同じ Listener への冗長な Push 通知がなくなります。ローカルの Mobile Link サーバに接続する必要があるときは、mlsrv11 -lsc サーバ・オプションを使用して、他のサーバに情報を渡すことができます。「-lsc オプション」『Mobile Link - サーバ管理』を参照してください。

この機能を使用すると、1 つの Notifier がプライマリ、他のすべての Notifier がセカンダリになります。プライマリ Notifier は、直接、またはセカンダリを通じて間接的に、Push 通知を制御します。セカンダリ Notifier も、Listener 情報をプライマリ Notifier にルート指定するので、Listener の場所と、Listener への経路を認識しています。

プライマリ Notifier を実行している Mobile Link サーバに障害が発生した場合、サーバ・ファームは新しいプライマリ Notifier を選択し、通知を継続します。

Listener は、どれがプライマリ・サーバかを知らなくても、ファーム内のどの Mobile Link サーバにも接続できます。

この機能を使用するには、ファーム内のすべての Mobile Link サーバに次の mlsrv11 コマンド・ライン・オプションが必要です。

- 「-lsc オプション」『Mobile Link - サーバ管理』
- 「-notifier オプション」『Mobile Link - サーバ管理』
- 「-zs オプション」『Mobile Link - サーバ管理』

例

host001 の場合：

```
mlsrv11 -notifier -zs ml001 -lsc tcpip(host=host001;port=2439) ...
```

host007 の場合：

```
mlsrv11 -notifier -zs ml007 -lsc tcpip(host=host007;port=2439) ...
```

Notifier イベントおよびプロパティの設定

Notifier イベントを使用することにより、サーバ起動同期処理全体を管理するスクリプトを埋め込むことができます。Notifier イベントの流れとその起動方法の詳細については、「Notifier イベント」 43 ページを参照してください。

たとえば、次のようなタスクを実行する Notifier イベントを設定できます。

- request_cursor イベントを使用して、Push 要求で送信する情報、送信方法、送信先を決定する。
- begin_poll イベントを使用して、統合データベースの変化に応じて Push 要求を作成する (高度な使用法)。
- request_delete イベントを使用して、Push 要求を削除する (要求に応じて)。
- end_poll イベントを使用して、Notifier のポーリングを追跡し、テーブル・データをクリーンアップする (高度な使用法)。

Notifier プロパティはイベントに似ています。イベントは通知プロセスを管理しますが、プロパティは Notifier の動作を管理します。たとえば、Notifier プロパティでは、Notifier が統合データベースをポーリングする頻度や起動時に Notifier を有効にするかどうかを決定します。Notifier プロパティおよびイベントは、サーバ側の設定として設定します。詳細については、「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

Notifier の起動

Mobile Link サーバをロードすると、有効な Notifier がすべて起動します。Notifier を無効にするには、有効な Notifier のプロパティ値を false に設定してください。「Notifier プロパティ」 55 ページを参照してください。

Notifier を起動するには、次のいずれかの方法を使用します。

- Notifier を設定し、指定した -notifier オプションで mlsrv11 を実行する。
- 設定が Notifier 設定ファイルに格納されている場合は、コマンド・ラインで mlsrv11 を実行してデータベースをロードし、-notifier オプションを使用してファイルを指定する。たとえば、ファイル *myfirst.Notifier* を使用する場合は、次のコマンドによって、このファイルに指定されているプロパティおよびイベントを使用するよう Mobile Link サーバを設定します。

  ```
  mlsrv11 ... -notifier c:¥myfirst.Notifier
  ```

  QAnywhere を使用している場合は、コマンド・ラインで、指定した -m オプションとともに mlsrv11 を実行してデータベースをロードします。

参照

- 「-notifier オプション」 『Mobile Link - サーバ管理』
- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ
- 「Notifier イベントおよびプロパティの設定」 17 ページ
- 「-m オプション」 『Mobile Link - サーバ管理』

Listener

Listener は、デバイス上で実行される 1 つのプログラムです。Listener は、Notifier からの Push 通知を受信して、アクションを開始します。サーバ起動同期にゲートウェイを使用している場合、Listener は、デバイス・トラッキング情報を統合データベースにアップロードできます。

参照

- 「デバイス・トラッキング・ゲートウェイ」 33 ページ
- 「メッセージ・ハンドラ」 19 ページ
- 「Windows デバイス用の Listener ユーティリティ」 64 ページ
- 「Palm デバイス用の Listener 設定ユーティリティ」 87 ページ

例

次のコマンドは、Windows デバイスの Mobile Link Listener ユーティリティを起動します。

```
dblsn -v2 -m -ot dblsn.log
   -l "poll_connect='host=localhost';
   poll_key=sis_user1;
   poll_every=10;
   subject=sync;
   action='start dbmlsync.exe
      -c eng=rem1;uid=DBA;pwd=sql
      -ot dbmlsyncOut.txt -qc';"
```

このコマンドは、冗長性レベルが 2 に設定された Listener をロードし、メッセージのロギングを有効にして、サーバが **localhost** にあることを指定します。*dblsn.log* ファイルは、出力が書き込まれる前にトランケートされます。Listener は、10 秒ごとに Push 通知をポーリングします。Listener が **sync** という件名の Push 通知を受信すると、Mobile Link クライアント・アプリケーションが起動します。

Windows Listener のコマンド・ライン・オプションの詳細については、「Windows 用の Listener オプション」 65 ページを参照してください。

メッセージ・ハンドラ

メッセージ・ハンドラは Listener コンポーネントで、Push 通知のメッセージの内容をスキャンしてアクションを開始します。また、メッセージ・ハンドラを使用すると、サーバ検索やポーリング頻度などのライトウェイト・ポーリング・オプションを指定できます。

メッセージ・ハンドラは、次のコンポーネントから構成されています。

- **フィルタのキーワード**　Push 通知が前処理されると、フィルタのキーワードを使用してメッセージの内容をスキャンできます。フィルタ条件が満たされると、アクションが開始されます。たとえば、**subject** キーワードを指定して特定の件名を含むメッセージをフィルタしたり、**sender** キーワードを指定して特定の Mobile Link サーバから受信したメッセージをフィルタしたりできます。
- **アクション**　メッセージでフィルタ条件が満たされると、アクションが開始されます。典型的なアプリケーションでは、アクションを指定して同期を開始しますが、別の操作も実行で

きます。エラー処理の支援として、元のアクションが失敗した場合にインスタンスを処理できるよう代替アクションを指定します。

- **ポーリング設定**　ポーリング設定では、Listener が Mobile Link サーバで Push 通知をポーリングする方法を設定できます。
- **オプション**　オプションを使用すると、配信確認やアクション確認などのリモート設定を制御できます。

> **注意**
> 一部のメッセージ・ハンドラ・オプションは Palm デバイスの Mobile Link Listener ユーティリティでは使用できません。「Palm デバイス用の Listener 設定ユーティリティ」 87 ページを参照してください。

メッセージ・ハンドラは、dblsn -l オプションを使用して作成できます。複数のメッセージ・ハンドラを指定できます。

参照

- 「-l オプション」 71 ページ
- 「Windows 用の Listener キーワード」 77 ページ
- 「Windows 用の Listener アクション・コマンド」 80 ページ

メッセージ・ハンドラの使用

Listener が Push 通知を受信すると、Push 通知はメッセージを抽出します。メッセージは複数のキーワードに分割されています。**message** キーワードには、メッセージ全体が未加工形式で記述されています。このメッセージは、**subject** キーワード、**content** キーワード、**sender** キーワードに分割されます。これらのキーワードは、メッセージ・フィルタを介して実行され、開始するアクションを決定します。これらのキーワードを使用したメッセージのフィルタリングの詳細については、「メッセージのフィルタリング」 20 ページを参照してください。

メッセージのフィルタリング

フィルタ・キーワードを使用して、Push 通知の一部とユーザ定義のフレーズを比較します。2 つのフレーズのテキストが同等の場合、アクションが開始されます。メッセージ・フィルタリング用の Push 通知の前処理については、「メッセージ構文」 128 ページを参照してください。

フィルタ・キーワードを指定するには、次の構文を使用して Listener を実行します。

```
dblsn ... -l "filter-keyword-name='content to filter';action='...'"
```

-l オプションを複数回使用すると複数のファイルを作成できますが、各 -l インスタンスのアクションも指定してください。アクションは、すべてのフィルタが満たされた場合にのみ開始されます。

次の各キーワードは、メッセージ・ハンドラに 1 回のみ表示されます。

- **content**　メッセージのフィルタリングには、このキーワードと subject キーワードを使用することをおすすめします。このキーワードは、内容に基づいてメッセージをフィルタリングするために使用します。次に例を示します。

  ```
  dblsn -l "content='your content filter here';action='...'"
  ```

- **subject**　メッセージのフィルタリングには、このキーワードと content キーワードを使用することをおすすめします。このキーワードは、件名に基づいてメッセージをフィルタリングするために使用します。次に例を示します。

  ```
  dblsn -l "subject='your subject filter here';action='...'"
  ```

- **message**　このキーワードは、未加工データに基づいてメッセージをフィルタリングするために使用します。フィルタ値がメッセージの正確な長さと一致するようにしてください。このキーワードには変数構造があるため、使用しないことをおすすめします。メッセージ・フィルタリング用の Push 通知の前処理については、「メッセージ構文」 128 ページを参照してください。

- **message_start**　このキーワードは、未加工データの先頭からの一部に基づいてメッセージをフィルタリングするために使用します。メッセージ・フィルタリング用の Push 通知の前処理については、「メッセージ構文」 128 ページを参照してください。

  このキーワードを指定すると、Listener は action 変数の $message_start と $message_end を作成します。

- **sender**　このキーワードは、送信者に基づいてメッセージをフィルタリングするために使用します。このキーワードは、特定の Notifier が送信した Push 通知を追跡するのに役立ちます。この値は、使用されているゲートウェイによって異なります。UDP ゲートウェイの場合、この値はゲートウェイのホストの IP アドレスです。SYNC ゲートウェイの場合は、**MobiLink** です。また、SMTP ゲートウェイの場合は、ご使用の無線通信事業者によって異なります。「ゲートウェイと Carrier」 32 ページを参照してください。

参照

- 「Windows 用の Listener キーワード」 77 ページ
- 「action 変数」 22 ページ

アクションの開始

メッセージがフィルタの条件を満たしている場合に、アクションが開始されます。Push 通知のフィルタリングの詳細については、「メッセージのフィルタリング」 20 ページを参照してください。

アクションを指定するには、次の構文を使用して Listener を実行します。

```
dblsn ... -l "...;action='action-command command statement'"
```

次のアクション・コマンドを使用すると、メッセージのフィルタリング時にさまざまなタスクを実行できます。

- **START**　アプリケーションを開始し、バックグラウンドで実行されるようにします。

- **RUN**　アプリケーションを実行し、追加の Push 通知を受信する前にアプリケーションの完了を待機します。
- **POST**　すでに実行中のプロセスにウィンドウ・メッセージを送信する。このコマンドは、Windows デバイスでしか使用できません。
- **SOCKET**　TCP/IP 接続を使用して、アプリケーションにメッセージを送信します。
- **DBLSN FULL SHUTDOWN**　Listener を停止します。

アクション・コマンドの詳細なリストと構文の参照情報詳細については、「Windows 用の Listener アクション・コマンド」 80 ページを参照してください。

アクションへの action 変数の組み込みの詳細については、「action 変数」 22 ページを参照してください。

action 変数

action 変数を使用すると、メッセージ・フィルタまたはアクションからの Push 通知の一部を参照できます。「アクションの開始」 21 ページと「メッセージのフィルタリング」 20 ページを参照してください。

action 変数の設定方法

ほとんどの action 変数は、Push 通知が受信されるたびに自動的に設定されます。変数名は、メッセージ構文で指定されている名前と似ています。たとえば、*message* は $message action 変数を、*subject* は $subject action 変数を、*sender* は $sender action 変数を、*content* は $content action 変数をそれぞれ設定します。「メッセージ構文」 128 ページを参照してください。

action 変数の使用

action 変数は、Listener の実行時にコマンド・ラインで使用します。使用方法は、メッセージ・ハンドラと開始するアクションによって異なります。次の例は、Mobile Link クライアント・アプリケーションの起動に使用する RUN アクション・コマンドの使用例を示します。

```
dblsn ... -l "subject=publish;action='RUN dbmlsync.exe @dbmlsync.txt -n $content'"
```

このメッセージ・ハンドラは、件名のテキストが "publish" と同等の場合にメッセージをフィルタリングします。フィルタリング後に、-n オプションを使用して dbmlsync が実行され、パラメータとして $content action 変数が渡されます。*content* は同期パブリケーションの名前を参照すると仮定して、dbmlsync はパブリケーションを使用して、デバイス・データベースと統合データベースを同期します。

次の例は、action 変数を使用したメッセージのフィルタリングを示します。

```
dblsn ... -l "subject=$content;action='RUN script.bat"
```

このメッセージ・ハンドラは、**subject** のテキストが **content** と同等の場合にメッセージをフィルタリングします。フィルタリング後に、デバイスはカスタム・バッチ・スクリプトを実行します。

参照

- 「Windows 用の Listener アクション・コマンド」 80 ページ
- 「Windows 用の Listener action 変数」 83 ページ
- 「リモート ID によるメッセージのフィルタリング」 24 ページ

ライトウェイト・ポーリング・オプションの設定

メッセージ・ハンドラを使用して、ポーリングを処理できます。ライトウェイト・ポーリング・オプションを使用すると、サーバのロケーション、Notifier 名、ポーリング頻度、ポーリング・キーを指定できます。または、ライトウェイト・ポーリング API を使用してこれらのプロパティを指定することもできます。

ライトウェイト・ポーリング・オプションを指定するには、次の構文を使用して Listener を実行します。

```
dblsn ... -l
   "poll_connect=protocol-options;
   poll_notifier=Notifier-name;
   poll_key=identifier-string;
   poll_every=number-of-seconds;..."
```

1 つのメッセージ・ハンドラには、次のオプションのいずれか 1 つのみを含めることができます。

- **poll_connect**　このオプションは、サーバへの接続に必要なプロトコル・オプションの指定に使用します。または、dblsn -x オプションを使用してデフォルトのプロトコル・オプションを指定することもできます。poll_connect オプションを使用すると、メッセージ・ハンドラのデフォルトのプロトコル・オプションが上書きされます。
- **poll_notifier**　このオプションは、Push 要求を処理するために Mobile Link サーバで使用される Notifier の指定に使用します。Mobile Link サーバでは複数の Notifier を受け入れることができるため、このオプションが必要になります。
- **poll_key**　このオプションは、Notifier に対する Listener の識別に使用します。Mobile Link サーバはこの値を使用して、デバイスを対象とした Push 通知を送信します。典型的なアプリケーションでは、この値をデバイスのリモート ID にしてください。
- **poll_every**　このオプションは、Listener が Notifier をポーリングする頻度の指定に使用します。デフォルトでは、Listener は Mobile Link サーバから自動的にこの値を取得します。この値は、秒単位です。

参照

- 「Push 要求の要件」 10 ページ
- 「Notifier イベントおよびプロパティの設定」 17 ページ
- 「ライトウェイト・ポーラ」 26 ページ
- 「Windows 用の Listener キーワード」 77 ページ
- 「ライトウェイト・ポーリング API」 26 ページ

メッセージ・ハンドラの高度な機能

リモート ID によるメッセージのフィルタリング

リモート ID によってメッセージをフィルタリングするには、-r オプションと $remote_id action 変数を使用します。

SQL Anywhere リモート・データベースを初めて同期すると、データベースの ID を含むリモート ID ファイルが作成されます。このファイルの名前は、データベースと同じで、拡張子 *.rid* が付き、データベースと同じディレクトリに保存されます。Ultra Light データベースの場合はリモート ID ファイルがなく、リモート ID はデータベースから直接抽出されます。

Listener を起動する場合は、dblsn -r オプションを使用してリモート ID ファイルまたは Ultra Light データベースの名前とロケーションを指定し、dblsn -l オプションを使用してメッセージ・ハンドラを作成します。

メッセージ・フィルタには、リモート ID を直接入力できます。ただし、リモート ID はデフォルトでは GUID なので、わかりやすい名前を指定しないと、簡単に覚えることができません。

注意
dblsn コマンド・ラインでは、-r オプションと -l オプションの複数のインスタンスを指定できます。-l オプションで使用される $remote_id action 変数は、通常、その前の -r オプションで指定されています。そのため、-l オプションの前に -r オプションを指定することが重要です。

次の例は、複数のリモート ID の使用方法を示します。ここでは、*business.db* という SQL Anywhere データベースと *personal.udb* という Ultra Light データベースがデバイス上にあることを前提としています。この例で、**ulpersonal** は Ultra Light アプリケーションのウィンドウ・クラス名です。

```
dblsn ... -r "c:¥app¥db¥business.rid"
   -l "subject=$remote_id;action='dbmlsync.exe -k -c dsn=business';"
   -r "c:¥ulapp¥personal.udb"
   -l "subject=$remote_id;action=post dbas_synchronize to ulpersonal;"
```

参照

- 「action 変数」 22 ページ
- 「リモート ID」 『Mobile Link - クライアント管理』
- 「-r オプション」 74 ページ
- 「Windows 用の Listener action 変数」 83 ページ

接続起動同期

Windows デバイスでは、接続の変更時に同期を開始できます。

IP 接続が確立されたり、失われたりすると、デバイスはメッセージ **_IP_CHANGED_** を含む Push 通知を Listener に送信します。デバイスは、Mobile Link サーバへの新しい最適パスを見つけると、メッセージ **_BEST_IP_CHANGED_** を含む Push 通知を Listener に送信します。メッセージ・ハンドラを使用すると、接続に関するこれらの変更を検出し、アクションを開始できます。

接続におけるすべての変更の識別

_IP_CHANGED_ メッセージは、IP 接続が変更されたことを示します。接続の変更は、デバイスが WiFi ネットワークの範囲に入ったり、ユーザが RAS 接続を開始したり、ユーザがデバイスをクレドールに置いたりした場合に発生します。**_IP_CHANGED_** メッセージを参照するには、次の構文を使用して Listener を実行します。

```
dblsn ... -l "message=_IP_CHANGED_;action='...'"
```

次の例は、**_IP_CHANGED_** メッセージの使用方法を示します。メッセージ・ハンドラはメッセージをフィルタリングし、サーバに送信します。接続が失われると、エラーが生成されます。

```
dblsn -l "message=_IP_CHANGED_;
   action='
      SOCKET port=12345;
      sendText=IP changed: $adapters|$network_names;
      recvText=beeperAck;
      timeout=5';
   continue=yes;"
```

Mobile Link サーバへの最適パスの変更の識別

_BEST_IP_CHANGED_ メッセージは、Mobile Link サーバへの最適パスが変更されたことを示します。このメッセージを参照するには、次の構文を使用して Listener を実行します。

```
dblsn ... -x MobiLink-protocol-options -l "message=_BEST_IP_CHANGED_;action='...'"
```

_BEST_IP_CHANGED_ メッセージの実行時に、最善の IP 接続を表すローカルの IP アドレスに置き換える $best_ip action 変数を使用すると、役立つアクションを開始できます。IP 接続が存在しない場合、$best_ip は 0.0.0.0 を返します。

次の例では、**_BEST_IP_CHANGED_** メッセージを使用して、最善の IP 接続が変更されたときに同期を起動しています。接続が失われると、エラーが生成されます。

```
dblsn -x http(host=mlserver.company.com)
  -v2 -m -i 3 -ot dblsn.log
  -l "message=_BEST_IP_CHANGED_;
     action='
        START dbmlsync.exe -ra -c eng=remote;uid=DBA;pwd=sql -n test_pub'"
```

注意
ご使用のアプリケーションで接続起動同期をテストする場合は、Listener を Mobile Link サーバとは別のコンピュータで実行します。

参照

- 「Mobile Link Listener ユーティリティ」 63 ページ
- 「Windows 用の Listener アクション・コマンド」 80 ページ
- 「Windows 用の Listener action 変数」 83 ページ

ライトウェイト・ポーラ

ライトウェイト・ポーラは、指定された時間間隔で 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。ゲートウェイを設定する代わりにライトウェイト・ポーラを使用できます。SYNC ゲートウェイとは異なりサーバへの永続的接続を必要とせず、また、UDP ゲートウェイとは異なり連続的な接続を必要としないため、ライトウェイト・ポーラの使用をおすすめします。

デバイスは、サーバのポーリング時に、ポーリング・キーと Notifier 名を送信します。Mobile Link サーバは、Notifier 名をチェックして、Push 要求のキャッシュをチェックする Notifier を確認します。ポーリング・キーは、ポーリング・キーを使用してデバイスを対象とした Push 要求を検出する Notifier に対するデバイスを識別します。Push 通知は Push 要求の検出後に送信されます。

Listener コマンド・ライン・オプションを使用して、ライトウェイト・ポーラを設定します。または、ライトウェイト・ポーリング API を使用して、ライトウェイト・ポーラをデバイス・アプリケーションに統合します。

注意
ライトウェイト・ポーリングは Palm デバイスではサポートされていません。Push 通知を取得するには、ゲートウェイを設定します。「ゲートウェイと Carrier」 32 ページを参照してください。

参照

- 「ライトウェイト・ポーリング・オプションの設定」 23 ページ
- 「Windows 用の Listener キーワード」 77 ページ

ライトウェイト・ポーリング API

ライトウェイト・ポーリング API は、ご使用のデバイス・アプリケーションに統合できるプログラミング・インタフェースです。ライトウェイト・ポーリング API には、サーバのポーリングに必要なメソッドが含まれています。

必要なファイル

すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。次に、ライトウェイト・ポーリング API のコンパイルに必要なファイルのリストを示します。

ファイル名またはロケーション	説明
SDK¥Lib¥x86¥mllplib.lib	ライトウェイト・ポーリング API ランタイム・ライブラリ
SDK¥Include¥mllplib.h	ライトウェイト・ポーリング API ヘッダ・ファイル

C の SIS_CarDealer_LP2 サンプル・アプリケーションは、*samples-dir¥MobiLink ¥SIS_CarDealer_LP2* 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。

API メソッド

メソッド	説明
「MLLightPoller クラス」 27 ページ	ライトウェイト・ポーラ・オブジェクトを表します。
「MLLPCreatePoller メソッド」 30 ページ	MLLightPoller のインスタンスを作成します。
「MLLPDestroyPoller メソッド」 30 ページ	MLLightPoller のインスタンスを破棄します。

MLLightPoller クラス

ライトウェイト・ポーラ・オブジェクトを表します。

構文

class **MLLightPoller**

参照

- 「auth_status 列挙」 27 ページ
- 「Poll メソッド」 28 ページ
- 「return_code 列挙」 29 ページ
- 「SetConnectInfo メソッド」 30 ページ

例

```
MLLightPoller * poller = MLLCreatePoller();
```

auth_status 列挙

可能な認証ステータス・コードを指定します。

構文

```
typedef enum auth_status {
   AUTH_UNKNOWN,
   AUTH_VALID,
   AUTH_VALID_BUT_EXPIRES_SOON,
   AUTH_EXPIRED,
   AUTH_INVALID,
   AUTH_IN_USE
} auth_status;
```

メンバ

名前	説明
AUTH_EXPIRED	認証の有効期限が切れています。
AUTH_IN_USE	ユーザ名はすでに認証されています。
AUTH_INVALID	認証が無効です。
AUTH_UNKNOWN	認証が不明です。
AUTH_VALID	認証が有効です。
AUTH_VALID_BUT_EXPIRES_SOON	認証は有効ですが、まもなく失効します。

Poll メソッド

Notifier にキャッシュの Push 要求をチェックさせ、サーバをポーリングします。

構文

```
return_code Poll(
   const char * Notifier,
   const char * key,
   char * subject = 0,
   size_t * subjectSize = 0,
   char * content = 0,
   size_t * contentSize = 0
) = 0;
```

パラメータ

- **Notifier**　Notifier の名前。
- **key**　Listener を識別するポーリング・キーの名前。
- **subject**　メッセージの件名を受け取るバッファ (NULL で終了)。
- **subjectSize**　IN：件名バッファのサイズ。

  OUT：受信した件名のサイズ。ゼロが NULL 件名を示す NULL ターミネータを含みます。
- **content**　メッセージの内容を受け取るバッファ (NULL で終了)。
- **contentSize**　IN：内容バッファのサイズ。

  OUT：受信した内容のサイズ。ゼロが NULL 内容を示す NULL ターミネータを含みます。

戻り値

return_code 列挙にリストされているコードの 1 つ。「return_code 列挙」 29 ページを参照してください。

備考

Listener は Notifier に接続し、Notifier がキャッシュで指定されたポーリング・キー宛ての Push 通知をチェックした後に、切断します。

return_code 列挙

可能なリターン・コードを指定します。

構文

```
typedef enum return_code {
   OK,
   BAD_STREAM_NAME,
   BAD_STREAM_PARAM,
   CONNECT_FAILED,
   KEY_NOT_FOUND,
   SUBJECT_OVERFLOW,
   CONTENT_OVERFLOW,
   COMMUNICATION_ERROR,
   AUTH_FAILED,
   NYI
} return_code;
```

メンバ

名前	説明
AUTH_FAILED	Mobile Link サーバへの接続が認証されませんでした。
BAD_STREAM_NAME	認識されないストリーム名。
BAD_STREAM_PARAM	ストリーム・パラメータを解析できませんでした。
COMMUNICATION_ERROR	通信エラーにより失敗しました。
CONNECT_FAILED	Mobile Link サーバに接続できませんでした。
CONTENT_OVERFLOW	内容バッファが小さすぎます。
KEY_NOT_FOUND	指定した Notifier から指定したキーの Push 通知がありません。
NYI	内部でのみ使用されます。
OK	メソッドが正常に実行されました。
SUBJECT_OVERFLOW	件名バッファが小さすぎます。

SetConnectInfo メソッド

Mobile Link クライアントのストリーム・タイプとネットワーク・プロトコル・オプションを設定します。

構文

```
return_code SetConnectInfo(
   const char * streamName,
   const char * streamParams
);
```

パラメータ

- **streamName**　使用するネットワーク・プロトコル。設定可能な値は、**tcpip**、**http**、**https**、または **tls** です。
- **streamParams**　セミコロンで区切ったリスト形式のプロトコル・オプション文字列。オプションの完全なリストについては、「Mobile Link クライアント・ネットワーク・プロトコル・オプション」 『Mobile Link - クライアント管理』を参照してください。

戻り値

return_code 列挙にリストされているコードの 1 つ。「return_code 列挙」 29 ページを参照してください。

例

```
poller_ret = poller->SetConnectInfo("http", "host=localhost;port=80;")
```

MLLPCreatePoller メソッド

MLLightPoller のインスタンスを作成します。

構文

```
MLLightPoller * _entry MLLPCreatePoller( );
```

戻り値

新しい MLLightPoller オブジェクト

参照

- 「MLLPDestroyPoller メソッド」 30 ページ

例

```
poller = MLLPCreatePoller();
```

MLLPDestroyPoller メソッド

MLLightPoller のインスタンスを破棄します。

構文

```
void _entry MLLPDestroyPoller(
   MLLightPoller * poller
);
```

パラメータ

- **poller**　破棄する MLLightPoller。

備考

このメソッドには、NULL MLLightPoller オブジェクトを指定できます。

参照

- 「MLLPCreatePoller メソッド」 30 ページ

例

```
MLLPDestroyPoller(poller);
```

ゲートウェイと Carrier

ライトウェイト・ポーラの代わりにゲートウェイを使用する

ゲートウェイは、Push 通知の送信に使用されるメカニズムで、ライトウェイト・ポーラの代わりに使用でき、継続したネットワーク接続を必要とします。

ゲートウェイのプロパティは、Mobile Link サーバで設定します。1 つの Mobile Link サーバに複数のゲートウェイを設定できます。「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

サポートされているゲートウェイ

Mobile Link サーバでは、次のゲートウェイがサポートされています。

- **SYNC ゲートウェイ** SYNC ゲートウェイは TCP/IP ベースのゲートウェイです。Push 通知は、Mobile Link による同期と同じプロトコルを使用して送信されます。

  デフォルトの SYNC ゲートウェイの名前は、Default-SYNC です。通常、デフォルトのゲートウェイ設定を変更する必要はありません。

  「SYNC ゲートウェイ・プロパティ」 59 ページを参照してください。

- **UDP ゲートウェイ** UDP ゲートウェイは、UDP ゲートウェイを介して Push 通知を送信します。

  デフォルトの UDP ゲートウェイの名前は、Default-UDP です。通常、デフォルトのゲートウェイ設定を変更する必要はありません。Listener は、Push 通知を受信するときにデフォルトで UDP を使用します。

  「UDP ゲートウェイ・プロパティ」 60 ページを参照してください。

- **SMTP ゲートウェイ** SMTP ゲートウェイは、通信業者の電子メールから SMS への変換サービスを使用して Push 通知を送信します。

  デフォルトの SMTP ゲートウェイの名前は、Default-SMTP です。

  「SMTP ゲートウェイ・プロパティ」 58 ページを参照してください。

デバイス・トラッキング・ゲートウェイ

サポートされているゲートウェイ以外に、Push 通知を送信する最適なゲートウェイを自動的に選択するデバイス・トラッキング・ゲートウェイを設定できます。デフォルトのデバイス・トラッキング・ゲートウェイは、Default-DeviceTracker です。ライトウェイト・ポーラを使用しない場合は、このゲートウェイを使用することをおすすめします。デバイス・トラッキングの詳細については、「デバイス・トラッキング・ゲートウェイ」 33 ページを参照してください。

デバイス・トラッキング・ゲートウェイ

デバイス・トラッキングを使用することにより、Mobile Link サーバでは、Push 要求のリモート ID 情報を使用してデバイスを追跡できます。デバイス・トラッキング・ゲートウェイは、自動追跡 IP アドレス、電話番号、公衆無線ネットワーク・プロバイダ ID を利用して SYNC ゲートウェイ、UDP ゲートウェイ、SMTP ゲートウェイを介して Push 通知を配信します。ゲートウェイは、最初に SYNC ゲートウェイを使用してデバイスへの接続を試みます。配信が失敗した場合は、UDP ゲートウェイ、続いて SMTP ゲートウェイが使用されます。この機能は、デバイスのアドレスを変更する場合に便利です。

デバイス・トラッキング・ゲートウェイには、最大で 3 つの従属ゲートウェイ (1 つの SYNC と 1 つの SMTP と 1 つの UDP) を持つことができます。Push 通知は、Listener から送信されたデバイス・トラッキング情報に基づいて、いずれかの従属ゲートウェイへ自動的にルーティングされます。従属ゲートウェイを有効にすると、デバイス・アドレスの変更は、Mobile Link サーバによって自動的に管理されます。アドレスが変更されると、Listener は統合データベースと同期して、ml_device_address システム・テーブル内のトラッキング情報を更新します。

9.0.1 以降のほとんどの Listener は、デバイス・トラッキングをサポートしています。デバイス・トラッキングをサポートしていない Listener を使用している場合、ユーザ自身でトラッキング情報を提供することで、デバイス・トラッキング・ゲートウェイを使用することもできます。「デバイス・トラッキングのサポートの追加」 33 ページを参照してください。

参照

- 「ライトウェイト・ポーラの代わりにゲートウェイを使用する」 32 ページ
- 「Carrier と Carrier 設定」 36 ページ

デバイス・トラッキングのサポートの追加

注意
Palm デバイス用の Listener または Adaptive Server Anywhere 9.0.0 以前で実行されている Listener を使用している場合にのみ、デバイス・トラッキングがサポートされている必要があります。その他すべての Listener では、デバイス・トラッキングがサポートされています。

9.0.0 Listener または Palm デバイス用の Listener のデバイス・トラッキングを手動で設定するためのシステム・プロシージャがいくつかあります。これらのシステム・プロシージャは、統合データベース上の Mobile Link システム・テーブル ml_device、ml_device_address、ml_listening を更新します。

手動で設定するデバイス・トラッキングでは、ネットワーク・アドレス情報を提供しないで Mobile Link ユーザ名によって受信者をアドレス指定できますが、情報が変更されている場合はこれを Mobile Link によって自動的に更新することはできません。ユーザ自身が手動で変更する必要があります。電子メール・アドレスは変更されることが少ないので、この方法は SMTP ゲートウェイで特に便利です。

UDP ゲートウェイでは、再接続のたびに IP アドレスが変更される場合、静的エントリに依存することはできません。この問題を解決するには、IP アドレスではなくホスト名をアドレス指定します。ただし、このソリューションでは、DNS サーバ・テーブルの更新速度が低下するため、

Push 通知が誤配信される可能性があります。システム・プロシージャを設定して、システム・テーブルをプログラムによって更新することもできます。

♦ 9.0.0 Listener または Palm Listener のデバイス・トラッキングを手動で設定するには、次の手順に従います。

1. 各デバイスに対して、ml_device システム・テーブルにデバイス・レコードを追加します。次に例を示します。

```
CALL ml_set_device(
   'myFirstTreo180',
   'MobiLink Listeners for Treo 180 - 9.0.1 Palm Listener',
   '1',
   'not used',
   'y',
   'manually entered by administrator'
);
```

   最初のパラメータである **myFirstTreo180** は、ユーザ定義のユニークなデバイス名です。2 番目のパラメータには、Listener バージョンに関するオプションの注釈が含まれています。3 番目のパラメータは、Listener のバージョンを指定します。SQL Anywhere 9.0.0 Listener の場合は **0**、9.0.0 以降の Palm 用の Listener の場合は **1**、9.0.0 以降の Windows 用の Listener の場合は **2** を使用します。4 番目のパラメータは、オプションのデバイス情報を指定します。5 番目のパラメータは、デバイス・トラッキングを無視するかどうかを指定します。最後のパラメータには、このエントリに関するオプションの注釈が含まれています。

   「ml_set_device システム・プロシージャ」 120 ページを参照してください。

2. 各デバイスに対して、ml_device_address システム・テーブルにアドレス・レコードを追加します。次に例を示します。

```
CALL ml_set_device_address(
   'myFirstTreo180',
   'ROGERS AT&T',
   '55511234567',
   'y',
   'y',
   'manually entered by administrator'
);
```

   最初のパラメータである **myFirstTreo180** は、ユーザ定義のユニークなデバイス名です。2 番目のパラメータはネットワーク・プロバイダ ID で、Carrier プロパティ network_provider_id と一致している必要があります。3 番目のパラメータは、UDP の IP アドレスです。4 番目のパラメータは、Push 通知の送信用にこのエントリをアクティブにするかどうかを設定します。5 番目のパラメータは、デバイス・トラッキングを無視するかどうかを指定します。最後のパラメータには、このエントリに関するオプションの注釈が含まれています。

3. 各リモート・データベースに対して、追加した各デバイスの ml_listening システム・テーブルに受信者レコードを追加します。これは、デバイスを Mobile Link ユーザ名にマッピングします。次に例を示します。

```
CALL ml_set_listening(
   'myULDB',
   'myFirstTreo180',
   'y',
   'y',
```



```
    'manually entered by administrator'
);
```

最初のパラメータは Mobile Link ユーザ名です。2 番目のパラメータは、ユーザ定義のユニークなデバイス名です。3 番目のパラメータは、デバイス・トラッキングのアドレス指定用にこのエントリをアクティブにするかどうかを設定します。4 番目のパラメータは、デバイス・トラッキングを無視するかどうかを指定します。最後のパラメータには、このエントリに関するオプションの注釈が含まれています。

参照

- 「ml_set_listening システム・プロシージャ」 124 ページ
- 「ml_set_device_address システム・プロシージャ」 122 ページ
- 「デバイス・トラッキング・ゲートウェイ」 33 ページ
- 「Carrier プロパティ」 62 ページ

デバイス・トラッキング・ゲートウェイ設定のクイック・ガイド

♦ **デバイス・トラッキングを設定するには、次の手順に従います。**

1. 必要に応じて、SYNC ゲートウェイ、UDP ゲートウェイ、または SMTP ゲートウェイを設定します。

   Mobile Link サーバを起動すると、これらのゲートウェイがデフォルト設定を使用して設定されています。プロパティの設定や独自のゲートウェイの作成の詳細については、「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

   注意
   SMTP ゲートウェイの場合、Carrier の設定が必要です。「Carrier と Carrier 設定」 36 ページを参照してください。

2. 次の条件に従って新しい Notifier を作成し、request_cursor イベントを設定します。
   - ゲートウェイ名は、使用するデバイス・トラッキング・ゲートウェイの名前にしてください。デフォルトのゲートウェイ名は、Default-DeviceTracker です。この名前は、結果セットの最初のカラムで指定されています。
   - アドレス名には、デバイスのリモート ID を設定してください。dblsn -t+ オプションを使用して、Mobile Link サーバにリモート ID を登録します。この名前は、結果セットの 4 番目のカラムで指定されています。

   request_cursor イベントの設定の詳細については、「request_cursor イベント」 47 ページを参照してください。

3. ml_user システム・テーブルに Listener 名を追加します。

   デフォルトの Listener 名は、*device-name*-**dblsn** (*device_name* はデバイス名) です。

   Listener を実行して、Listener メッセージ・ウィンドウ内のデバイス名を確認します。または、dblsn -e オプションを使用してデバイス名を設定するか、dblsn -u オプションを使用して別の

Listener 名を設定できます。「-e オプション」 69 ページと「-u オプション」 75 ページを参照してください。

Mobile Link ユーザの登録の詳細については、「Mobile Link ユーザの作成と登録」 『Mobile Link - クライアント管理』を参照してください。

4. 必要なオプションを指定して Listener を起動します。Listener の起動の詳細については、「Mobile Link Listener ユーティリティ」 63 ページを参照してください。

Carrier と Carrier 設定

Notifier では有効な電子メール・アドレスを作成する必要があるため、SMTP ゲートウェイを使用して Push 通知を送信するには、無線通信事業者を設定してください。また、従属 SMTP ゲートウェイが有効なデバイス・トラッキング・ゲートウェイを使用する場合にも、無線通信業者を設定してください。

ネットワーク・プロバイダ ID や SMS 電子メールのプレフィクスなど、Carrier のプロパティは、Mobile Link サーバで設定します。複数の Carrier サービスに対応するには、Mobile Link サーバで複数の Carrier を設定します。「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

送信者の構文

Push 通知は、Listener で受信され、メッセージ・フィルタリング用に前処理されると、複数のキーワードに分割されます。**message** の **sender** キーワードは電子メール・アドレスです。この電子メール・アドレスは、デバイスで生成され、無線通信事業者によって異なります。メッセージの前処理の詳細については、「メッセージ構文」 128 ページを参照してください。

sender 構文は、次のフォーマットになります。

```
sender = sms_email_user_prefix phone-number@sms_email_domain
```

注意
sms_email_user_prefix と *phone-number* の間には、スペースを入れません。

sms_email_user_prefix 値と *sms_email_domain* 値は Carrier プロパティです。Mobile Link サーバで設定してください。*phone-number* 値は、ml_device_address システム・テーブルの address カラムから取得されます。

送信者の構文を指定するには、Carrier サービスを使用するデバイスで Listener を実行します。メッセージのロギングを有効にし、dblsn -m and -v オプションを使用して冗長レベルを 2 に設定します。Listener のロード後に、メッセージ・ログを確認します。

参照

- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ
- 「Carrier プロパティ」 62 ページ

サーバ起動同期の Mobile Link サーバ設定

目次

サーバ側設定は、Notifier プロパティ、ゲートウェイ・プロパティ、Carrier プロパティ、Notifier イベントで構成されます。これらの設定を行うには、次のいずれかの方法を使用します。

- Sybase Central
- Notifier 設定ファイル
- ml_add_property システム・プロシージャ

Sybase Central と ml_add_property システム・プロシージャを使用する方法では、ml_property システム・テーブルにイベントと設定が追加されます。

注意
サーバ側設定を変更しても、Mobile Link サーバの稼働中は変更が有効になりません。新しい設定を適用するには、Mobile Link サーバを停止して再度起動する必要があります。

ml_property システム・テーブルでサーバ側設定を行ってあるときに Notifier 設定ファイルを使用する場合は、システム・テーブル設定が必ず最初にロードされ、その次にファイル設定がロードされます。Notifier 設定ファイルによって既存のサーバ側設定が上書きされますが、この変更は統合データベースに永続的には適用されません。

ml_add_property システム・プロシージャを使用したサーバ側設定の実行

SQL Anywhere 統合データベースのサーバ側設定を行うには、ml_add_property システム・プロシージャを使用します。これらのプロパティとイベントは、Interactive SQL を使用して設定できます。

共通プロパティ構文

```
CALL ml_add_property('SIS', '', 'Property', Value);
```

Notifier プロパティとイベントの構文

```
CALL ml_add_property('SIS', 'Notifier(NotifierName)', 'Event-or-Property', Value);
```

ゲートウェイ・プロパティ構文

```
CALL ml_add_property('SIS', 'DeviceTracker(DeviceTrackerName)', 'Property', Value);
```

```
CALL ml_add_property('SIS', 'SMTP(SMTPName)', 'Property', Value);
```

```
CALL ml_add_property('SIS', 'UDP(UDPName)', 'Property', Value);
```

```
CALL ml_add_property('SIS', 'SYNC(SYNCName)', 'Property', Value);
```

Carrier プロパティ構文

```
CALL ml_add_property('SIS', 'Carrier(CarrierName)', 'Property', Value);
```

詳細については、「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。

Sybase Central を使用したサーバ側設定の実行

Sybase Central には、プロパティとイベントを変更するためのグラフィカル・ユーザ・インタフェースが用意されています。Sybase Central を使用すると、複数の Notifier、ゲートウェイ、Carrier を設定できます。Sybase Central を使用してサーバ側設定を実行すると、mlsrv11 -notifier オプションを使用するときに、コマンド・ラインで Notifier 設定ファイルを指定する必要がありません。

♦ Sybase Central を使用して Notifier、ゲートウェイ、Carrier を設定するには、次の手順に従います。

1. Mobile Link プラグインは、Sybase Central から統合データベースに接続する場合に使用します。
2. **[モード]** - **[管理]** を選択します。
3. **[通知]** をクリックします。

   右ウィンドウ枠に、使用可能な Notifier、ゲートウェイ、Carrier がすべて表示されます。
4. 新しい Notifier、ゲートウェイ、Carrier を作成します。
   - 新しい Notifier を作成するには、右ウィンドウ枠の **[Notifier]** タブをクリックし、**[ファイル]** - **[新規]** - **[Notifier]** を選択します。
   - 新しいゲートウェイを作成するには、右ウィンドウ枠の **[ゲートウェイ]** タブをクリックし、**[ファイル]** - **[新規]** - **[ゲートウェイ]** を選択します。
   - 新しい Carrier を作成するには、右ウィンドウ枠の **[Carrier]** タブをクリックし、**[ファイル]** - **[新規]** - **[Carrier]** を選択します。
5. 設定する Notifier、ゲートウェイ、または Carrier を選択します。
   - Notifier プロパティまたはイベントを設定するには、右ウィンドウ枠の **[Notifier]** タブをクリックし、設定する Notifier を選択します。
   - ゲートウェイ・プロパティを設定するには、右ウィンドウ枠の **[ゲートウェイ]** タブをクリックし、設定するゲートウェイを選択します。
   - Carrier プロパティを設定するには、右ウィンドウ枠の **[Carrier]** タブをクリックし、設定する Carrier を選択します。

   [ファイル] - **[プロパティ]** を選択します。

   選択した Notifier、ゲートウェイ、または Carrier に適用可能なすべての設定を調整できるウィンドウが表示されます。
6. **[OK]** をクリックします。

Notifier 設定ファイルからのサーバ側設定のインポート

サーバ側設定は、Notifier 設定ファイルから ml_property_table にインポートできます。

♦ **Notifier 設定ファイルからサーバ側設定をインポートするには、次の手順に従います。**

1. Mobile Link プラグインは、Sybase Central から統合データベースに接続する場合に使用します。
2. **[モード]** - **[管理]** を選択します。
3. **[通知]** をクリックします。
4. **[ファイル]** - **[インポートの設定]** を選択し、ウィザードの指示に従います。

Notifier 設定ファイルへのサーバ側設定のエクスポート

サーバ側設定は、ml_property テーブルから Notifier 設定ファイルにエクスポートできます。設定をエクスポートすると、?複数のバージョンのサーバ側設定を作成し、mlsrc11 -notifier オプションを使用して異なるバージョンをロードできます。

♦ **Notifier 設定ファイルにサーバ側設定をエクスポートするには、次の手順に従います。**

1. Mobile Link プラグインは、Sybase Central から統合データベースに接続する場合に使用します。
2. **[モード]** - **[管理]** を選択します。
3. **[通知]** をクリックします。
4. **[ファイル]** - **[インポートの設定]** を選択し、ウィザードの指示に従います。

参照

- 「モデル・モードでのサーバ起動同期の設定」 『Mobile Link - クイック・スタート』

Notifier 設定ファイルを使用したサーバ側設定の実行

サーバ側設定は、Notifier 設定ファイルに格納できます。このファイルを使用すると、複数の Notifier、ゲートウェイ、Carrier を設定できます。

Notifier 設定ファイルの作成と設定

Notifier 設定ファイルは、テキスト・エディタを使用して作成したり、Sybase Central からエクスポートしたプロパティとイベントの設定から生成したりできます。「Sybase Central を使用したサーバ側設定の実行」 39 ページを参照してください。

標準的な Notifier 設定ファイルのレイアウトを表示するには、*samples-dir¥MobiLink ¥template.Notifier* テンプレート・ファイルを開きます。ここで、*samples-dir* は SQL Anywhere 11 サンプル・ディレクトリのロケーションです。このテンプレート・ファイルでは、サーバ側プロパティとイベントを設定するための例を提供します。

必要な設定が完了したら Notifier 設定ファイルを保存し、サーバ側プロパティとイベントを Mobile Link サーバにロードします。

共通プロパティ構文

```
Property = Value
```

Notifier イベント構文

```
Notifier(NotifierName).Event = ¥
# Replace this text with SQL script.      ¥
# Be sure to put a backslash (¥) at       ¥
# the end of every line of code          ¥
# if your event requires multiple        ¥
# lines of text.
```

Notifier プロパティ構文

```
Notifier(NotifierName).Property = Value
```

ゲートウェイ・プロパティ構文

```
# For Device tracking gateways:
DeviceTracker(DeviceTrackerName).Property = Value
# For SMTP gateways:
SMTP(SMTPName).Property = Value
# For SYNC gateways:
SYNC(SYNCName).Property = Value
# For UDP gateways:
UDP(UDPName).Property = Value
```

Carrier プロパティ構文

```
Carrier(CarrierName).Property = Value
```

Notifier 設定ファイルのロード

Notifier 設定ファイルを Mobile Link サーバにロードするには、コマンド・ラインから -notifier オプションを指定して mlsrv11 を実行します。たとえば、CarDealer.Notifier 設定ファイルに定義されたサーバ側設定を使用するには、次のコマンドを実行します。

```
mlsrv11 ... -notifier "c:¥CarDealer.Notifier"
```


ファイルを指定しない場合は、デフォルトで *config.Notifier* ファイルがロードされます。

mlsrv11 -notifier オプションの詳細については、「-notifier オプション」『Mobile Link - サーバ管理』を参照してください。

注意
デフォルトの SYNC ゲートウェイを使用する場合、サーバ側設定を Notifier 設定ファイルには保存できません。別の方法を使用して、この設定を ml_property システム・テーブルに格納する必要があります。「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

エスケープ・シーケンスの使用

円記号 (¥) はエスケープ文字です。Notifier 設定ファイルで使用できる一般的なエスケープ・シーケンスのリストは、次のとおりです。

エスケープ・シーケンス	説明
¥b	バックスペース
¥t	タブ
¥n	改行
¥r	キャリッジ・リターン
¥"	二重引用符 (")
¥'	一重引用符 (')
¥¥	円記号 (¥)
¥e	エスケープ

Unicode のエスケープ・シーケンス形式は ¥uXXXX、ASCII のエスケープ・シーケンス形式は ¥xXX です。ここで、各 X は 16 進数字を表します。

複数行のテキストが必要なプロパティまたはイベントを編集する場合は、1 つの円記号 (¥) を各行の最後に追加します。

Notifier イベント

イベントは、Notifier が Listener をポーリングするたびに起動されます。イベントが起動すると、そのイベントに対応する SQL スクリプトが実行されます。SQL スクリプトは、この項で示すいずれの Notifier イベントに組み込むことができます。スクリプトの実行は任意ですが、request_cursor ポーリング・イベントを記述する必要があります。

Notifier イベントは、ポーリング・イベント、接続イベント、非同期イベントの 3 つに分類されます。ポーリング・イベントは、Notifier が統合データベースをチェックするたびに起動し、begin_poll イベントと end_poll イベントの間に発生するすべてのイベントが含まれます。接続イベントは、Notifier のデータベース接続中に起動します。非同期イベントは、同期処理中の任意の時点で起動する可能性があります。

特に指定しないかぎり、Notifier イベントはおすすめする方法のいずれかを使用して設定できます。Notifier イベントの設定の詳細については、「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

Listener が Notifier をポーリングすると、これらのイベントが次の順序で起動します。

```
Fire begin_connection event
For each poll (
   Fire begin_poll event
      Fire shutdown_query event
      Fire request_cursor event
      For all requests expired before required confirmation (
         Fire error_handler event
      )
      Fire request_delete event
   Fire end_poll event
)
Fire end_connection event
```

ポーリング・イベント

ポーリング・イベントは、Notifier が統合データベースをチェックするたびに起動する Notifier イベントに分類されます。これらのイベントには、begin_poll イベントと end_poll イベントの間に発生するすべてのイベントが含まれます。

begin_poll イベント

このポーリング・イベントは SQL スクリプトを受け入れ、Notifier が統合データベースをチェックして Push 要求があるかどうかを確認する前に起動されます。デフォルトでは、値は NULL であるため、このイベントは起動されません。

参照

- 「Push 要求」 10 ページ
- 「Notifier イベント」 43 ページ
- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ

例

この例では、Notifier A という名前の Notifier で使用する Push 要求を作成します。SQL 文を使用して、PushRequest という名前のテーブルにローを挿入します。このテーブルの各ローは、1 つのアドレスに送信するメッセージを表しています。WHERE 句によって、PushRequest テーブルに挿入される Push 要求が決まります。

ml_add_property システム・プロシージャを SQL Anywhere 統合データベースで使用するには、次のコマンドを実行します。

```
ml_add_property(
   'SIS',
   'Notifier(Notifier A)',
   'begin_connection',
   'INSERT INTO PushRequest
      (gateway, mluser, subject, content)
      SELECT "MyGateway", DISTINCT mluser, "sync",
         stream_param
         FROM MLUserExtra, mluser_union, Dealer
         WHERE MLUserExtra.mluser = mluser_union.name
         AND (push_sync_status = "waiting for request"
            OR datediff( hour, last_status_change, now() ) > 12 )
         AND ( mluser_union.publication_name is NULL
            OR mluser_union.publication_name ="FullSync" )
         AND Dealer.last_modified > mluser_union.last_sync_time'
);
```

end_poll イベント

このポーリング・イベントは SQL スクリプトを受け入れ、Notifier が統合データベースをチェックして Push 要求があるかどうかを確認した後に起動されます。デフォルトでは、値は NULL であるため、このイベントは起動されません。

このイベントを使用すると、テーブルのクリーンアップを実行したり、ポーリングの結果をログに記録できます。

参照

- 「Notifier イベント」 43 ページ
- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ

error_handler イベント

転送に失敗した場合や転送が確認されなかった場合を示すには、このイベントを設定します。たとえば、転送に失敗した場合、このイベントを使用すると、監査テーブルにローを挿入したり、Push 通知を送信したりできます。

次の表に、error_handler イベントを使用して取得できるパラメータの詳細を示します。

スクリプト・パラメータ	型	説明
request_option (out)	Integer	エラー・ハンドラが戻った後に Notifier が Push 要求に対して実行する処理を制御します。出力は、次のいずれかの値になります。 ● **0**：エラー・コードに基づいてデフォルト・アクションを実行し、エラーを記録します。 ● **1**：何もしません。 ● **2**：request_delete イベントを実行します。 ● **3**：セカンダリ・ゲートウェイへの配信を試行します。
error_code (in)	Integer	エラー・コードには、次のいずれかの値を使用します。 ● **-1**：確認が成功したあと、要求はタイムアウトされました。 ● **-8**：配信試行中にエラーが発生しました。
request_id (in)	Integer	要求を識別します。
gateway (in)	varchar	Push 要求に関連付けられているゲートウェイを指定します。
address (in)	varchar	Push 要求に関連付けられているアドレスを指定します。
subject (in)	varchar	Push 要求に関連付けられている件名を指定します。
content (in)	varchar	Push 要求に関連付けられている内容を指定します。

注意
このイベントにはシステム・プロシージャの使用が必要です。Sybase Central を使用して、このイベントを直接設定することはできません。「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

参照

- 「Notifier イベント」 43 ページ
- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ

例

次の例では、CustomError というテーブルを作成し、CustomErrorHandler というストアド・プロシージャを使用してエラーをテーブルに記録します。出力パラメータ Notifier_opcode は常に 0 で、デフォルトの Notifier 処理が使用されます。

```
CREATE TABLE CustomError(
   error_code  integer,
   request_id  integer,
   gateway  varchar(255),
   address  varchar(255),
   subject  varchar(255),
   content  varchar(255),
   occurAt  timestamp not null default timestamp
);

CREATE PROCEDURE CustomErrorHandler(
   out @Notifier_opcode integer,
   in @error_code  integer,
   in @request_id  integer,
   in @gateway   varchar(255),
   in @address   varchar(255),
   in @subject   varchar(255),
   in @content   varchar(255)
)

BEGIN
   INSERT INTO CustomError(
      error_code,
      request_id,
      gateway,
      address,
      subject,
      content)
   VALUES(
      @error_code,
      @request_id,
      @gateway,
      @address,
      @subject,
      @content
   );
   SET @Notifier_opcode = 0;
END
```

ml_add_property システム・プロシージャを SQL Anywhere 統合データベースで使用するには、次のコマンドを実行します。

```
call ml_add_property(
   'SIS',
   'Notifier(myNotifier)',
   'error_handler',
   'call CustomErrorHandler(?, ?, ?, ?, ?, ?, ?)');
```

または、Notifier 設定ファイルに次の行を追加しても、このイベントを起動できます。

```
Notifier(myNotifier).error_handler = call CustomErrorHandler(?, ?, ?, ?, ?, ?, ?)
```

mlsrv11 -notifier オプションを使用してファイルを実行します。Notifier 設定ファイルを設定する方法については、「Notifier 設定ファイルを使用したサーバ側設定の実行」 41 ページを参照してください。

request_cursor イベント

このポーリング・イベントは SQL スクリプトを受け入れ、Push 要求を検出すると起動されます。このイベントは設定が必要です。

ライトウェイト・ポーラを使用する場合の Push 要求のフェッチ (推奨)

このイベントで結果セットに最大 3 つのカラムが含まれる場合、Notifier はサーバとデバイスの間に永続的な接続がないことと、デバイスが Notifier をポーリングしてから Push 通知を送信する必要があることを確認します。Notifier は、結果セットをキャッシュしてから Push 通知を送信します。Mobile Link サーバでは、ポーリング・キーによってデバイスを識別します。ポーリング・キーは、デバイスが Notifier をポーリングするたびにデバイスが送信します。

このイベントの結果セットには、次のカラムが指定した順序で含まれている必要があります。

- ポーリング・キー
- 件名 (オプション)
- 内容 (オプション)

ゲートウェイを使用する場合の Push 通知のフェッチ

このイベントで結果セットに 3 つを超えるカラムが含まれる場合、Notifier はサーバとデバイスの間に永続的な接続が存在することを確認し、Push 要求が検出されたときにゲートウェイを使用して Push 通知を送信します。

このイベントの結果セットには、次のカラムが指定した順序で含まれている必要があります。

- 要求 ID (オプション)
- ゲートウェイ
- 件名
- 内容
- アドレス
- 再送間隔 (オプション)
- 存続期間 (オプション)

参照

- 「Push 要求の要件」 10 ページ
- 「Notifier イベント」 43 ページ
- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ

例

次の例では、ml_add_property システム・プロシージャを使用して、Simple という名前のカスタム Notifier 用の request_cursor イベント・スクリプトを作成します。SELECT 文では、Notifier に PushRequest という名前のテーブルから Push 要求を検出するように指示します。

```
CALL ml_add_property('SIS', 'Notifier(Simple)', 'request_cursor',
   'SELECT poll_key,
      subject,
      content
   FROM PushRequest'
);
```

スクリプトに WHERE 句を追加して、送信済みの要求をフィルタすることをおすすめします。たとえば、要求を挿入した時刻を追跡する Push 要求カラムを追加して、このイベントで WHERE 句を使用すると、ユーザが最後に同期を行った時刻よりも前に挿入された要求をフィルタできます。

request_delete イベント

このポーリング・イベントは SQL スクリプトを受け入れ、Push 要求の削除の必要性が検出されるとクリーンアップ処理を実行するために起動されます。このイベントではパラメータとして要求 ID を受け入れ、要求 ID ごとに実行されます。request_cursor イベントには、request_delete イベントを使用するための要求 ID カラムが含まれている必要があります。指定したパラメータまたは疑問符 (?) を使用すると、要求 ID を参照できます。別のプロセスや end_poll イベントなどのイベントにクリーンアップ処理を割り当ててある場合、このイベントはオプションです。

Notifier では、DELETE 文を使用して、次の形式の Push 要求を削除できます。

- **暗黙的に除外**　この Push 要求は、以前発生したが、request_cursor イベントから取得された現在の要求セットにはありません。
- **確認済み**　配信が確認された Push 要求です。
- **失効**　この Push 要求は、resend 属性と現在の時刻に基づき、有効期限が切れています。resend 属性のない要求は、次回の要求に表示された場合でも、有効期限が切れていると見なされます。

request_delete イベントを使用すると、有効期限が切れた要求または暗黙的に除外された要求を削除できなくなります。たとえば、*samples-dir¥MobiLink¥SIS_CarDealer* ディレクトリの CarDealer サンプルでは、request_delete イベントを使用して、PushRequest テーブルのステータス・フィールドを 'processed' に設定しています。

```
UPDATE PushRequest SET status='processed' WHERE req_id = ?
```

このサンプルの begin_poll イベントでは、最後の同期時間を利用して、処理済みの Push 要求を削除する前にリモート・デバイスが最新状態であるかどうかをチェックしています。

参照

- 「Notifier イベント」 43 ページ
- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ

shutdown_query イベント

このポーリング・イベントは SQL スクリプトを受け入れ、begin_poll イベントの後に起動されます。戻り値は Notifier の停止ステータスを示します。デフォルトでは、値は NULL であるため、このイベントは起動されません。

Notifier を停止するには、'yes' を返すように SQL スクリプトを設定します。それ以外の場合は、'no' を返すように設定します。Notifier が停止した場合、end_poll イベントは起動されません。

停止ステータスをテーブルに格納している場合は、end_connection イベントを使用してステータスをリセットします。

参照

- 「end_connection イベント」 50 ページ
- 「Notifier イベント」 43 ページ
- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ

例

次の例では、ml_add_property システム・プロシージャを使用して、Simple という名前のカスタム Notifier 用の shutdown_query イベント・スクリプトを作成します。SELECT 文によって、tooManyNotifierErrors メソッドから true が返された場合に停止するよう Notifier に通知しています。

```
CALL ml_add_property('SIS', 'Notifier(Simple)', 'shutdown_query',
   'SELECT
      IF tooManyNotifierErrors() THEN
         'yes'
      ELSE
         'no'
      ENDIF'
);
```

接続イベント

接続イベントは、Notifier データベースの接続時に起動する Notifier イベントに分類されます。

begin_connection イベント

このイベントは SQL スクリプトを受け入れ、Notifier が統合データベースに接続した後、Push 要求をチェックする前に起動されます。デフォルトでは、値は NULL であるため、このイベントは起動されません。

このイベントを使用すると、テンポラリ・テーブルまたは変数を作成できます。このイベントを使用して、独立性レベルを変更しないでください。独立性レベルを指定するには、isolation プロパティを使用します。「Notifier プロパティ」 55 ページを参照してください。

統合データベースへの接続が失われると、Notifier は再接続した直後にこのイベントを再実行します。

参照

- 「Notifier イベント」 43 ページ
- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ

end_connection イベント

このイベントは SQL スクリプトを受け入れ、Notifier が統合データベースから切断する直前に起動されます。デフォルトでは、値は NULL であるため、このイベントは起動されません。

このイベントを使用すると、SQL 変数やテンポラリ・テーブルなどの一時的な記憶領域をクリーンアップできます。

参照

- 「Notifier イベント」 43 ページ
- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ

非同期イベント

非同期イベントは、同期処理中の任意の時点で起動する可能性のある Notifier イベントに分類されます。

confirmation_handler イベント

Listener がアップロードした配信確認情報を処理するには、このイベントを設定します。ステータス・パラメータが 0 を返す場合、request_id で識別された Push 要求は、remote_device パラメータで識別された Listener によって正常に受信されています。

request_option パラメータを使用すると、配信確認への応答としてアクションを開始できます。request_option が 0 の場合、confirmation_handler イベントはデフォルトのアクションを開始します。つまり、request_delete イベントが実行されて、元の Push 要求が削除されます。配信確認を送信するデバイスが request_id で識別されたデバイスと一致しない場合、デフォルトのアクションでは、元の Push 要求がセカンダリ・ゲートウェイを使用して送信されます。

注意
Listener が配信確認情報をアップロードできるようにするには、dblsn -x オプションを使用します。配信確認は必要だが IP 追跡は不要な場合は、dblsn -ni オプションを使用します。「Windows 用の Listener オプション」 65 ページを参照してください。

注意
このイベントにはシステム・プロシージャの使用が必要です。Sybase Central を使用する方法では、このイベントを直接設定できません。「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

confirmation_handler イベントを使用して、次のパラメータを取得できます。

スクリプト・パラメータ	型	説明
request_option (out)	Integer	ハンドラが戻った後に Notifier が要求に対して実行する処理を制御します。次の値が返されます。 ● **0**：status パラメータの値に基づいてデフォルトの Notifier アクションを実行します。応答デバイスがターゲット・デバイスであることを status が示している場合、Notifier は要求を削除します。そうでない場合は、Notifier はセカンダリ・ゲートウェイへの配信を試行します。 ● **1**：何もしません。 ● **2**：Notifier.request_delete を実行します。 ● **3**：セカンダリ・ゲートウェイへの配信を試行します。
status (in)	Integer	状況の概要。ステータスは、開発時に不適切なフィルタやハンドラ属性などの問題の識別に使用できます。次の値が返されます。 ● **0**：受信され、確認されました。 ● **-2**：正しい応答相手でしたが、メッセージは拒否されました。 ● **-3**：正しい応答相手で、メッセージは受け入れられましたが、アクションは失敗しました。 ● **-4**：間違った応答相手でしたが、メッセージは受け入れられました。 ● **-5**：間違った応答相手で、メッセージは拒否されました。 ● **-6**：間違った応答相手でした。メッセージは受け入れられ、アクションは正常に終了しました。 ● **-7**：間違った応答相手でした。メッセージは受け入れられましたが、アクションは失敗しました。
request_id (in)	Integer	要求 ID。request_cursor イベントには、confirmation_handler イベントを使用するための要求 ID カラムが含まれている必要があります。
remote_code (in)	Integer	リモート Listener からレポートされた概要です。次の値が返されます。 ● **1**：メッセージは受け入れられました。 ● **2**：メッセージは拒否されました。 ● **3**：メッセージは受け入れられ、アクションは正常に終了しました。 ● **4**：メッセージは受け入れられ、アクションは失敗しました。
remote_device (in)	varchar	応答 Listener のデバイス名です。
remote_mluser (in)	varchar	応答 Listener の Mobile Link ユーザ名です。

スクリプト・パラメータ	型	説明
remote_action_return (in)	varchar	リモート・アクションのリターン・コードです。
remote_action (in)	varchar	アクション・コマンド用に予約済みです。
gateway (in)	varchar	要求に関連付けられているゲートウェイです。
address (in)	varchar	要求に関連付けられているアドレスです。
subject (in)	varchar	要求に関連付けられている件名です。
content (in)	varchar	要求に関連付けられている内容です。

参照

- 「Notifier イベント」 43 ページ
- 「サーバ起動同期の Mobile Link サーバ設定」 37 ページ
- 「ゲートウェイ・プロパティ」 57 ページ

例

次の例では、CustomConfirmation というテーブルを作成し、CustomConfirmationHandler という名前のストアド・プロシージャを使用して確認をログ記録します。出力パラメータ request_option は常に 0 に設定され、デフォルト Notifier 処理が使用されます。

```
CREATE TABLE CustomConfirmation(
   error_code   integer,
   request_id   integer,
   remote_code   integer,
   remote_device  varchar(128),
   remote_mluser  varchar(128),
   remote_action_return varchar(128),
   remote_action  varchar(128),
   gateway   varchar(255),
   address   varchar(255),
   subject   varchar(255),
   content   varchar(255),
   occurAt   timestamp not null default timestamp
)

CREATE PROCEDURE CustomConfirmationHandler(
   out @request_option integer,
   in @error_code  integer,
   in @request_id  integer,
   in @remote_code  integer,
   in @remote_device  varchar(128),
   in @remote_mluser  varchar(128),
```



```
    in @remote_action_return varchar(128),
    in @remote_action  varchar(128),
    in @gateway   varchar(255),
    in @address   varchar(255),
    in @subject   varchar(255),
    in @content   varchar(255)
)

BEGIN
   INSERT INTO CustomConfirmation(
      error_code,
      request_id,
      remote_code,
      remote_device,
      remote_mluser,
      remote_action_return,
      remote_action,
      gateway,
      address,
      subject,
      content)
   VALUES (
      @error_code,
      @request_id,
      @remote_code,
      @remote_device,
      @remote_mluser,
      @remote_action_return,
      @remote_action,
      @gateway,
      @address,
      @subject,
      @content
   );
   SET @request_option = 0;
END
```

ml_add_property システム・プロシージャを SQL Anywhere 統合データベースで使用するには、次のコマンドを実行します。

```
call ml_add_property(
   'SIS',
   'Notifier(myNotifier)',
   'confirmation_handler',
   'call CustomConfirmation(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)');
```

または、Notifier 設定ファイルに次の行を追加して、このイベントを呼び出すこともできます。

```
Notifier(myNotifier).confirmation_handler = call CustomConfirmation(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

mlsrv11 -notifier オプションを使用してファイルを実行します。Notifier 設定ファイルを設定する方法については、「Notifier 設定ファイルを使用したサーバ側設定の実行」 41 ページを参照してください。

共通プロパティ

共通プロパティは、Notifier、ゲートウェイ、Carrier で共有されます。共通プロパティは、すべてオプションです。これらのプロパティの設定については、「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

プロパティ	値	説明
verbosity	{ **0** | **1** | **2** | **3** }	Notifier、ゲートウェイ、Carrier の冗長性レベルを指定します。次の値を使用できます。 ● **0**：トレーシングなし ● **1**：起動、シャットダウン、プロパティのトレーシング ● **2**：通知を表示 ● **3**：完全レベルのトレーシング デフォルト値は **0** です。

Notifier プロパティ

Notifier プロパティを使用すると、Notifier の動作を変更できます。Notifier のプロパティは、すべてオプションです。これらのプロパティの設定については、「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

プロパティ	値	説明
connect_string	*connection_string*	データベースへの接続に使用されるデフォルトの接続動作を上書きします。デフォルト値は **ianywhere.ml.script.ServerContext** です。この値では、mlsrv11 コマンド・ラインで指定された接続文字列を使用します。 別のデータベースに接続するときに通知ロジックとデータを同期データから分離するのに便利です。ほとんどの展開ではこのプロパティを設定しません。
enable	{ **yes** | **no** }	Notifier を有効にするかどうかを指定します。-notifier mlsrv11 オプションを実行すると、有効な Notifier がすべて起動します。
gui	{ **yes** | **no** }	Notifier の動作中に Notifier ウィンドウを表示するかどうかを指定します。デフォルト値は **yes** です。 この Notifier ウィンドウを使用すると、ポーリング間隔を一時的に変更したり、すぐにポーリングを実行したりできます。また、Mobile Link サーバを停止せずに Notifier を停止するために使用することも可能です一度停止すると、Mobile Link サーバを停止して再度起動しないと、Notifier を再度起動できません。

プロパティ	値	説明
isolation	{ **0** | **1** | **2** | **3** }	Notifier のデータベース接続の独立性レベルを指定します。次の値を使用できます。 ● **0**：コミットされない読み出し ● **1**：コミットされた読み出し ● **2**：繰り返し可能読み出し ● **3**：直列化可能 デフォルト値は **1** です。レベルが高くなると競合が増加しますが、パフォーマンスが逆に低下することがあります。独立性レベルを **0** に設定すると、コミットされていないデータ (ロールバックする可能性のあるデータ) を読み出すことができます。
poll_every	*number*{ **s** | **m** | **h** }	確認がタイムアウトになるまでに待機する時間を指定します。次に、使用可能な時間単位のリストを示します。 ● **s**：秒単位。 ● **m**：分単位。 ● **h**：時間単位。 デフォルト値は **1m** です。時間単位は、*HH***h** *MM***m** *SS***s** のフォーマットで組み合わせることができます。時間単位が指定されていない場合、時間は秒単位で測定されます。
shared_database_connection	{ **yes** | **no** }	Notifier がデータベース接続を共有するかどうかを指定します。デフォルト値は **no** です。Notifier が接続を共有できるのは、その独立性レベルが同じ場合のみです。 パフォーマンスに悪影響を出さずにリソースを節約するには、**yes** を指定します。状況によっては、接続を共有できないことがあります。たとえば、アプリケーションが Notifier 間でユニークでない SQL 変数名を使用している場合などが該当します。

ゲートウェイ・プロパティ

デフォルトでは、Mobile Link サーバを起動すると、あらかじめ定義されている 4 つのゲートウェイが作成されます。これらのゲートウェイは、統合データベース用の Mobile Link 設定スクリプトを実行したときにインストールされます。デフォルトのゲートウェイの名前は、次のとおりです。

- Default-DeviceTracker ゲートウェイ
- Default-SYNC ゲートウェイ
- Default-UDP ゲートウェイ
- Default-SMTP ゲートウェイ

デフォルト・ゲートウェイを削除したり、名前を変更したりしないでください。名前の異なる追加のゲートウェイを作成することをおすすめします。

DefaultSYNC と DefaultUDP で定義されているプロパティを変更する必要はありませんが、DefaultSYNC ゲートウェイには、SMTP サーバ情報を指定する必要があります。デフォルトのゲートウェイを使用する必要がありますが、必要に応じて代替設定を使用できます。この項では、ゲートウェイ・プロパティをカスタマイズする手順について説明します。

デバイス・トラッキング・ゲートウェイ・プロパティ

デバイス・トラッキング・ゲートウェイ・プロパティを使用すると、デバイス・トラッキング・ゲートウェイの動作を変更できます。デバイス・トラッキング・ゲートウェイ・プロパティは、すべてオプションです。これらのプロパティの設定については、「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

プロパティ	値	説明
confirm_action	{ **yes** | **no** }	確認が配信時にこのゲートウェイを通じて送信されるかどうかを指定します。デフォルト値は **no** です。
confirm_delivery	{ **yes** | **no** }	Listener がメッセージを受信する統合データベースを確認するかどうかを指定します。デフォルト値は **yes** です。Listener は、-x Listener オプションを指定して起動する必要があります。
description	*description_text*	ゲートウェイに関する説明です。
enable	{ **yes** | **no** }	デバイス・トラッキング・ゲートウェイを使用するかどうかを指定します。

プロパティ	値	説明
smtp_gateway	*smtp_gateway_name*	SMTP 従属ゲートウェイの名前を指定します。デフォルト値は **DefaultSMTP** です。デバイス・トラッキング・ゲートウェイが使用できる SMTP ゲートウェイは、1 つのみです。このゲートウェイは有効にしておく必要があります。
sync_gateway	*sync_gateway_name*	SYNC 従属ゲートウェイの名前を指定します。デフォルト値は **DefaultSYNC** です。デバイス・トラッキング・ゲートウェイが使用できる SYNC ゲートウェイは、1 つのみです。このゲートウェイは有効にしておく必要があります。
udp_gateway	*udp_gateway_name*	UDP 従属ゲートウェイの名前を指定します。デフォルト値は **DefaultUDP** です。デバイス・トラッキング・ゲートウェイが使用できる UDP ゲートウェイは、1 つのみです。このゲートウェイは有効にしておく必要があります。

SMTP ゲートウェイ・プロパティ

SMTP ゲートウェイ・プロパティを使用すると、SMTP ゲートウェイの動作を変更できます。サーバのプロパティは必須ですが、他の SMTP ゲートウェイ・プロパティは、すべてオプションです。これらのプロパティの設定については、「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

プロパティ	値	説明
confirm_action	{ **yes** | **no** }	確認が配信時にこのゲートウェイを通じて送信されるかどうかを指定します。デフォルト値は **no** です。
confirm_delivery	{ **yes** | **no** }	このゲートウェイが配信を確認するかどうかを指定します。デフォルト値は **no** です。

プロパティ	値	説明
confirm_timeout	*number*{ **s** | **m** | **h** }	確認がタイムアウトになるまでに待機する時間を指定します。次に、使用可能な時間単位のリストを示します。 ● **s**：秒単位。 ● **m**：分単位。 ● **h**：時間単位。 デフォルト値は **1m** です。時間単位は、*HH***h** *MM***m** *SS***s** のフォーマットで組み合わせることができます。時間単位が指定されていない場合、時間は秒単位で測定されます。
description	*description_text*	ゲートウェイに関する説明です。
enable	{ **yes** | **no** }	SYNC ゲートウェイを使用するかどうかを指定します。
Listeners_are_900	{ **yes** | **no** }	すべての Listener が Adaptive Server Anywhere 9.0.0 クライアントであるかどうかを指定します。デフォルト値は **no** です。Adaptive Server Anywhere 9.0.1 以降のクライアントについては、この値を **no** のままにしておきます。
password	*password*	SMTP サービスのパスワードを指定します。一部のサービスでは必須です。
sender	*SMTP_address*	SMTP Push 通知の送信側アドレスを指定します。デフォルト値は **anonymous** です。
server	*IP_address_or_hostname*	メッセージを Listener に送信するために使用する SMTP サーバの IP アドレスまたはホスト名を指定します。デフォルト値は **mail** です。
user	*username*	SMTP サービスのユーザ名を指定します。一部のサービスでは必須です。

SYNC ゲートウェイ・プロパティ

SYNC ゲートウェイ・プロパティを使用すると、SYNC ゲートウェイの動作を変更できます。SYNC ゲートウェイ・プロパティは、すべてオプションです。これらのプロパティの設定については、「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

プロパティ	値	説明
confirm_action	{ **yes** | **no** }	確認が配信時にこのゲートウェイを通じて送信されるかどうかを指定します。デフォルト値は **no** です。
confirm_delivery	{ **yes** | **no** }	このゲートウェイが配信を確認するかどうかを指定します。デフォルト値は **no** です。
confirm_timeout	*number*{ **s** | **m** | **h** }	確認がタイムアウトになるまでに待機する時間を指定します。次に、使用可能な時間単位のリストを示します。 ● **s**：秒単位。 ● **m**：分単位。 ● **h**：時間単位。 デフォルト値は **1m** です。時間単位は、*HH***h** *MM***m** *SS***s** のフォーマットで組み合わせることができます。時間単位が指定されていない場合、時間は秒単位で測定されます。
description	*description_text*	ゲートウェイに関する説明です。
enable	{ **yes** | **no** }	SYNC ゲートウェイを使用するかどうかを指定します。
Listeners_are_900	{ **yes** | **no** }	すべての Listener が Adaptive Server Anywhere 9.0.0 クライアントであるかどうかを指定します。デフォルト値は **no** です。Adaptive Server Anywhere 9.0.1 以降のクライアントについては、この値を **no** のままにしておきます。

UDP ゲートウェイ・プロパティ

UDP ゲートウェイ・プロパティを使用すると、IP アドレスやポート番号など、UDP ゲートウェイの動作を変更できます。UDP ゲートウェイ・プロパティは、すべてオプションです。これらのプロパティの設定については、「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

プロパティ	値	説明
confirm_action	{ **yes** | **no** }	確認が配信時にこのゲートウェイを通じて送信されるかどうかを指定します。デフォルト値は **no** です。

プロパティ	値	説明
confirm_delivery	{ **yes** | **no** }	このゲートウェイが配信を確認するかどうかを指定します。デフォルト値は **yes** です。
confirm_timeout	*number*{ **s** | **m** | **h** }	確認がタイムアウトになるまでに待機する時間を指定します。次に、使用可能な時間単位のリストを示します。 ● **s**：秒単位。 ● **m**：分単位。 ● **h**：時間単位。 デフォルト値は **1m** です。時間単位は、*HH***h** *MM***m** *SS***s** のフォーマットで組み合わせることができます。時間単位が指定されていない場合、時間は秒単位で測定されます。
description	*description_text*	ゲートウェイに関する説明です。
enable	{ **yes** | **no** }	UDP ゲートウェイを使用するかどうかを指定します。
Listeners_are_900	{ **yes** | **no** }	すべての Listener が Adaptive Server Anywhere 9.0.0 クライアントであるかどうかを指定します。デフォルト値は **no** です。Adaptive Server Anywhere 9.0.1 以降のクライアントについては、この値を **no** のままにしておきます。
Listener_port	*port_number*	リモート・デバイスが UDP パケットの送信に使用するポートを指定します。デフォルト値は **5001** です。
sender	*IP_address_or_hostname*	マルチホームのホストの場合にのみ使用します。送信者の IP アドレスまたはホスト名を指定します。デフォルト値は **localhost** です。
sender_port	*port_number*	UDP パケットの送信に使用するポート番号を指定します。デフォルトでは、オペレーティング・システムによって空きポート番号がランダムに割り当てられます。

Carrier プロパティ

Carrier プロパティを使用すると、無線通信事業者設定の動作を変更できます。この設定では、電話番号自動追跡のマッピングや電子メール・アドレスに対するネットワーク・プロバイダのマッピングに関する情報を提供します。Carrier プロパティはすべてオプションで、SMTP ゲートウェイを使用している場合にのみ必須です。

これらのプロパティの設定については、「サーバ起動同期の Mobile Link サーバ設定」 37 ページを参照してください。

プロパティ	値	説明
enable	{ **yes** | **no** }	Carrier を使用するかどうかを指定します。
description	*description_text*	Carrier に関する説明です。
network_provider_id	*id_text*	ネットワーク・プロバイダ ID を指定します。 Windows Mobile Phone Edition で SMS を使用するには、このプロパティを **_generic_** に設定します。
sms_email_domain	*domain_name*	Carrier のドメイン名を指定します。
sms_email_user_prefix	*prefix_name*	電子メール・アドレスで使用されるプレフィクスを指定します。

Mobile Link Listener ユーティリティ

目次

Windows デバイス用の Listener ユーティリティ

Listener ユーティリティは、Windows Mobile デバイスを含む Windows デバイスで実行します。

構文

dblsn [*options*] **-l** *message-handler* [**-l** *message-handler*...]

message-handler :
[*polling-option*;...] [*filter*;...]*action*; [*option*;...]

polling-option :
[**;poll_connect =** *string*]
[**;poll_notifier =** *string*]
[**;poll_key =** *string*]
[**;poll_every =** *number*]

option :
[**;continue = yes**]
[**;confirm_action = yes**]
[**;confirm_delivery = no**]
[**;maydial = no**]

filter :
[**subject =** *string*]
[**content =** *string*]
[**message =** *string* | **message_start =** *string*]
[**sender =** *string*]

action :
action = *command*[**;altaction =** *command*]

command :
START *program* [*program-arguments*]
| **RUN** *program* [*program-arguments*]
| **POST** *window-message* **TO** { *window-class-name* | *window-title* }
| *tcpip-socket-action*
| **DBLSN FULL SHUTDOWN**

tcpip-socket-action :
SOCKET port=*app-port*
[**;host=***app-host*]
[**;sendText=***text1*]
[**;recvText=** *text2* [**;timeout=***num-sec*]]

window-message : *string* | *message-id*

参照

- 「Palm デバイス用ユーティリティ」 87 ページ
- 「Windows 用の Listener オプション」 65 ページ
- 「Windows 用の Listener キーワード」 77 ページ
- 「Windows 用の Listener アクション・コマンド」 80 ページ
- 「Windows 用の Listener action 変数」 83 ページ

Windows 用の受信ライブラリ

Listener には、UDP 受信ライブラリ *lsn_udp.dll* が付属しており、デフォルトではこのライブラリがロードされます。

SMTP ゲートウェイを使用する場合は、SMTP 受信ライブラリを指定する必要があります。ライブラリは、dblsn -d オプションを使用して指定できます。ライブラリのオプションは、dblsn -a オプションを使用して指定できます。

UDP (lsn_udp.dll)

UDP 受信ライブラリでサポートされているオプションのリストは、次のとおりです。

オプション	説明
Port=*port-number*	このオプションでは、受信するポート番号を指定します。デフォルトのポートは **5001** です。
Timeout=*seconds*	このオプションでは、UDP 受信ポートでの読み込み処理の最大ブロック時間を指定します。この値は、UDP 受信スレッドのポーリング間隔より小さくしてください。デフォルトは **0** です。
ShowSenderPort	このオプションは、$sender action 変数のすべての出現箇所で送信側ポート番号を示します。デフォルトでは、ポート番号は非表示です。このオプションを指定すると、ポート番号が **:***port-number* の構文で送信側アドレスの最後に追加されます。
HideWSAErrorBox	ソケット操作でのエラーを示すエラー・ボックスを表示しません。
CodePage=*number*	マルチバイト文字は、この番号に基づいて Unicode に変換されます。このオプションが適用されるのは、Windows Mobile のみです。

参照

- 「-d オプション」 69 ページ
- 「-a オプション」 68 ページ

Windows 用の Listener オプション

次のオプションは、Listener の設定に使用できます。

オプション	説明
@{ *variable* | *filename* }	指定された環境変数またはテキスト・ファイルからの Listener オプションを適用します。「@ option」 67 ページを参照してください。
-a *value*	受信ライブラリの 1 つのライブラリ・オプションを指定します。「-a オプション」 68 ページを参照してください。
-d *filename*	受信ライブラリを指定します。「-d オプション」 69 ページを参照してください。
-e *device-name*	デバイス名を指定します。「-e オプション」 69 ページを参照してください。
-f *string*	デバイスに関する追加の情報を指定します。「-f オプション」 70 ページを参照してください。
-gi *seconds*	IP トラッカのポーリング間隔を指定します。「-gi オプション」 70 ページを参照してください。
-i *seconds*	SMTP 接続のポーリング間隔を指定します。「-i オプション」 70 ページを参照してください。
-l "*keyword=value;...***"**	メッセージ・ハンドラの定義と作成を行います。「-l オプション」 71 ページを参照してください。
-m	メッセージのロギングを有効にします。「-m オプション」 71 ページを参照してください。
-ni	IP 追跡を無効にします。「-ni オプション」 71 ページを参照してください。
-ns	SMS 受信を無効にします。「-ns オプション」 72 ページを参照してください。
-nu	UDP 受信を無効にします。「-nu オプション」 72 ページを参照してください。
-o *filename*	ファイルに出力のログを取ります。「-o オプション」 72 ページを参照してください。
-os *bytes*	ログ・ファイルの最大サイズを指定します。「-os オプション」 72 ページを参照してください。

オプション	説明
-ot *filename*	ファイルをトランケートし、そのファイルに出力を記録します。「-ot オプション」 73 ページを参照してください。
-p	アイドル状態になったときにデバイスを自動的に停止できます。「-p オプション」 73 ページを参照してください。
-pc { + | - }	永続的な接続を有効または無効にします。「-pc オプション」 73 ページを参照してください。
-q	Listener をクワイエット・モードで実行します。「-q オプション」 74 ページを参照してください。
-r *filename*	メッセージ・フィルタの応答アクションに関わるリモート・データベースを指定します。「-r オプション」 74 ページを参照してください。
-sv *script-version*	認証に使用されるスクリプト・バージョンを指定します。「-sv オプション」 74 ページを参照してください。
-t {+ | - } *name*	リモート・データベースのリモート ID の登録または登録解除を行います。「-t オプション」 75 ページを参照してください。
-u *username*	Mobile Link ユーザ名を指定します。「-u オプション」 75 ページを参照してください。
-v { **0** | **1** | **2** | **3** }	メッセージ・ログの冗長性レベルを指定します。「-v オプション」 76 ページを参照してください。
-w *password*	Mobile Link パスワードを指定します。「-w オプション」 76 ページを参照してください。
-x { **http** | **https** | **tcpip** } [**(** *protocol-option=value;...* **)**]	ネットワーク・プロトコルと、Mobile Link サーバのプロトコル・オプションを指定します。「-x オプション」 77 ページを参照してください。
-y *newpassword*	新しい Mobile Link パスワードを指定します。「-y オプション」 77 ページを参照してください。

@ option

指定された環境変数またはテキスト・ファイルからの Listener オプションを適用します。

構文

dblsn @{ *variable* | *filename* } ...

備考

デフォルトでは、パラメータなしで Listener を実行した場合の引数ファイルは *dblsn.txt* です。

同じ名前を持つファイルと環境変数が存在する場合は、環境変数が使用されます。

ファイル難読化ユーティリティは、テキスト・ファイル内のパスワードなどの機密性の高い情報を難読化する場合に使用します。「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。

参照

- 「設定ファイルの使用」 『SQL Anywhere サーバ - データベース管理』

例

サンプルのテキスト・ファイルは、*samples-dir¥MobiLink¥SIS_SimpleListener¥dblsn.txt* にあります。

次の例では、**dblsnoptions** 環境変数にコマンド・ライン・オプションを保存します。

```
dblsn @dblsnoptions
```

次の例では、完全に修飾されたテキスト・ファイルである **mydblsn.txt** にコマンド・ライン・オプションを保存します。

```
dblsn @mydblsn.txt
```

-a オプション

受信ライブラリの 1 つのライブラリ・オプションを指定します。

構文

dblsn -a *value* ...

備考

デフォルトでは、ライブラリを指定しない場合、Listener は *lsn_udp.dll* を使用します。他のライブラリまたは追加のライブラリを指定するには、-d オプションを使用します。

使用可能なライブラリ・オプションをすべて表示するには、**?** 値を使用します。

追加のライブラリ・オプションを設定するには、-a オプションを複数回使用します。

参照

- 「Windows 用の受信ライブラリ」 65 ページ
- 「-d オプション」 69 ページ

例

次の例では、ポート・オプションを指定し、*lsn_udp.dll* 受信ライブラリの ShowSenderPort オプションを宣言しています。

```
dblsn -d lsn_udp.dll -a port=1234 -a ShowSenderPort
```

次の例では、2 つの異なるライブラリのポート・オプションを指定しています。

```
dblsn -d lsn_udp.dll -a port=1234 -d maac750.dll -a port=2345
```

次の例では、デフォルトのライブラリで使用できるライブラリ・オプションをすべて表示します。

```
dblsn -a ?
```

-d オプション

受信ライブラリを指定します。

構文

dblsn -d *filename* ...

備考

デフォルトでは、Listener は *lsn_udp.dll* 受信ライブラリを使用します。

複数の媒体での受信を可能にするマルチ・チャネル受信を有効にするには、-d オプションを複数回使用します。

参照

- 「Windows 用の受信ライブラリ」 65 ページ

例

次の例では、*maac750.dll* 受信ライブラリを指定しています。

```
dblsn -d maac750.dll
```

-e オプション

デバイス名を指定します。

構文

dblsn -e *device-name* ...

備考

デフォルトでは、デバイス名はオペレーティング・システムから自動的に生成されます。

Mobile Link サーバに接続するときは、すべてのデバイス名がユニークであることを確認してください。

デバイス名は Listener ウィンドウで参照できます。

-f オプション

デバイスに関する追加の情報を指定します。

構文

dblsn -f *string* ...

備考

デフォルトでは、この情報はデバイス上で実行されているオペレーティング・システムのバージョン番号です。

-gi オプション

IP トラッカのポーリング間隔を指定します。

構文

dblsn -gi *number* ...

備考

デフォルトでは、IP トラッカは 60 秒ごとにポーリングします。

-i オプション

SMTP 接続のポーリング間隔を指定します。

構文

dblsn -i *number* ...

備考

-i オプションでは、Listener がメッセージをチェックする頻度を指定します。

SMTP 接続の場合、デフォルト値は 30 秒です。UDP 接続の場合、Listener はすぐに接続します。

-i オプションは、-d オプションで指定された受信ライブラリごとに 1 回使用できます。

参照

● 「-d オプション」 69 ページ

例

次の例では、2 つの異なるライブラリのポーリング間隔を指定しています。

```
dblsn -d lsn_udp.dll -i 60 -d maac750.dll -i 45
```


-l オプション

メッセージ・ハンドラの定義と作成を行います。

構文

dblsn -l "*keyword=value;...***" ...**

備考

指定可能なキーワードのリストについては、「Windows 用の Listener キーワード」 77 ページを参照してください。

Push 通知の追加のメッセージ・ハンドラを定義するには、-l オプションを複数回使用します。メッセージ・ハンドラは、指定した順序で処理されます。

参照

- 「メッセージ・ハンドラ」 19 ページ

-m オプション

メッセージのロギングを有効にします。

構文

dblsn -m ...

備考

デフォルトでは、メッセージのロギングはオフです。

-ni オプション

IP 追跡を無効にします。

構文

dblsn -ni ...

備考

デフォルトでは、IP 追跡は有効です。

このオプションでは、配信確認は停止しません。

-ni オプションと -x オプションを一緒に使用すると、UDP アドレスのトラッキングが無効になります。この機能は、デバイス・トラッキングで UDP アドレスの更新を除外する場合に役立ちます。

参照

- 「-x オプション」 77 ページ

-ns オプション

SMS 受信を無効にします。

構文

dblsn -ns ...

備考

デフォルトでは、Windows Mobile 2003 以降では SMS 受信が有効です。

-nu オプション

UDP 受信を無効にします。

構文

dblsn -nu ...

備考

デフォルトでは、UDP 受信は有効です。

-o オプション

ファイルに出力のログを取ります。

構文

dblsn -o *filename* ...

備考

デフォルトでは、出力は Listener ウィンドウに表示されます。

参照

- 「-ot オプション」 73 ページ

-os オプション

ログ・ファイルの最大サイズを指定します。

構文

dblsn -os *bytes* ...

備考

デフォルトでは、最大サイズは無制限となります。最小のサイズ制限は 10000 です。

-ot オプション

ファイルをトランケートし、そのファイルに出力を記録します。

構文

dblsn -ot *filename* ...

備考

ファイルの内容が削除されてから、出力が記録されます。

参照

- 「-o オプション」 72 ページ

-p オプション

アイドル状態になったときにデバイスを自動的に停止できます。

構文

dblsn -p ...

備考

デフォルトでは、Listener がデバイスの停止を防止します。

このオプションが適用されるのは、Windows Mobile のみです。

-pc オプション

構文

永続的な接続を有効または無効にします。

dblsn -pc { + | - } ...

備考

デフォルトでは、永続的接続は有効です。- フラグを指定すると、永続的接続が無効になります。+ フラグを指定すると、有効になります。

永続的接続を無効にすると、Listener は Push 通知を受信できなくなりますが、短命に終わった永続的接続がデバイス・トラッキングと確認用に有効になります。

永続的接続が切断された場合、Listener は継続的に再接続を試みます。

例

次の例では、永続的接続を無効にします。

```
dblsn -pc-
```

-q オプション

Listener をクワイエット・モードで実行します。

構文

dblsn -q ...

備考

-q オプションを指定すると、Listener ウィンドウが最小化されます。デフォルトでは、Listener ウィンドウが表示されます。

-r オプション

メッセージ・フィルタの応答アクションに関わるリモート・データベースを指定します。

構文

dblsn -r *filename* ...

備考

filename には、RID ファイルのフル・パスを指定する必要があります。このファイルは、最初に同期した後に dbmlsync によって自動的に作成されます。データベース・ファイルと同じロケーションと名前が使用されます。Ultra Light データベースの場合は、*filename* をデータベース名と同じにする必要があります。

-r オプションを適用すると、メッセージ・ハンドラで $remote_id action 変数を使用して、RID ファイルのリモート ID を参照できます。デフォルトでは、リモート ID には GUID が使用されます。

複数のデータベースを識別するには、-r オプションを複数回使用します。

参照

- 「メッセージ・ハンドラの使用」 20 ページ
- 「Windows 用の Listener アクション・コマンド」 80 ページ

-sv オプション

認証に使用されるスクリプト・バージョンを指定します。

構文

dblsn -sv *script-version* ...

備考

デフォルトでは、ml_global サーバ・スクリプト・バージョンが定義されていると、Listener はこのスクリプト・バージョンを使用します。

-t オプション

リモート・データベースのリモート ID の登録または登録解除を行います。

構文

dblsn -t { + | - } *name ...*

備考

+ フラグを指定すると、リモート ID が登録されます。- フラグを指定すると、ID の登録が解除されます。

登録を行うと、Listener はそのリモート ID を参照して Push 通知を指定できます。

デバイス・トラッキング情報のアップロードに成功すると、登録された ID がサーバの ml_listening システム・テーブルに保持されます。ID の登録が必要となるのは一度のみです。

複数の ID を登録または登録解除するには、-t オプションを複数回使用します。複数の ID を登録すると、複数のリモート・データベースに Push 通知を指定する場合に役立ちます。

参照

- 「Windows 用の Listener アクション・コマンド」 80 ページ

-u オプション

Listener の Mobile Link ユーザ名を指定します。

構文

dblsn -u *username ...*

備考

デフォルトでは、ユーザ名は *device-name*-**dblsn** です。*device-name* には、デバイスの名前を指定します。デバイス名は、-e オプションを使用して指定できます。

Listener では、ユーザ名を使用して Mobile Link サーバに接続し、デバイス・トラッキング、確認、永続的接続を行います。

ユーザ名には、Mobile Link サーバに登録されているユニークな Mobile Link ユーザ名を使用する必要があります。この名前は、統合データベースの ml_user システム・テーブルに存在します。

参照

- 「-e オプション」 69 ページ
- 「-w オプション」 76 ページ
- 「Mobile Link ユーザの作成と登録」 『Mobile Link - クライアント管理』

-v オプション

メッセージ・ログの冗長性レベルを指定します。

構文

dblsn -v { 0 | 1 | 2 | 3 } ...

備考

デフォルトでは、冗長性レベルは 0 に設定されています。

次の表に、使用可能な冗長性レベル値の概要を示します。

冗長性レベル	説明
0	冗長性はオフです。
1	受信ライブラリ・メッセージ、基本的なアクション・トレース段階、コマンド・ライン・オプションを表示します。
2	レベル 1 の冗長性メッセージと、詳細なアクションのトレース段階を表示します。
3	レベル 2 の冗長性メッセージ、ポーリング・ステータス、受信ステータスを表示します。

メッセージ・ログに Push 通知を出力するには、-m オプションを使用する必要があります。

参照

- 「-m オプション」 71 ページ
- 「データベース・サーバ・メッセージをファイルにロギングする」 『SQL Anywhere サーバ - データベース管理』

-w オプション

Mobile Link パスワードを指定します。

構文

dblsn -w *password* ...

備考

パスワードは、関連する Mobile Link ユーザ名のもとで Mobile Link サーバに登録されている必要があります。

Listener では、パスワードを使用して Mobile Link サーバに接続し、デバイス・トラッキング、確認、永続的接続を行います。

参照

- 「-u オプション」 75 ページ

-x オプション

ネットワーク・プロトコルと、Mobile Link サーバのプロトコル・オプションを指定します。

構文

dblsn -x { **http** | **https** | **tcpip** } [(*protocol-option*=*value*;...)] ...

備考

Mobile Link サーバへの接続は、Listener がデバイス・トラッキング情報や配信確認を統合データベースに送信するために必要です。Mobile Link サーバのロケーションを指定するには、**host** プロトコル・オプションを使用します。「host」 『Mobile Link - クライアント管理』を参照してください。

-x オプションを指定すると、サーバのアドレスが変更された場合に、デバイスで統合データベースを更新できます。

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」 『Mobile Link - クライアント管理』

-y オプション

新しい Mobile Link パスワードを指定します。

構文

dblsn -y *newpassword* ...

備考

リモート・デバイスによるパスワードの変更が認証システムで許可されていない場合は、-y オプションを利用できません。

Windows 用の Listener キーワード

次のキーワードを使用すると、dblsn -l オプションを使用して作成されたメッセージ・ハンドラを設定できます。Listener キーワードの使用方法の詳細については、「-l オプション」 71 ページを参照してください。

フィルタのキーワード

Push 通知内のメッセージをフィルタするには、次のキーワードを使用します。

キーワードの構文	説明
subject=*subject-string*	件名のテキストが *subject-string* と同等の場合にメッセージをフィルタします。
content=*content-string*	内容のテキストが *content-string* と同等の場合にメッセージをフィルタします。
message=*message-string*	メッセージ全体のテキストが *message-string* と同等の場合にメッセージをフィルタします。
message_start=*message-string*	メッセージが *message-string* で始まる場合にメッセージをフィルタします。
sender=*sender-string*	メッセージの送信者が *sender-string* の場合にメッセージをフィルタします。

アクションのキーワード

フィルタ条件が満たされている場合にアクションを開始するには、次のキーワードを使用します。

キーワードの構文	説明
action=*command*	アクション・コマンドを指定します。「Windows 用の Listener アクション・コマンド」 80 ページを参照してください。
altaction=*command*	アクション・コマンドが失敗した場合に開始される代替アクション・コマンドを指定します。「Windows 用の Listener アクション・コマンド」 80 ページを参照してください。

ポーリングのオプション

ライトウェイト・ポーラを設定するには、次のオプションを使用します。

キーワードの構文	説明
poll_connect={ **http** | **https** | **tcpip** } [(*protocol-option*=*value*;...)]	サーバへの接続に必要なライトウェイト・ネットワーク・プロトコル・オプションを指定します。デフォルト値は、dblsn -x オプションから継承されます。「-x オプション」 77 ページを参照してください。
poll_notifier=*Notifier-string*	Push 要求を処理する Notifier の名前を指定します。必須。

キーワードの構文	説明
poll_key=*key-string*	Notifier に Listener 自体を示すための Listener の名前を指定します。この値は、ユニークにする必要があります。必須。
poll_every=*seconds-number*	Listener がサーバをポーリングする頻度を指定します。間隔は秒単位で測定されます。デフォルト値は、Mobile Link サーバから自動的に取得されます。

オプション

次のオプションを使用すると、メッセージ・ハンドラの動作を設定できます。

キーワードの構文	説明
continue=[**yes** | **no**]	最初の一致を検出した後に Listener が受信を継続するかどうかを指定します。デフォルト値は **no** です。**yes** 値を使用すると、複数のフィルタを指定するときに、1 つのメッセージによって複数のアクションが開始される場合に役立ちます。
confirm_action=[**yes** | **no**]	フィルタがアクションを確認するかどうかを指定します。デフォルト値は **yes** です。
confirm_delivery=[**yes** | **no**]	フィルタが条件付きのメッセージ配信を確認するかどうかを指定します。デフォルト値は **yes** です。したがって、最初のフィルタがメッセージを受け入れたときに配信確認が送信されます。 メッセージの確認が必要で、フィルタがメッセージを受け入れた場合にのみ、配信を確認できます。指定したゲートウェイに、**yes** に設定された confirm_delivery キーワード値が定義されている場合は、メッセージに確認が必要です。**no** 値は、複数のフィルタが同一メッセージを受け入れる場合に、どのフィルタが配信確認をするかを細かく制御するために使用できます。サーバでの配信確認の処理の詳細については、「confirmation_handler イベント」 50 ページを参照してください。
maydial=[**yes** | **no**]	アクションがモデムにダイヤル接続するパーミッションがあるかどうかを指定します。デフォルト値は **yes** です。**no** 値を指定すると、Listener はアクションの前にモデムを解放します。

参照

- 「メッセージ・ハンドラ」 19 ページ
- 「Windows 用の Listener アクション・コマンド」 80 ページ

Windows 用の Listener アクション・コマンド

アクションは、新しいメッセージ・ハンドラを設定する場合に指定されます。フィルタ条件が満たされると、アクションが開始されます。アクションが失敗した場合は、代替アクションが開始されます。アクションは、**action** キーワードを使用して定義されます。代替アクションは、**altaction** キーワードを使用して定義されます。Listener の使用方法の詳細については、「Windows 用の Listener キーワード」 77 ページを参照してください。

アクション・コマンドのリストを次に示します。

コマンド	説明
START *program arglist*	バックグラウンドで Listener が実行されているときにプログラムを開始します。「START アクション・コマンド」 81 ページを参照してください。
RUN *program arglist*	Listener を一時停止してプログラムを実行します。「RUN アクション・コマンド」 81 ページを参照してください。
POST *windowmessage* | *id* to *windowclass* | *windowtitle*	ウィンドウ・クラスにウィンドウ・メッセージを送信します。「POST アクション・コマンド」 81 ページを参照してください。
SOCKET port=*windowname*[;**host**=*hostname*] [;**sendText**=*text*] [;**recvText**=*text*[;**timeout**=*seconds*]]	TCP/IP 接続を使用して、アプリケーションにメッセージを送信します。「SOCKET アクション・コマンド」 82 ページを参照してください。
DBLSN FULL SHUTDOWN	強制的に Listener をシャットダウンします。「DBLSN FULL SHUTDOWN アクション・コマンド」 83 ページを参照してください。

action キーワードまたは **altaction** キーワードごとに指定できるアクションは 1 つのみです。1 つのアクションで複数のタスクを実行する場合、複数のコマンドを含むバッチ・ファイルを作成し、**RUN** アクション・コマンドを使用してファイルを実行します。

参照

- 「アクションの開始」 21 ページ

START アクション・コマンド

バックグラウンドで Listener が実行されているときにプログラムを開始します。

構文

action='START *program arglist***'**

備考

プログラムを起動すると、Listener は Push 通知の受信を再開します。

Listener はプログラムの終了を待機しません。したがって、アクション・コマンドの実行に失敗したか、または指定したプログラムを起動できないかどうかのみを判定できます。

例

次の例では、コマンド・ライン・オプションをいくつか使用して dbmlsync を起動します。オプションの一部は、$content action 変数を使用してメッセージから取得されます。

```
dblsn -l "action='start dbmlsync.exe @dbmlsync.txt -n
  $content -wc dbmlsync_$content -e sch=INFINITE';"
```

RUN アクション・コマンド

Listener を一時停止してプログラムを実行します。

構文

action='RUN *program arglist***'**

備考

Listener は、プログラムの終了を待機してから受信を再開します。

プログラムを実行すると、Listener がプログラムを起動できない場合またはプログラムが 0 以外のリターン・コードを返した場合に、Listener はプログラムの実行に失敗したかどうかを判定します。

例

次の例では、コマンド・ライン・オプションをいくつか使用して dbmlsync を実行します。オプションの一部は、$content action 変数を使用してメッセージから取得されます。

```
dblsn -l "action='run dbmlsync.exe @dbmlsync.txt -n $content';"
```

POST アクション・コマンド

ウィンドウ・クラスにウィンドウ・メッセージを送信します。

構文

action='POST *windowmessage* | *id* to *windowclass* | *windowtitle***'**

備考

POST コマンドを使用すると、ウィンドウ・メッセージを使用するアプリケーションに通知できます。

ウィンドウ・メッセージは、メッセージの内容またはウィンドウ・メッセージ ID (存在する場合) によって識別できます。

ウィンドウ・クラスは、クラス名またはウィンドウ・タイトルによって識別できます。名前で識別する場合は、-wc dbmlsync オプションを使用すると、ウィンドウ・クラス名を指定できます。ウィンドウ・タイトルでウィンドウ・クラスを識別する場合は、最上位レベルのウィンドウのみでウィンドウ・クラスを参照できます。

ウィンドウ・メッセージまたはウィンドウ・クラス名に、スペースや句読点などの英数字以外の文字が含まれる場合は、テキストを一重引用符 (') で囲みます。また、エスケープ文字にも一重引用符を使用します。したがって、ウィンドウ・メッセージまたはウィンドウ・クラスに一重引用符が含まれる場合は、2 つの一重引用符 ('') を使用して引用符を参照してください。

例

一重引用符が含まれるウィンドウとメッセージの使用方法を示すため、次の例では mike's_message ウィンドウ・メッセージを mike's_class ウィンドウ・クラスに送信します。

```
dblsn -l "action='post mike''s_message to mike''s_class';"
```

次の例では、ウィンドウ・メッセージ dbas_synchronize を、クラス名 dbmlsync_FullSync で登録された dbmlsync インスタンスに送信します。

```
dblsn -l "action='post dbas_synchronize to dbmlsync_FullSync';"
```

参照

- 「-wc オプション」 『Mobile Link - クライアント管理』

SOCKET アクション・コマンド

TCP/IP 接続を使用して、アプリケーションにメッセージを送信します。

構文

action='SOCKET port=*windowname*[;**host=***hostname*][;**sendText=***text*]
[;**recvText=***text*[;**timeout=***seconds*]]'

備考

SOCKET コマンドを使用して、実行中のアプリケーションに動的な情報を渡し、Java アプリケーションや Visual Basic アプリケーションにメッセージを統合します。どちらの言語でも、カスタム・ウィンドウ・メッセージ機能はサポートされていません。また、eMbedded Visual Basic では、コマンド・ライン・パラメータがサポートされていません。

ソケットに接続するには、ポートとホストを指定する必要があります。メッセージの入力には sendText を使用します。

アプリケーションが sendText の受信に成功したことを確認するときにメッセージを表示するには、recvText を使用します。recvText を使用すると、タイムアウト制限を指定できます。Listener が接続できない場合、受信確認を送信できない場合、またはタイムアウト制限内に確認を受信できない場合は、アクションの実行に失敗します。

例

次の例では、ポート 12345 で受信しているローカル・アプリケーションに、$sender=$message で文字列を転送します。Listener では、確認としてアプリケーションが 5 秒以内に "beeperAck" を送信することが予期されます。

```
dblsn -l "action='socket port=12345;
  sendText=$sender=$message;
  recvText=beeperAck;
  timeout=5'"
```

DBLSN FULL SHUTDOWN アクション・コマンド

強制的に Listener をシャットダウンします。

構文

action='DBLSN FULL SHUTDOWN'

備考

停止すると、Listener は Push 通知の処理を停止し、デバイス・トラッキング情報の同期を停止します。サーバ起動同期を続行するには、Listener を再度起動する必要があります。

Windows 用の Listener action 変数

アクションまたはフィルタでは、次の action 変数を使用できます。action 変数は、メッセージ・ハンドラを開始する前に値に置き換えられます。

action 変数は、ドル記号 ($) で始まります。エスケープ文字もドル記号であるため、1 つのドル記号をプレーン・テキストとして指定するには、2 つのドル記号 ($$) を使用します。

変数	説明
$subject	メッセージの件名。
$content	メッセージの内容。
$message	件名、内容、送信者を含むメッセージ全体。
$message_start	message_start フィルタ・キーワードで指定された、メッセージの先頭の一部。この変数を使用できるのは、message_start フィルタ・キーワードを指定した場合のみです。「Windows 用の Listener キーワード」 77 ページを参照してください。

変数	説明
$message_end	message_start フィルタ・キーワードで除外された、メッセージの一部。この変数を使用できるのは、message_start フィルタ・キーワードを指定した場合のみです。「Windows 用の Listener キーワード」 77 ページを参照してください。
$ml_connect	dblsn -x オプションによって指定された Mobile Link ネットワーク・プロトコル・オプション。デフォルトは、空の文字列です。「-x オプション」 77 ページを参照してください。
$ml_user	dblsn -u オプションによって指定された Mobile Link ユーザ名。デフォルトの名前は *device-name*-**dblsn** です。
$ml_password	dblsn -w オプションによって指定される Mobile Link パスワード、または -y を使用した場合は新しい Mobile Link パスワード。
$priority	この変数の意味は、carrier ライブラリに依存します。
$request_id	Push 要求で指定された要求 ID。「Push 要求」 10 ページを参照してください。
$remote_id	リモート ID です。この変数は、dblsn -r オプションが指定された場合にだけ使用されます。「リモート ID によるメッセージのフィルタリング」 24 ページを参照してください。
$sender	メッセージの送信者。
$type	この変数の意味は、carrier ライブラリに依存します。
$year	この変数の意味は、carrier ライブラリに依存します。
$month	この変数の意味は、carrier ライブラリに依存します。値は 1 ～ 12 までです。
$day	この変数の意味は、carrier ライブラリに依存します。値は 1 ～ 31 までです。
$hour	この変数の意味は、carrier ライブラリに依存します。値は 0 ～ 23 までです。
$minute	この変数の意味は、carrier ライブラリに依存します。値は 0 ～ 59 までです。
$second	この変数の意味は、carrier ライブラリに依存します。値は 0 ～ 59 までです。

変数	説明
$best_adapter_mac	dblsn -x オプションによって指定された Mobile Link サーバに到達するための最善の NIC の MAC アドレス。最善のルートが NIC を経由しない場合、この変数の値は空文字列になります。
$best_adapter_name	dblsn -x オプションによって指定された Mobile Link サーバに到達するための最善の NIC のアダプタ名。最善のルートが NIC を経由しない場合、この変数の値は空文字列になります。
$best_ip	dblsn -x オプションによって指定された Mobile Link サーバに到達するための最善の IP インタフェースの IP アドレス。サーバが到達不能な場合、この変数の値は 0.0.0.0 になります。
$best_network_name	dblsn -x オプションによって指定された Mobile Link サーバに到達するための最善のプロファイルの RAS またはダイヤルアップ・プロファイル名。最善のルートが RAS またはダイヤルアップ接続を経由しない場合、この変数の値は空文字列になります。
$adapters	アクティブなネットワーク・アダプタ名のリストで、それぞれパイプ記号 (|) で分割します。
$network_names	接続 RAS エントリ名のリストで、それぞれパイプ記号 (|) で分割します。RAS エントリ名は、ダイヤルアップ・ネットワーク (DUN) のダイヤルアップ・エントリ名と呼ばれる場合もあります。
$poll_connect	poll_connect ポーリング・キーワードによって指定された Mobile Link ネットワーク・プロトコル・オプション。デフォルトは、空の文字列です。「Windows 用の Listener キーワード」 77 ページを参照してください。
$poll_notifier	poll_notifier ポーリング・キーワードによって指定された Notifier の名前。「Windows 用の Listener キーワード」 77 ページを参照してください。
$poll_key	poll_key ポーリング・キーワードによって指定されたポーリング・キー。「Windows 用の Listener キーワード」 77 ページを参照してください。
$poll_every	poll_every ポーリング・キーワードによって指定されたポーリング頻度。「Windows 用の Listener キーワード」 77 ページを参照してください。

参照

- 「-l オプション」 71 ページ
- 「Windows 用の Listener キーワード」 77 ページ
- 「Windows 用の Listener アクション・コマンド」 80 ページ

例

次の例では、$message_end action 変数を使用して、同期するパブリケーションを特定しています。

```
dblsn -l "message_start=start-of-message;action='run dbmlsync.exe -c ... -n $message_end'"
```


Palm デバイス用ユーティリティ

次の 2 つのユーティリティでは、サーバ起動同期を実行する必要があります。

- Palm デバイス用の Listener 設定ユーティリティ (dblsncfg)
- Palm デバイス用の Listener ユーティリティ (LsnT650.prc)

Palm Listener を準備するには、Windows デスクトップ上で Listener 設定ユーティリティを実行し、Palm 用の設定ファイルを作成してから HotSync を使用してデバイスにファイルを転送します。

注意
Windows デスクトップ上で Listener 設定ユーティリティを実行して Palm デバイス用の設定ファイルを生成する場合は、**RUN** アクション・コマンドを指定してください。ただし、Palm デバイスでは、Listener で Handler Editor を使用して、**RUN** アクションを削除できます。この方法では、アクションを開始することなくメッセージを消費できます。

Palm デバイス用の Listener 設定ユーティリティ

Windows デスクトップ上で Listener 設定ユーティリティを実行すると、Palm デバイス用の設定ファイルが作成されます。

構文

dblsncfg -n [*filename*] **-l** *message-handler* [**-l** *message-handler...*]

message-handler : [*filter***;**...] *action*

filter :
[**subject =** *string*]
[**content =** *string*]
[**message =** *string* | **message_start =** *string*]
[**sender =** *string*]

action : **action=run** *application-name* [*arguments*]

参照

- 「Palm 用の Listener オプション」 87 ページ
- 「Palm 用の Listener キーワード」 89 ページ
- 「Palm 用の Listener action 変数」 91 ページ

Palm 用の Listener オプション

次のオプションは、Listener の設定に使用できます。

オプション	説明
@{ *variable* | *filename* }	指定された環境変数またはテキスト・ファイルからの Listener オプションを適用します。「@ option」 88 ページを参照してください。
-l "*keyword=value;...***"**	メッセージ・ハンドラの定義と作成を行います。「-l オプション」 89 ページを参照してください。
-n *filename*	PDB 設定ファイルを作成します。「-n オプション」 89 ページを参照してください。

@ option

指定された環境変数またはテキスト・ファイルからの Listener オプションを適用します。

構文

dblsncfg @{ *variable* | *filename* } ...

備考

デフォルトでは、パラメータなしで Listener を実行すると、*dblsncfg.txt* が引数ファイルとして使用されます。

同じ名前を持つファイルと環境変数が存在する場合は、環境変数が使用されます。

ファイル難読化ユーティリティは、テキスト・ファイル内のパスワードなどの機密性の高い情報を難読化する場合に使用します。「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。

参照

- 「設定ファイルの使用」 『SQL Anywhere サーバ - データベース管理』

例

サンプルのテキスト・ファイルは、*samples-dir¥MobiLink¥SIS_SimpleListener¥dblsn.txt* にあります。

次の例では、**dblsnoptions** 環境変数にコマンド・ライン・オプションを保存します。

```
dblsn @dblsnoptions
```

次の例では、完全に修飾されたテキスト・ファイルである **mydblsn.txt** にコマンド・ライン・オプションを保存します。

```
dblsn @mydblsn.txt
```


-l オプション

メッセージ・ハンドラの定義と作成を行います。

構文

dblsncfg -l "*keyword=value;...***" ...**

備考

指定可能なキーワードのリストについては、「Palm 用の Listener キーワード」 89 ページを参照してください。

Push 通知の追加のメッセージ・ハンドラを定義するには、-l オプションを複数回使用します。メッセージ・ハンドラは、指定した順序で処理されます。

参照

- 「メッセージ・ハンドラ」 19 ページ

-n オプション

PDB 設定ファイルを作成します。

構文

dblsncfg -n *filename* **...**

備考

デフォルトでは、Listener によって *lsncfg.pdb* 設定ファイルが作成されます。

Palm 用の Listener キーワード

次のキーワードのリストを使用すると、dblsncfg -l オプションを使用して作成されたメッセージ・ハンドラを設定できます。Listener キーワードの使用方法の詳細については、「-l オプション」 89 ページを参照してください。

フィルタのキーワード

Push 通知内のメッセージをフィルタするには、次のキーワードを使用します。

キーワードの構文	説明
subject=*subject-string*	件名のテキストが *subject-string* と同等の場合にメッセージをフィルタします。
content=*content-string*	内容のテキストが *content-string* と同等の場合にメッセージをフィルタします。

キーワードの構文	説明
message=*message-string*	メッセージ全体のテキストが *message-string* と同等の場合にメッセージをフィルタします。
message_start=*message-string*	メッセージが *message-string* で始まる場合にメッセージをフィルタします。
sender=*sender-string*	メッセージの送信者が *sender-string* の場合にメッセージをフィルタします。

アクションのキーワード

フィルタ条件が満たされている場合にアクションを開始するには、次のキーワードを使用します。

キーワードの構文	説明
action=*command*	アクション・コマンドを指定します。「Palm 用の Listener action 変数」 91 ページを参照してください。
altaction=*command*	アクション・コマンドが失敗した場合に開始される代替アクション・コマンドを指定します。「Palm 用の Listener action 変数」 91 ページを参照してください。

Action キーワードと Altaction キーワード

アクションは、新しいメッセージ・ハンドラを設定する場合に指定されます。フィルタ条件が満たされると、アクションが開始されます。アクションが失敗した場合は、代替アクションが開始されます。アクションは、**action** キーワードを使用して定義されます。代替アクションは、**altaction** キーワードを使用して定義されます。

action キーワードまたは **altaction** キーワードごとに指定できるアクションは 1 つのみです。1 つのアクションで複数のタスクを実行する場合、複数のコマンドを含むバッチ・ファイルを作成し、**RUN** アクション・コマンドを使用してファイルを実行します。

Listener 設定ユーティリティでは、**RUN** *program arglist* コマンドをサポートしています。ここで、*program* は実行するプログラムの名前、*arglist* はアプリケーションによって異なる文字列です。

次の例に、Listener 設定ユーティリティを使用して、アクション・コマンドで yourapp.exe を実行する方法を示します。

```
dblsncfg -l "action='RUN yourapp.exe';"
```

ターゲット・アプリケーションの PilotMain ルーチンは、文字列をコマンド・ブロックとして取得します。

注意
Windows デスクトップ上で Listener 設定ユーティリティを実行して Palm デバイス用の設定ファイルを生成する場合は、**RUN** アクション・コマンドを指定してください。ただし、Palm デバイスでは、Listener で Handler Editor を使用して、**RUN** アクション・コマンドを削除できます。この方法では、アクションを開始することなくメッセージを消費できます。

参照

- 「メッセージ・ハンドラの使用」 20 ページ
- 「Palm 用の Listener action 変数」 91 ページ

Palm 用の Listener action 変数

アクションまたはフィルタでは、次の action 変数を使用できます。action 変数は、メッセージ・ハンドラを開始する前に値に置き換えられます。

action 変数は、ドル記号 ($) で始まります。エスケープ文字もドル記号であるため、1 つのドル記号をプレーン・テキストとして指定するには、2 つのドル記号 ($$) を使用します。

action 変数	説明
$subject	メッセージの件名。
$content	メッセージの内容。
$message	件名、内容、送信者を含むメッセージ全体。
$message_start	message_start フィルタ・キーワードで指定された、メッセージの先頭の一部。この変数を使用できるのは、message_start フィルタ・キーワードを指定した場合のみです。「Palm 用の Listener キーワード」 89 ページを参照してください。
$message_end	message_start フィルタ・キーワードで除外された、メッセージの一部。この変数を使用できるのは、message_start フィルタ・キーワードを指定した場合のみです。「Palm 用の Listener キーワード」 89 ページを参照してください。
$sender	メッセージの送信者。
$time	これは、1904 年 1 月 1 日 12:00 AM からの現在の時刻 (秒単位) です。

参照

- 「-l オプション」 89 ページ
- 「Palm 用の Listener キーワード」 89 ページ

Palm デバイス用の Listener ユーティリティ

Listener の配備

Palm アプリケーションでサーバ起動同期を使用するには、デバイスに Listener をインストールする必要があります。配備する必要のある Listener ファイルのリストは、次のとおりです。

- **LsnT650.prc**　Treo 650 用の Listener。
- **設定ファイル**　設定ファイルは、Windows デスクトップ上で Listener 設定ユーティリティを使用して作成する必要があります。現在、Listener によって読み込まれるのは、*lsncfg.pdb* 設定ファイルのみです。

Treo のデバイスは、Listener を動作させるために電源をオンにする必要はありません。

Listener オプション

Listener を使用すると、3 つのオプションを設定できます。これらのオプションは、明示的に変更されるまで、またはリセットが実行されるまで有効です。

- **リスニング**　Listener がメッセージを消費しないようにする方法です。
- **アクションの有効化**　これは、リスニングが有効な場合にのみ使用できます。無効な場合は、アクションが開始されません。
- **アクションの前のプロンプト表示**　これは、アクションが有効化されているときにのみ使用できます。このオプションを設定すると、アクションが開始される前に確認ウィンドウがポップアップ表示されます。

追加の Palm サポート

他の Palm デバイスをサポートするため、Palm デバイス用の Mobile Link Listener C API が提供されています。「Palm デバイス用 Mobile Link Listener C API」 93 ページを参照してください。

Palm デバイス用 Mobile Link Listener C API

目次

Palm デバイス用 Mobile Link Listener C API は、メッセージ処理用のメソッドとデバイス依存のメソッドが含まれる簡潔なプログラミング・インタフェースです。このインタフェースを使用すると、新しい Palm デバイス用の Listener や無線ネットワーク・アダプタ用の Listener を作成できます。

Listener API ファイル

API の実行に必要なファイルのリストと Palm デバイス用の実装例を次に示します。すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。

ファイル名またはロケーション	説明
SDK¥Listener¥Palm¥68k¥cw¥lib¥PalmLsn.lib	ランタイム・ライブラリ。メッセージ処理ルーチン、Listener コントロール、Handler Editor が含まれます。
SDK¥Listener¥Palm¥68k¥cw¥rsc	UI リソースが含まれます。
SDK¥Listener¥Palm¥src¥PalmLsn.h	ランタイム・ライブラリのヘッダと API。
SDK¥Listener¥Palm¥src¥Treo650.c	Treo 650 の実装

メッセージ処理用メソッド

Listener API では、a_palm_msg 構造体を使用して Listener メッセージを表現します。a_palm_msg インスタンスの割り当てと処理には、次のメソッドを使用します。

メソッド	説明
「PalmLsnAllocate メソッド」 98 ページ	新しい a_palm_msg インスタンスを返します。
「PalmLsnDupMessage メソッド」 100 ページ	a_palm_msg インスタンスのメッセージ・フィールドの値を初期化します。
「PalmLsnDupSender メソッド」 101 ページ	a_palm_msg インスタンスの送信元フィールドを初期化します。
「PalmLsnDupTime メソッド」 102 ページ	a_palm_msg インスタンスの時刻フィールドを初期化します。
「PalmLsnFree メソッド」 103 ページ	メッセージ用のメモリ領域を開放します。
「PalmLsnProcess メソッド」 108 ページ	設定データベース内のレコードに従ってメッセージを処理します。

デバイス依存メソッド

次のメソッドでは、本人確認の機能、登録機能、イベント処理機能を提供します。

メソッド	説明
「PalmLsnCheckConfigDB メソッド」 99 ページ	Listener 設定データベース内のエラーをレポートします。
「PalmLsnGetConfigFileName メソッド」 104 ページ	Listener 設定データベースの名前を含む文字列を返します。

メソッド	説明
「PalmLsnNormalHandleEvent メソッド」 105 ページ	アプリケーション・イベントを処理します。
「PalmLsnNormalStart メソッド」 106 ページ	Listener アプリケーション起動時のカスタムのアクションを登録します。
「PalmLsnNormalStop メソッド」 107 ページ	Listener アプリケーションがイベント・ループを終了するときのカスタム・アクションを実行します。
「PalmLsnSpecialLaunch メソッド」 110 ページ	デバイス依存の起動コードに応答します。
「PalmLsnTargetCompanyID メソッド」 113 ページ	デバイスの企業 ID または製造元 ID を返します。
「PalmLsnTargetDeviceID メソッド」 114 ページ	ターゲット・デバイス ID を返します。

LsnMain メソッド

Listener ライブラリ *PalmLsn.lib* のメイン・エントリ・ポイントを提供します。

構文

```
UInt32 LsnMain(
   UInt16 cmd,
   MemPtr cmdPBP,
   UInt16 launchFlags
)
```

パラメータ

- **cmd**　Palm OS アプリケーションの起動コード。
- **cmdPBP**　起動コード・パラメータが格納された構造体へのポインタ。起動コマンド固有のパラメータを持たないアプリケーションの場合は NULL。
- **launchFlags**　起動に関する追加情報を提供するフラグ。

戻り値

Palm OS エラー・コード。Listener ライブラリによって起動コードが正常に処理された場合は、errNone を返します。

備考

LsnMain に渡される値は、Palm OS アプリケーションのメイン・エントリ・ポイント関数 PilotMain に渡される起動コード・パラメータに似ています。

これらのパラメータの詳細については、ご使用の Palm OS のリファレンスを参照してください。

参照

- 「PalmLsnProcess メソッド」 108 ページ
- 「Palm デバイス用 Mobile Link Listener C API」 93 ページ

例

次に示す例は、Treo 650 smartphone の実装での使用例です。Listener アプリケーションのメイン・エントリ・ポイント LsnMain に、起動コード・パラメータを渡しています。

```
UInt32 PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags ) {
   return(LsnMain(cmd, cmdPBP, launchFlags));
}
```

palm_lsn_ret 列挙体

palm_lsn_ret 列挙体は、想定されるリターン・コードを指定します。

構文

```
typedef enum {
   PalmLsnOk = errNone,
   PalmLsnMissingConfig = appErrorClass,
   PalmLsnProblemReadingConfig,
   PalmLsnProblemParsingCmd,
   PalmLsnOutOfMemory,
   PalmLsnUnrecognizedAction,
   PalmLsnRunMissingApp
} palm_lsn_ret;
```

メンバ

名前	説明
PalmLsnOk	メソッド呼び出しの成功を示します。この値には、エラーなしを示す errNone と同じ値が格納されます。
PalmLsnMissingConfig	Listener 設定データベースが見つからないことを示します。このフィールドには、アプリケーション定義エラーを表す appErrorClass エラー・コードと同じ値が格納されます。
PalmLsnProblemReadingConfig	Listener 設定データベースの読み取りエラーを示します。
PalmLsnProblemParsingCmd	Listener 設定データベースに格納されているコマンドを処理できないことを示します。
PalmLsnOutOfMemory	メッセージ処理用のメモリ割り当て中にエラーが発生したため、メソッドの実行を継続できないことを示します。
PalmLsnUnrecognizedAction	Listener が、Listener 設定データベースで指定されたアクションをサポートしていないことを示します。
PalmLsnRunMissingApp	Listener が、run アクション・コマンドで指定されたアプリケーションを起動できないことを示します。

参照

- 「PalmLsnProcess メソッド」 108 ページ

PalmLsnAllocate メソッド

新しい a_palm_msg インスタンスを返します。

構文

struct a_palm_msg * **PalmLsnAllocate( )**

戻り値

すべてのフィールドが 0 に初期化された新しい a_palm_msg インスタンス。

参照

- 「PalmLsnFree メソッド」 103 ページ

例

次の例では、PalmLsnAllocate を使用して、a_palm_msg インスタンスを割り当てています。

```
a_palm_msg * ulMsg;

// Allocate a message structure
ulMsg = PalmLsnAllocate();
```


PalmLsnCheckConfigDB メソッド

Listener 設定データベース内のエラーをレポートします。

構文

```
palm_lsn_ret PalmLsnCheckConfigDB(
 Char const *  cfg,
 UInt16 * const rec
)
```

パラメータ

- **cfg**　設定データベースの名前を保持する文字配列。設定データベース名を取得するには、PalmLsnGetConfigFileName メソッドを使用します。「PalmLsnGetConfigFileName メソッド」 104 ページを参照してください。
- **rec**　設定データベース内の問題のあるレコードや間違った形式のレコードのインデックスを特定する出力パラメータ。

戻り値

palm_lsn_ret 列挙体にリストされているリターン・コードの 1 つ。「palm_lsn_ret 列挙体」 97 ページを参照してください。

備考

このメソッドを使用して、設定データベースのオープンやデータベース内のレコードの読み取りのときに発生したエラーを検出できます。

参照

- 「PalmLsnProcess メソッド」 108 ページ

例

次の例では、PalmLsnCheckConfigDB を使用して、設定データベース内の問題のあるレコードや間違った形式のレコードを検出しています。

```
Err     ret;
UInt16  badRec;
Char    configDb[dmDBNameLength];

// Get configuration database name
PalmLsnGetConfigFileName(configDb);

// check for errors in the configuration database
ret = PalmLsnCheckConfigDB(configDb, &badRec);
if (ret!=errNone) {
   // handle error
}
```


PalmLsnDupMessage メソッド

a_palm_msg インスタンスのメッセージ・フィールドの値を初期化します。

構文

```
Err PalmLsnDupMessage(
   struct a_palm_msg * const msg,
   Char const * message
)
```

パラメータ

- **msg**　a_palm_msg インスタンスへのポインタ。
- **message**　入力元メッセージのテキストを格納する入力パラメータ。

戻り値

Palm OS のエラー・コード。errNone は正常終了を表します。

備考

PalmLsnDupMessage メソッドは、テキスト・メッセージを複製し、件名、内容、送信元の各フィールドを抽出して、これらの値を a_palm_msg インスタンスに割り当てます。

送信元フィールドは、メッセージ内に現れない場合は抽出されません。PalmLsnDupSender を使用すると、PalmLsnDupMessage で抽出された送信元フィールドが上書きされます。

参照

- 「PalmLsnDupSender メソッド」 101 ページ
- 「PalmLsnDupTime メソッド」 102 ページ
- 「メッセージ処理用メソッド」 94 ページ

例

次に示す例は、Treo 650 smartphone の実装での使用例です。テキスト・メッセージを取り出し、PalmLsnDupMessage を呼び出して a_palm_msg インスタンス内の適切なフィールドを初期化しています。

```
// Retrieve the entire message body
ret = PhnLibGetText( libRef, id, &msgBodyH );
if (ret != errNone) {
    // handle error
    goto done;
}
msgBody = (Char *)MemHandleLock( msgBodyH );
ret = PalmLsnDupMessage( ulMsg, msgBody );

// msgBodyH must be disposed of by the caller
MemHandleUnlock(msgBodyH);
MemHandleFree(msgBodyH);
if (ret != errNone) {
    // handle error
    goto done;
}
```


PalmLsnDupSender メソッド

a_palm_msg インスタンスの送信元フィールドを初期化します。

構文

```
Err PalmLsnDupSender(
   struct a_palm_msg * const msg,
   Char const * sender
)
```

パラメータ

- **msg**　a_palm_msg インスタンスへのポインタ。
- **sender**　送信元フィールドを格納する入力パラメータ。

戻り値

Palm OS のエラー・コード。errNone は正常終了を表します。

備考

PalmLsnDupSender メソッドは、送信元入力パラメータを複製して、その値を a_palm_msg インスタンスに割り当てます。

参照

- 「PalmLsnDupMessage メソッド」 100 ページ
- 「PalmLsnDupTime メソッド」 102 ページ
- 「メッセージ処理用メソッド」 94 ページ

PalmLsnDupTime メソッド

a_palm_msg インスタンスの時刻フィールドを初期化します。

構文

```
Err PalmLsnDupTime(
   struct a_palm_msg * const msg,
   UInt32 const time
)
```

パラメータ

- **msg** a_palm_msg インスタンスへのポインタ。
- **time** 送信元の時刻フィールドを格納する入力パラメータ。

戻り値

Palm OS のエラー・コード。errNone は正常終了を表します。

備考

PalmLsnDupTime メソッドは、時刻入力パラメータを複製して、その値を a_palm_msg インスタンスに割り当てます。

参照

- 「PalmLsnDupMessage メソッド」 100 ページ
- 「PalmLsnDupSender メソッド」 101 ページ
- 「メッセージ処理用メソッド」 94 ページ

PalmLsnFree メソッド

メッセージ用のメモリ領域を開放します。

構文

void **PalmLsnFree(** struct a_palm_msg * const *msg* **)**

パラメータ

- **msg**　解放される a_palm_msg インスタンス。

例

次の例は、メッセージ構造体の割り当て、メッセージの処理、PalmLsnFree を使用したリソースの解放を実行するコードの一部です。

```
a_palm_msg * ulMsg;
...

// Allocate the message structure
ulMsg = PalmLsnAllocate();
...

// Fill the message fields
ret = PalmLsnDupMessage(ulMsg, msgBody);
...

// Process the message
ret = PalmLsnProcess(ulMsg, configDb, NULL, handled);
...

// Free the message
PalmLsnFree(ulMsg);
```


PalmLsnGetConfigFileName メソッド

Listener 設定データベースの名前を含む文字列を返します。

構文

void **PalmLsnGetConfigFileName(** Char * *configPDBName* **)**

パラメータ

- **configPDBName** Listener 設定データベースの名前が格納される出力パラメータ。

備考

このメソッドを使用すると、PalmLsnProcess に渡す設定データベース・ファイル名を取得できます。

デフォルトの設定データベース・ファイル名 *lsncfg* を使用するには、(*PalmLsn.h* に定義されている) PalmLsnDefaultConfigDB を出力パラメータにコピーします。

参照

- 「PalmLsnProcess メソッド」 108 ページ

例

次に示す例は、Treo 650 smartphone の実装での使用例です。デフォルトの設定データベース名を出力パラメータに返しています。

```
void PalmLsnGetConfigFileName(Char * configPDBName) {
   StrCopy(configPDBName, PalmLsnDefaultConfigDB);
}
```


PalmLsnNormalHandleEvent メソッド

アプリケーション・イベントを処理します。

構文

Boolean **PalmLsnNormalHandleEvent(** EventPtr *eventP* **)**

パラメータ

- **eventP**　アプリケーション・イベントへのポインタ。

戻り値

イベントが処理された場合、TRUE を返します。

備考

このメソッドを使用すると、アプリケーション・イベントを処理できます。

PalmLsnNormalStart メソッド

Listener アプリケーション起動時のカスタムのアクションを登録します。

構文

Err **PalmLsnNormalStart( )**

戻り値

Palm OS のエラー・コード。errNone は正常終了を表します。

備考

PalmLsnNormalStart を使用すると、Mobile Link サーバにデバイスを登録できます。

参照

- 「PalmLsnNormalStop メソッド」 107 ページ
- 「PalmLsnSpecialLaunch メソッド」 110 ページ

PalmLsnNormalStop メソッド

Listener アプリケーションがイベント・ループを終了するときのカスタム・アクションを実行します。

構文

void **PalmLsnNormalStop( )**

備考

受信を継続する場合は、PalmLsnNormalStop を使用してデバイスを登録解除しないでください。

このメソッドを使用すると、現在のアプリケーション設定を取得または設定できます。

参照

- 「PalmLsnNormalStart メソッド」 106 ページ

PalmLsnProcess メソッド

設定データベース内のレコードに従ってメッセージを処理します。

構文

```
palm_lsn_ret PalmLsnProcess(
   struct a_palm_msg * msg,
   Char const * configPDBName,
   UInt16 * const problematicRecNum,
   Boolean * handled
)
```

パラメータ

- **msg**　a_palm_msg インスタンスへのポインタ。
- **configPDBName**　設定データベースの名前を保持する文字配列。設定データベース名は、PalmLsnGetConfigFileName メソッドを使用して取得します。「PalmLsnGetConfigFileName メソッド」 104 ページを参照してください。
- **problematicRecNum**　設定データベース内の問題のあるレコードや間違った形式のレコードのインデックスを特定する出力パラメータ。
- **handled**　PalmLsnProcess が正常にメッセージを処理したかどうかを示す出力パラメータ。

戻り値

palm_lsn_ret 列挙体に定義されているリターン・コード。「palm_lsn_ret 列挙体」 97 ページを参照してください。

備考

PalmLsnProcess は、着信メッセージに対して実行すべき適切なアクションを決定します。メッセージの各フィールドを、設定データベース内に格納されているフィルタと比較します。

Listener 設定データベースの作成方法の詳細については、「Palm デバイス用の Listener 設定ユーティリティ」 87 ページを参照してください。

設定データベース内のレコードには、メッセージ・フィルタに関する情報と受け入れたメッセージに対して実行されるアクションが格納されています。

設定レコードの形式は次のとおりです。

```
[subject=<string>;] [content=<string>;]
[message|message_start=<string>;] [sender=<string>;]
action=run <app name> [arguments]
```

arguments はアプリケーションによって異なる文字列で、action 変数を含めることができます。

参照

- 「Palm デバイス用の Listener 設定ユーティリティ」 87 ページ
- 「メッセージ・ハンドラ」 19 ページ
- 「Windows 用の Listener アクション・コマンド」 80 ページ
- 「PalmLsnCheckConfigDB メソッド」 99 ページ
- 「メッセージ処理用メソッド」 94 ページ

例

以下に、メッセージを処理するコードの一部を示します。この例では、メッセージ構造体を割り当て、フィールドを初期化し、PalmLsnProcess を使用してメッセージを処理しています。

```
a_palm_msg * ulMsg;
Boolean * handled
Char configDb[dmDBNameLength];
...

// Allocate the message structure
ulMsg = PalmLsnAllocate();
...

// Fill the message fields
ret = PalmLsnDupMessage(ulMsg, msgBody);
...

// Get the configuration database name
PalmLsnGetConfigFileName(configDb);

// Process the message
ret = PalmLsnProcess(ulMsg, configDb, NULL, handled);
...

// Free the message
PalmLsnFree(ulMsg);
```

PalmLsnSpecialLaunch メソッド

デバイス依存の起動コードに応答します。

構文

```
Err PalmLsnSpecialLaunch(
   UInt16   cmd,
   MemPtr   cmdPBP,
   UInt16  launchFlags
)
```

パラメータ

- **cmd**　Palm OS アプリケーションの起動コード。
- **cmdPBP**　起動コード・パラメータが格納された構造体へのポインタ。起動コマンド固有のパラメータを持たないアプリケーションの場合は NULL。
- **launchFlags**　アプリケーションのステータス情報を示すフラグ。

戻り値

Palm OS のエラー・コード。errNone は正常終了を表します。

備考

このメソッドは、sysAppLaunchCmdNormalLaunch として定義されていない、デバイス依存または標準の起動コードに応答します。

例

次の例は、Treo 650 smartphone の実装での使用例です。PalmLsnSpecialLaunch を使用して Listener イベントを処理しています。

```
Err PalmLsnSpecialLaunch( UInt16 cmd, MemPtr cmdPBP, UInt16 /*launchFlags*/ ) {
   switch(cmd) {
      case sysAppLaunchCmdSystemReset:
         // Fall through
      case sysAppLaunchCmdSyncNotify:
         // Fall through
      case phnLibLaunchCmdRegister:
         return registerListener();
      case phnLibLaunchCmdEvent: {
         if (!IsFeatureOn(PalmLsnGetFeature(), Listening)) {
            return(errNone);
         }
         PhnEventPtr phoneEventP = (PhnEventPtr) cmdPBP;
         if (phoneEventP->eventType == phnEvtMessageInd) {
            return(handleMessage(phoneEventP->data.params.id, &phoneEventP->acknowledge));
         }
      }
      default:
         break;
   }
   return(errNone);
}
```

メッセージが検出されたら、handleMessage を使用してメッセージを処理し、適切なアクションを実行します。

```
static Err handleMessage( PhnDatabaseID id, Boolean * handled )
/*************************************************************/
// This routine will construct a_palm_msg and then call
// PalmLsnProcess to process it.
{
    a_palm_msg *     ulMsg;
    Err              ret;
    Boolean          newlyLoaded;
    PhnAddressList   addrList;
    PhnAddressHandle addrH;
    MemHandle        msgBodyH;
    Char *           msgSender;
    Char *           msgBody;
    UInt32           msgTime;
    Char             configDb[ dmDBNameLength ];
    UInt16           libRef = 0;

    // CDMA workaround recommended by Handspring
    DmOpenRef        openRef = 0;

    *handled = false;

    // Allocate a message structure for passing over
    // to PalmLsnProcess later
    ulMsg = PalmLsnAllocate();
    if (ulMsg == NULL) {
        return(sysErrNoFreeRAM);
    }

    // Load the phone library
    ret = findOrLoadPhoneLibrary(&libRef, &newlyLoaded);
    if (ret != errNone) {
        goto done;
    }
    openRef = PhnLibGetDBRef(libRef);

    // Retrieve sender of the message
    ret = PhnLibGetAddresses(libRef, id, &addrList);
    if (ret != errNone) {
        goto done;
    }
    ret = PhnLibGetNth(libRef, addrList, 1, &addrH);
    if (ret != errNone) {
        PhnLibDisposeAddressList(libRef, addrList);
        goto done;
    }

    msgSender = PhnLibGetField(libRef, addrH, phnAddrFldPhone);
    if (msgSender != NULL) {
        ret = PalmLsnDupSender(ulMsg, msgSender);
        MemPtrFree(msgSender);
    }
    PhnLibDisposeAddressList( libRef, addrList );
    if (ret != errNone) {
        goto done;
    }

    // Retrieve message time
    ret = PhnLibGetDate(libRef, id, &msgTime);
    if (ret != errNone) {
```

```
            goto done;
        }
        ret = PalmLsnDupTime(ulMsg, msgTime);
        if (ret != errNone) {
            goto done;
        }

        // Retrieve the entire message body
        ret = PhnLibGetText(libRef, id, &msgBodyH);
        if (ret != errNone) {
            goto done;
        }
        msgBody = (Char *) MemHandleLock(msgBodyH);
        ret = PalmLsnDupMessage(ulMsg, msgBody);

        // msgBodyH must be disposed of by the caller
        MemHandleUnlock(msgBodyH);
        MemHandleFree(msgBodyH);
        if (ret != errNone) {
            goto done;
        }

        // Get the configuration database name
        PalmLsnGetConfigFileName(configDb);

        // Call PalmLsnProcess to process the message
        ret = PalmLsnProcess(ulMsg, configDb, NULL, handled);

    done:
        if (ulMsg != NULL) {
            PalmLsnFree(ulMsg);
        }
        PhnLibReleaseDBRef(libRef, openRef);

        // Unload the phone library before any possible application switch
        if (newlyLoaded) {
            unloadPhoneLibrary(libRef);
            newlyLoaded = false;
        }
        return(ret);
    }
```


PalmLsnTargetCompanyID メソッド

デバイスの企業 ID または製造元 ID を返します。

構文

UInt32 **PalmLsnTargetCompanyID( )**

戻り値

デバイスの企業 ID または製造元 ID が含まれる値。

備考

PalmLsnTargetCompanyID と PalmLsnTargetDeviceID を使用して、デバイスに互換性があるかどうかをチェックします。

参照

- 「PalmLsnTargetDeviceID メソッド」 114 ページ

例

次に示す例は、Treo 650 smartphone の実装での使用例です。Handspring 社の企業 ID として 'hspr' を返しています。

```
UInt32 PalmLsnTargetCompanyID(void) {
   return('hspr');
}
```

PalmLsnTargetDeviceID メソッド

ターゲット・デバイス ID を返します。

構文

UInt32 **PalmLsnTargetDeviceID( )**

戻り値

デバイス ID を表す正の整数。

備考

PalmLsnTargetCompanyID と PalmLsnTargetDeviceID を使用すると、デバイスに互換性があるかどうかをチェックできます。

参照

- 「PalmLsnTargetCompanyID メソッド」 113 ページ

例

次の例では、Treo 650 シミュレータのデバイス ID を返しています。

```
UInt32 PalmLsnTargetDeviceID(void) {
   return(kPalmOneDeviceIDTreo650);
}
```


サーバ起動同期のシステム・プロシージャ

目次

サーバ起動同期のシステム・プロシージャは、Mobile Link システム・テーブル内のローの追加と削除を実行します。

注意
これらのシステム・プロシージャは、デバイス・トラッキングに使用します。自動デバイス・トラッキングをサポートするリモート・デバイスを使用する場合、これらのシステム・プロシージャを使用する必要はありません。自動デバイス・トラッキングをサポートしないリモート・デバイスを使用する場合、これらのシステム・プロシージャを使用して、手動のデバイス・トラッキングを設定できます。

参照

- 「デバイス・トラッキング・ゲートウェイ」 33 ページ
- 「デバイス・トラッキングのサポートの追加」 33 ページ
- 「Mobile Link サーバのシステム・テーブル」 『Mobile Link - サーバ管理』
- 「Mobile Link サーバ・システム・プロシージャ」 『Mobile Link - サーバ管理』

IBM DB2 メインフレームのサーバ起動同期システム・プロシージャの名前変換

DB2 メインフレームのバージョン 8.1 は下位互換性モードをサポートしており、カラム名とその他の識別子が最大 18 文字までに制限されています。この環境をサポートするためには、DB2 メインフレーム内のすべての Mobile Link システム・オブジェクトの名前を 18 文字以下にする必要があります。次の表は、DB2 メインフレーム統合データベースのプロシージャ名が、他のすべてのタイプの統合データベースのシステム・プロシージャ名にどのようにマッピングされるのかを示します。

以下の表にないシステム・プロシージャ名では、変換の必要はありません。

システム・プロシージャ名	DB2 メインフレーム統合データベースのシステム・プロシージャ名
「ml_delete_device_address システム・プロシージャ」 118 ページ	ml_del_dev_addr
「ml_delete_listening システム・プロシージャ」 119 ページ	ml_del_listen
「ml_set_device_address システム・プロシージャ」 122 ページ	ml_set_dev_addr
「ml_set_sis_sync_state システム・プロシージャ」 126 ページ	ml_set_sis_state

ml_delete_device システム・プロシージャ

手動でデバイス・トラッキングを設定している場合、このシステム・プロシージャを使用して、リモート・デバイスに関するすべての情報を削除します。

パラメータ

項目	パラメータ	説明
1	device	VARCHAR(255)。デバイス名。

備考

この機能は、デバイス・トラッキングを手動で設定する場合にだけ役立ちます。「デバイス・トラッキングのサポートの追加」 33 ページを参照してください。

例

デバイス・レコードとこれを参照するすべての関連レコードを削除します。

```
CALL ml_delete_device('myOldDevice');
```

ml_delete_device_address システム・プロシージャ

手動でデバイス・トラッキングを設定している場合、このシステム・プロシージャを使用して、デバイスのアドレスを削除します。

パラメータ

項目	パラメータ	説明
1	device	VARCHAR(255)
2	medium	VARCHAR(255)

備考

このシステム・プロシージャは、デバイス・トラッキングを手動で設定する場合にだけ役立ちます。「デバイス・トラッキングのサポートの追加」 33 ページを参照してください。

DB2 メインフレーム統合データベースの場合、このプロシージャは ml_del_dev_addr という名前になります。「IBM DB2 メインフレームのサーバ起動同期システム・プロシージャの名前変換」 116 ページを参照してください。

例

アドレス・レコードを削除します。

```
CALL ml_delete_device_address('myFirstTreo180', 'ROGERS AT&T');
```


ml_delete_listening システム・プロシージャ

手動でデバイス・トラッキングを設定している場合、このシステム・プロシージャを使用して、Mobile Link ユーザとリモート・デバイス間のマッピングを削除します。

パラメータ

項目	パラメータ	説明
1	ml_user	VARCHAR(128)

備考

このシステム・プロシージャは、デバイス・トラッキングを手動で設定する場合にだけ役立ちます。「デバイス・トラッキングのサポートの追加」 33 ページを参照してください。

DB2 メインフレーム統合データベースの場合、このプロシージャは ml_del_listen という名前になります。「IBM DB2 メインフレームのサーバ起動同期システム・プロシージャの名前変換」 116 ページを参照してください。

例

受信者レコードを削除します。

```
CALL ml_delete_listening('myULDB');
```

ml_set_device システム・プロシージャ

手動でデバイス・トラッキングを設定している場合、このシステム・プロシージャを使用して、リモート・デバイスに関する情報を追加または変更します。ml_device テーブル内のローを追加または更新します。

パラメータ

項目	パラメータ	説明
1	device	VARCHAR(255)。ユーザが定義したユニークなデバイス名。
2	listener_version	VARCHAR(128)。Listener のバージョンに関するオプションの注釈。
3	listener_protocol	INTEGER。バージョン 9.0.0 の場合は **0**、9.0.0 以降の Palm Listener は **1**、9.0.0 以降の Windows Listener は **2** を使用します。
4	info	VARCHAR(255)。オプションのデバイス情報。
5	ignore_tracking	CHAR(1)。トラッキングを無視し、手動で入力したデータがトラッキングによって上書きされないようにする場合、**y** に設定します。
6	source	VARCHAR(255)。このレコードのソースにあるオプションの注釈。

備考

システム・プロシージャ ml_set_device、ml_set_device_address、ml_set_listening は、Mobile Link システム・テーブル ml_device、ml_device_address、ml_listening にある情報を変更することで自動デバイス・トラッキングを上書きするのに使用します。たとえば、リモート・デバイスが Palm デバイスの場合、自動デバイス・トラッキングを使用できますが、手動で Palm デバイス用のデータを挿入します。

このシステム・プロシージャは、デバイス・トラッキングを手動で設定する場合にだけ役立ちます。「デバイス・トラッキングのサポートの追加」 33 ページを参照してください。

参照

- 「ml_set_device_address システム・プロシージャ」 122 ページ
- 「ml_set_listening システム・プロシージャ」 124 ページ
- 「ml_device」 『Mobile Link - サーバ管理』
- 「ml_device_address」 『Mobile Link - サーバ管理』
- 「ml_listening」 『Mobile Link - サーバ管理』

例

各デバイスについて、デバイス・レコードを追加します。

```
CALL ml_set_device(
   'myFirstTreo180',
   'MobiLink Listeners for Treo 180 - 9.0.1',
   '1',
   'not used',
   'y',
   'manually entered by administrator'
);
```

ml_set_device_address システム・プロシージャ

手動でデバイス・トラッキングを設定している場合、このシステム・プロシージャを使用して、リモート・デバイス・アドレスに関連する情報を追加または変更します。ml_device_address テーブル内のローを追加または更新します。

パラメータ

項目	パラメータ	説明
1	device	VARCHAR(255)。既存のデバイス名。
2	medium	VARCHAR(255)。ネットワーク・プロバイダ ID (Carrier の network_provider_id プロパティと一致する必要があります)。
3	address	VARCHAR(255)。SMS 対応デバイスの電話番号。
4	active	CHAR(1)。Push 通知の送信に使用するためにこのレコードをアクティブにする場合は、**y** に設定します。
5	ignore_tracking	CHAR(1)。トラッキングを無視し、手動で入力したデータがトラッキングによって上書きされないようにする場合、**y** に設定します。
6	source	VARCHAR(255)。このレコードのソースにあるオプションの注釈。

備考

システム・プロシージャ ml_set_device、ml_set_device_address、ml_set_listening は、Mobile Link システム・テーブル ml_device、ml_device_address、ml_listening にある情報を変更することで自動デバイス・トラッキングを上書きするのに使用します。たとえば、リモート・デバイスが Palm の場合、自動デバイス・トラッキングを使用できますが、Palm デバイス用のデータは手動で挿入します。

このシステム・プロシージャは、デバイス・トラッキングを手動で設定する場合にだけ役立ちます。「デバイス・トラッキングのサポートの追加」 33 ページを参照してください。

DB2 メインフレーム統合データベースの場合、このプロシージャは ml_set_dev_addr という名前になります。「IBM DB2 メインフレームのサーバ起動同期システム・プロシージャの名前変換」 116 ページを参照してください。

参照

- 「ml_set_device システム・プロシージャ」 120 ページ
- 「ml_set_listening システム・プロシージャ」 124 ページ
- 「ml_device」 『Mobile Link - サーバ管理』
- 「ml_device_address」 『Mobile Link - サーバ管理』
- 「ml_listening」 『Mobile Link - サーバ管理』

例

各デバイスについて、デバイスのアドレス・レコードを追加します。

```
CALL ml_set_device_address(
   'myFirstTreo180',
   'ROGERS AT&T',
   '3211234567',
   'y',
   'y',
   'manually entered by administrator'
);
```

ml_set_listening システム・プロシージャ

手動でデバイス・トラッキングを設定している場合、このシステム・プロシージャを使用して、Mobile Link ユーザとリモート・デバイス間のマッピングを追加または変更します。ml_listening テーブル内のローを追加または更新します。

パラメータ

項目	パラメータ	説明
1	ml_user	VARCHAR(128)。Mobile Link ユーザ名。
2	device	VARCHAR(255)。既存のデバイス名。
3	listening	CHAR(1)。DeviceTracker のアドレス設定に使用するためにこのレコードをアクティブにする場合、**y** に設定します。
4	ignore_tracking	CHAR(1)。トラッキングを無視し、手動で入力したデータがトラッキングによって上書きされないようにする場合、**y** に設定します。
5	source	VARCHAR(255)。このレコードのソースにあるオプションの注釈。

備考

システム・プロシージャ ml_set_device、ml_set_device_address、ml_set_listening は、Mobile Link システム・テーブル ml_device、ml_device_address、ml_listening にある情報を変更することで自動デバイス・トラッキングを上書きするのに使用します。たとえば、リモート・デバイスが Palm の場合、自動デバイス・トラッキングを使用できますが、手動で Palm デバイス用のデータを挿入します。

このシステム・プロシージャは、デバイス・トラッキングを手動で設定する場合にだけ役立ちます。「デバイス・トラッキングのサポートの追加」 33 ページを参照してください。

参照

- 「ml_set_device システム・プロシージャ」 120 ページ
- 「ml_set_device_address システム・プロシージャ」 122 ページ
- 「ml_device」 『Mobile Link - サーバ管理』
- 「ml_device_address」 『Mobile Link - サーバ管理』
- 「ml_listening」 『Mobile Link - サーバ管理』

例

各リモート・データベースについて、デバイスの受信者レコードを追加します。これは、デバイスを Mobile Link ユーザ名にマッピングします。

```
CALL ml_set_listening(
   'myULDB',
```



```
    'myFirstTreo180',
    'y',
    'y',
    'manually entered by administrator'
);
```

ml_set_sis_sync_state システム・プロシージャ

このシステム・プロシージャを使用して、Mobile Link 同期ステータスを ml_sis_sync_state システム・テーブルに記録します。

パラメータ

項目	パラメータ	説明
1	remote_id	VARCHAR(128)。
2	subscription_id	VARCHAR(255)。
3	publication_name	VARCHAR(128)。
4	user_name	VARCHAR(128)。
5	last_upload	TIMESTAMP。
6	last_download	TIMESTAMP。

備考

publication_nonblocking_download_ack イベントで ml_set_sis_sync_state システム・プロシージャを呼び出すと、ユーザは ml_sis_sync_state テーブルを参照する request_cursor イベントを作成できます。

DB2 メインフレーム統合データベースの場合、このプロシージャは ml_set_sis_state という名前になります。「IBM DB2 メインフレームのサーバ起動同期システム・プロシージャの名前変換」 116 ページを参照してください。

参照

- 「ml_sis_sync_state」 『Mobile Link - サーバ管理』
- 「publication_nonblocking_download_ack 接続イベント」 『Mobile Link - サーバ管理』

例

次のスクリプトでは、publication_nonblocking_download_ack イベント・スクリプトを指定して、同期ステータスを記録しています。

```
CALL ml_set_sis_sync_state(
   {ml s.remote_id},
   NULL,
   {ml s.publication_name},
   {ml s.username},
   NULL,
   {ml s.last_publication_download}
);
```


サーバ起動同期の高度なトピック

目次

メッセージ構文

ライトウェイト・ポーリング (デフォルト)、UDP ゲートウェイ、SYNC ゲートウェイには、次のメッセージ構文が適用されます。

message = [subject]content

SMTP ゲートウェイを使用して送信されるメッセージは、次のいずれかの構文構造をしています。

- **message** = *sender*[*subject*]*content*
- **message** = *sender*(*subject*)*content*
- **message** = *sender*{*subject*}*content*
- **message** = *sender*<*subject*>*content*
- **message** = *sender*'*subject*'*content*
- **message** = *sender*"*subject*"*content*

正しいメッセージ構文と *sender* の電子メール・アドレス構文は、各自の無線通信事業者によって異なります。メッセージ構文を判定するには、メッセージのロギングを有効にして Listener を実行します。このとき、冗長性レベルは dblsn の -m オプションと -v オプションを使用して 2 に設定します。最初に Listener を実行したときに、メッセージ・ログには正しい構文が記録されています。

デバイス・トラッキング・ゲートウェイを使用する場合、メッセージ構文はメッセージの送信に使用する従属ゲートウェイによって異なります。SMTP 従属ゲートウェイを使用する場合、構文は各自の公衆無線通信事業者によって異なります。

備考

大カッコ、シェブロン、二重引用符、カッコ、一重引用符、角カッコは、内部使用のために予約されています。**subject** 内では使用しないでください。メッセージの制限事項の詳細については、「Push 要求の使用」 12 ページを参照してください。

参照

- 「Windows 用の Listener キーワード」 77 ページ
- 「ゲートウェイと Carrier」 32 ページ
- 「-m オプション」 71 ページ
- 「-v オプション」 76 ページ

SA_SEND_UDP を使用した Push 通知の送信

SQL Anywhere 統合データベースを使用している場合は、SA_SEND_UDP システム・プロシージャを使用して、UDP ゲートウェイ経由でデバイスに Push 通知を送信できます。この方法は、Notifier を使用して Push 通知を送信する方法の代替となる手段です。

元のメッセージの最後に **1** を追加して、このメッセージを SA_SEND_UDP システム・プロシージャの **msg** 引数で使用すると、元のメッセージが Listener に送信されます。

♦ SA_SEND_UDP システム・プロシージャを使用して Push 通知を送信するには

このプロシージャは、デバイス上で Listener が設定済みで、Push 通知を受信していることを前提にしています。デバイス上では次のコマンドを使用します。

```
dblsn -l "message=RunBrowser;action='START iexplore.exe http://www.ianywhere.com';"
```

デモンストレーションのため、Listener が **RunBrowser** メッセージを受信するたびに Internet Explorer をロードすると想定します。また、このプロシージャでは、SQL Anywhere 統合データベースが Mobile Link サーバ上で稼働していることを想定しています。

1. Interactive SQL を実行し、統合データベースに接続します。
2. 次のコマンドを実行します。

   ```
   dbisql -c "dsn=consdb_source_name"
   ```

   consdb_source_name は、統合データベースの ODBC 名に置き換えます。
3. 次のコマンドを実行して、Push 通知を送信します。

   ```
   CALL SA_SEND_UDP('device_ip_address', 5001, 'RunBrowser1')
   ```

   最初の引数によって、Push 通知は必ず正しいデバイスに送信されます。*device_ip_address* は、デバイスの IP アドレスに置き換えます。Mobile Link サーバと同じコンピュータで Listener が実行されている場合は、**localhost** を使用してください。

   2 番目の引数はポート番号です。デフォルトでは、Listener はポート 5001 を使用して UDP を受信します。

   3 番目の引数は、最後に **1** を追加して送信するメッセージです。予約済みのサーバ起動同期プロトコルである 1 を追加すると、UDP ゲートウェイを使用して **RunBrowser** メッセージがデバイスに送信されます。

   システム呼び出しが実行されると、**RunBrowser** メッセージがデバイスに送信され、そのデバイスで Internet Explorer が起動して iAnywhere ホーム・ページがロードされます。

SA_SEND_UDP システム・プロシージャの詳細については、「sa_send_udp システム・プロシージャ」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

サーバ起動同期チュートリアル

目次

チュートリアル：ライトウェイト・ポーリングを使用したサーバ起動同期

Mobile Link サーバ起動同期チュートリアルの概要

このチュートリアルでは、サーバ起動同期を使用できるように SQL Anywhere 統合データベースとリモート・データベースを設定する方法について説明します。このチュートリアルは、*samples-dir¥MobiLink¥SIS_CarDealer_LP_DBLSN* に配置されているサンプル・コードに基づいています。

サーバ起動同期の実装サンプルのいくつかは、*samples-dir¥MobiLink* にあります。サーバ起動同期のすべてのサンプル・ディレクトリ名には、プレフィクスの SIS_ が付いています。

必要なソフトウェア

- SQL Anywhere 11

前提知識と経験

- Mobile Link イベント・スクリプトの基本的な知識

目的

- SQL Anywhere 統合データベースをサーバ起動同期用に設定する。
- サーバ側プロパティを設定する。
- サーバ起動同期を要求する Push 要求を発行する。

関連項目

- 「サーバ起動同期の概要」 2 ページ

レッスン 1：統合データベースの設定

このレッスンでは、dbinit ユーティリティを使用して、同期に必要なスクリプトで **SIS_CarDealer_LP_DBLSN_CONDB** という名前の統合データベースを作成します。

♦ **新しい SQL Anywhere 統合データベースを作成して起動するには、次の手順に従います。**

1. 新しい SQL Anywhere 統合データベースを作成します。

   コマンド・プロンプトで、dbinit ユーティリティを使用し、データベースのファイル名とパスを指定します。たとえば、次のコマンドを実行します。

   ```
   dbinit c:¥MLsis¥SIS_CarDealer_LP_DBLSN_CONDB.db
   ```

2. 統合データベースを起動します。

   ```
   dbeng11 c:¥MLsis¥SIS_CarDealer_LP_DBLSN_CONDB.db
   ```

SQL Anywhere 11 ドライバを使用して、データベース用の ODBC データ・ソースを定義します。

♦ **統合データベース用の ODBC データ・ソースを定義するには、次の手順に従います。**

1. **[スタート] - [プログラム] - [SQL Anywhere 11] - [ODBC アドミニストレータ]** を選択します。
2. **[ユーザー DSN]** タブをクリックし、**[追加]** をクリックします。
3. **[名前]** リストで **[SQL Anywhere 11]** をクリックします。**[完了]** をクリックします。
4. **[SQL Anywhere 11 の ODBC 設定]** ウィンドウで、次の操作を行います。
   a. **[ODBC]** タブをクリックします。
   b. **[データ・ソース名]** フィールドに **SIS_CarDealer_LP_DBLSN_CONDB** と入力します。
   c. **[ログイン]** タブをクリックします。
   d. **[ユーザ ID]** フィールドに **DBA** と入力します。
   e. **[パスワード]** フィールドに **sql** と入力します。
   f. **[データベース]** タブをクリックします。
   g. **[サーバ名]** フィールドに、**SIS_CarDealer_LP_DBLSN_CONDB** と入力します。
   h. **[データベース・ファイル]** フィールドに、*c:¥MLsis ¥SIS_CarDealer_LP_DBLSN_CONDB.db* と入力します。
   i. **[OK]** をクリックします。
5. **[OK]** をクリックします。

参照

● 「ODBC データ・ソースの作成」 『SQL Anywhere サーバ - データベース管理』

レッスン 2：データベース・スキーマの生成

このレッスンでは、1 つのデータベース・スキーマを生成します。このスキーマには、Dealer テーブル、non_sync_request テーブル、download_cursor 同期スクリプトが含まれます。このデータベース・スキーマは、Push 要求を生成するための稼働条件を満たしています。

♦ **データベース・スキーマを設定するには、次の手順に従います。**

1. 統合データベースに接続します。
   a. Sybase Central で、**[SQL Anywhere 11]** を右クリックし、**[接続]** を選択します。
   b. **[ID]** タブをクリックします。
   c. **[ODBC データ・ソース名]** をクリックし、**SIS_CarDealer_LP_DBLSN_CONDB** と入力します。**[OK]** をクリックします。
2. Interactive SQL を起動します。

   左ウィンドウ枠で、データベースを右クリックし、**[Interactive SQL を開く]** を選択します。
3. 次の文を実行し、Dealer テーブルと non_sync_request テーブルを作成して設定します。

```
CREATE TABLE Dealer (
   name          VARCHAR(10) NOT NULL PRIMARY KEY,
   rating        VARCHAR(5),
   last_modified  TIMESTAMP DEFAULT TIMESTAMP
)

CREATE TABLE non_sync_request(
   poll_key       varchar(128)
)
```

4. 次の文を使用して、Dealer テーブルにデータを挿入します。

```
INSERT INTO Dealer(name, rating) VALUES ('Audi', 'a');
INSERT INTO Dealer(name, rating) VALUES ('Buick', 'b');
INSERT INTO Dealer(name, rating) VALUES ('Chrysler', 'c');
INSERT INTO Dealer(name, rating) VALUES ('Dodge', 'd');
INSERT INTO Dealer(name, rating) VALUES ('Eagle', 'e');
INSERT INTO Dealer(name, rating) VALUES ('Ford', 'f');
INSERT INTO Dealer(name, rating) VALUES ('Geo', 'g');
INSERT INTO Dealer(name, rating) VALUES ('Honda', 'h');
INSERT INTO Dealer(name, rating) VALUES ('Isuzu', 'I');
COMMIT;
```

5. 次のコマンドを実行して Mobile Link のシステム・テーブルとストアド・プロシージャを作成します。*c:¥Program Files¥SQL Anywhere 11¥* は、SQL Anywhere 11 のインストール環境のディレクトリ名に置き換えてください。

```
read "c:¥Program Files¥SQL Anywhere 11¥MobiLink¥setup¥syncsa.sql"
```

6. 次の文を実行し、download_cursor 同期スクリプトを指定して ml_sis_sync_state システム・テーブルに同期ステータスを記録します。

```
CALL ml_add_table_script(
   'CarDealer',
   'Dealer',
   'download_cursor',
   'SELECT * FROM Dealer WHERE last_modified >= ?'
);
CALL ml_add_connection_script(
   'CarDealer',
   'publication_nonblocking_download_ack',
   'CALL ml_set_sis_sync_state(
            {ml s.remote_id},
      NULL,
            {ml s.publication_name},
            {ml s.username},
            NULL,
            {ml s.last_publication_download}
   )'
);
COMMIT;
```

   このスクリプトによって、ダウンロード専用同期を記録するように ml_sis_sync_state が設定されます。同期ステータスを記録すると、request_cursor イベントから ml_sis_sync_state システム・テーブルを参照できます。次のレッスンでは request_cursor イベントを指定します。

7. Interactive SQL を閉じます。

参照

- 「SQL Anywhere データベース・サーバ」『SQL Anywhere サーバ - データベース管理』
- 「CREATE TABLE 文」『SQL Anywhere サーバ - SQL リファレンス』
- 「同期スクリプトの作成」『Mobile Link - サーバ管理』
- 「download_cursor テーブル・イベント」『Mobile Link - サーバ管理』
- 「ml_sis_sync_state」『Mobile Link - サーバ管理』
- 「publication_nonblocking_download_ack 接続イベント」『Mobile Link - サーバ管理』

レッスン 3：Notifier の設定

このレッスンでは、Notifier イベントを設定して、Notifier が Push 要求を作成する方法と Push 通知をデバイスに送信する方法を定義します。

♦ **新しい Notifier を作成するには、次の手順に従います。**

1. Mobile Link 同期プラグインを使用して、統合データベースに接続します。
   a. Sybase Central を開きます。
   b. 左ウィンドウ枠で、**[Mobile Link 11]** をクリックします。
   c. **[モード]** - **[管理]** をクリックします。
   d. **[ファイル]** - **[接続]** をクリックします。
   e. **[ID]** タブをクリックします。
   f. **[ODBC データ・ソース名]** フィールドに **SIS_CarDealer_LP_DBLSN_CONDB** と入力します。
   g. **[OK]** をクリックします。
2. 左ウィンドウ枠で、**[通知]** フォルダをクリックします。
3. **[ファイル]** - **[新規]** - **[Notifier]** をクリックします。
4. [Notifier] フィールドで **CarDealerNotifier** を指定し、**[完了]** をクリックします。
5. request_cursor イベント・スクリプトを入力します。

   request_cursor イベント・スクリプトによって、Push 要求が検出されます。各 Push 要求によって、送信される情報と情報を受信するデバイスが決まります。
   a. 右ウィンドウ枠で、**CarDealerNotifier** を選択します。**[ファイル]** - **[プロパティ]** を選択します。
   b. **[イベント]** タブをクリックし、イベントの **[request_cursor]** を選択します。
   c. request_cursor スクリプトに、次のスクリプトを入力します。

```
SELECT ml_sis_sync_state.remote_id + '.sync' FROM ml_sis_sync_state
  WHERE (EXISTS (SELECT 1 FROM Dealer
  WHERE last_modifier >= ml_sis_sync_state.last_download)
  AND
  EXISTS(select poll_key FROM non_sync_request))
```

6. **[OK]** をクリックして Notifier イベントを保存します。

参照

- 「request_cursor イベント」 47 ページ
- 「ml_set_sis_sync_state システム・プロシージャ」 126 ページ

レッスン 4：Mobile Link サーバの起動

このレッスンでは、デバイスに Push 通知を送信できるように Notifier を使用して Mobile Link サーバを起動します。

♦ **Mobile Link サーバ (mlsrv11) を実行するには、次の手順に従います。**

- コマンド・プロンプトで、統合データベースのディレクトリに移動します。次のコマンドを 1 行で入力します。

```
mlsrv11 -notifier -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB" -o ml.log -fr -v+ -zu+ -x tcpip
```

次の表は、mlsrv11 ユーティリティで使用できるオプションを示しています。オプション -o、-v、は、デバッグとトラブルシューティングの情報を提供します。これらのロギング・オプションは、開発環境での使用に適しています。パフォーマンス上の理由から、一般的に -v は運用環境では使用しません。

オプション	説明
-notifier	サーバ起動同期用に有効なすべての Notifier を起動します。 「-notifier オプション」 『Mobile Link - サーバ管理』を参照してください。
-c	接続文字列を指定します。 「-c オプション」 『Mobile Link - サーバ管理』を参照してください。
-o	メッセージ・ログ・ファイル *ml.log* を指定します。 「-o オプション」 『Mobile Link - サーバ管理』を参照してください。
-fr	データをアップロードするスクリプトとダウンロードするスクリプトが 1 つも同期に含まれない場合に、Mobile Link サーバがアボートしないようにします。このオプションは、ダウンロードのみの同期を使用するこのチュートリアルでは必須です。 「-fr オプション」 『Mobile Link - サーバ管理』を参照してください。
-v+	-v オプションは、ログを取る情報を指定します。-v+ を使用して、最大冗長ロギングをオンに設定します。 「-v オプション」 『Mobile Link - サーバ管理』を参照してください。

オプション	説明
-zu+	自動的に新しいユーザを追加します。 「-zu オプション」『Mobile Link - サーバ管理』を参照してください。
-x	Mobile Link クライアントの通信プロトコルとプロトコル・オプションを設定します。 「-x オプション」『Mobile Link - サーバ管理』を参照してください。

Mobile Link サーバが動作中であることを示すウィンドウが表示されます。Notifier は、デバイスから Push 要求を受信する準備が整ったことを示します。

参照

このレッスンのトピックの詳細については、次の各項を参照してください。

- 「Mobile Link サーバ」『Mobile Link - サーバ管理』
- 「Mobile Link サーバ・オプション」『Mobile Link - サーバ管理』

レッスン 5：リモート・データベースの設定

このレッスンでは、SQL Anywhere リモート・データベースを作成し、同期パブリケーション、ユーザ、サブスクリプションを作成します。また、Listener のコマンド・ファイルを作成し、Listener を起動します。

♦ 新しい SQL Anywhere リモート・データベースを作成して設定するには、次の手順に従います。

1. コマンド・プロンプトで、データベースを作成するディレクトリに移動します。
2. 次のコマンドを入力して、データベースを作成します。

   ```
   dbinit c:¥MLsis¥SIS_CarDealer_LP_DBLSN_REM.db
   ```

3. 次のコマンドを入力して、データベースを起動します。

   ```
   dbeng11 c:¥MLsis¥SIS_CarDealer_LP_DBLSN_REM.db
   ```

4. リモート・データベース・スキーマを生成します。
   a. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Sybase Central]** をクリックします
   b. 左ウィンドウ枠で、**[SQL Anywhere 11]** をクリックします。
   c. **[ファイル] - [接続]** をクリックします。
   d. **[ID]** タブをクリックします。
   e. **[ユーザ ID]** フィールドに **DBA** と入力します。
   f. **[パスワード]** フィールドに **sql** と入力します。
   g. **[データベース]** タブをクリックします。

h. **[サーバ名]** フィールドに、**SIS_CarDealer_LP_DBLSN_REM** と入力します。

i. **[OK]** をクリックします。

j. **[ファイル] - [Interactive SQL を開く]** をクリックします。

k. 次のコマンドを実行して、Dealer テーブルを作成します。

```
CREATE TABLE Dealer (
   name          VARCHAR(10) NOT NULL PRIMARY KEY,
   rating        VARCHAR(5),
   last_modified  TIMESTAMP DEFAULT TIMESTAMP
)
COMMIT;
```

5. 次のコマンドを実行して、同期パブリケーション、同期ユーザ、同期サブスクリプションを作成します。

```
CREATE PUBLICATION CarDealer(TABLE DEALER WHERE 0=1)
CREATE SYNCHRONIZATION USER test_mluser OPTION ScriptVersion='CarDealer'
CREATE SYNCHRONIZATION SUBSCRIPTION TO CarDealer FOR test_mluser
SET OPTION public.ml_remote_id = remote_id;
COMMIT;
```

参照

- 「Mobile Link クライアント」『Mobile Link - クライアント管理』
- 「CREATE TABLE 文」『SQL Anywhere サーバ - SQL リファレンス』
- 「データのパブリッシュ」『Mobile Link - クライアント管理』
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『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』
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「スクリプト・バージョン」『Mobile Link - サーバ管理』

レッスン 6：Listener の設定

このレッスンでは、Listener オプションをテキスト・ファイルに保存してから、コマンド・ラインでファイル名を指定して dblsn を実行し、Listener を設定します。

♦ **Listener を設定するには、次の手順に従います。**

1. 次のコマンドを実行して Mobile Link サーバと同期し、*SIS_CarDealer_LP_DBLSN_REM.rid* ファイルを作成します。

```
dbmlsync -c filedsn=SIS_CarDealer_LP_DBLSN_REM.dsn -ot dbmlsync.log -qc -e sa=on
```

Listener は、$remote_id action 変数を使用してポーリング・キーを定義できます。このキーは、Mobile Link サーバでデバイスの識別に使用されます。この変数は、リモート ID ファイル *SIS_CarDealer_LP_DBLSN_REM.rid* から取得します。このファイルは、Mobile Link サーバと

初期同期するときに作成されます。リモート ID ファイルを使用する場合は、Mobile Link サーバと同期する必要があります。

2. 次の内容の *mydblsn.txt* というテキスト・ファイルを作成します。

```
# Verbosity level
-v2

# Show notification messages in console and log
-m

# Truncate, then write output to dblsn.log
-ot dblsn.log

# Remote ID file (defining the scope of $remote_id)
-r SIS_CarDealer_LP_DBLSN_REM.rid

# Message handlers

# Watch for a notification without action
-l "poll_connect=tcpip(host=localhost);
    poll_notifier=CarDealerNotifier;
    poll_key=$remote_id.no_action;"

# Signal dbmlsync to launch, sync and then shutdown
-l "poll_connect=tcpip(host=localhost);
    poll_notifier=CarDealerNotifier;
    poll_key=$remote_id.sync;
    action='run dbmlsync.exe -c filedsn=SIS_CarDealer_LP_DBLSN_REM.dsn -ot dbmlsync.log -qc -
e sa=on';"

# Shutdown the listener
-l "poll_connect=tcpip(host=localhost);
    poll_notifier=CarDealerNotifier;
    poll_key=$remote_id.shutdown;
    action='DBLSN FULL SHUTDOWN';"
```

3. *mydblsn.txt* ファイルを保存します。
4. Listener を起動します。

   コマンド・プロンプトで、Listener コマンド・ファイルのディレクトリに移動します。

   次のように入力して Listener を起動します。

```
dblsn @mydblsn.txt
```

   Listener が動作中であることを示すウィンドウが表示されます。

参照

- 「Listener」 19 ページ
- 「Windows デバイス用の Listener ユーティリティ」 64 ページ
- 「@ option」 67 ページ
- 「action 変数」 22 ページ

レッスン 7：Push 要求の発行

このレッスンでは、統合データベースの Dealer テーブルを変更して、Listener が Push 通知をポーリングするときに情報をリモート・データベースにダウンロードできるようにします。次に、統合データベースにポーリング・キー値を挿入して、サーバ起動同期を要求します。Notifier は request_cursor イベントを実行し、non_sync_request テーブル内のポーリング・キーを検出して Listener に Push 通知を送信します。Listener が Push 通知を受信すると、Mobile Link データベースと同期してリモート・データベースを更新します。

♦ 統合データベースを変更するには、次の手順に従います。

1. Interactive SQL を通じて **SIS_CarDealer_LP_DBLSN_CONDB** データベースに接続します。
2. 次のスクリプトを入力します。

```
UPDATE Dealer
   SET RATING = 'B' WHERE name = 'Geo';
COMMIT;
```

Push 要求を発行するには、non_sync_request テーブルに直接移植します。ポーリング・キー・カラムによって、Push 通知を受信するデバイスが決まります。

♦ サーバ起動同期を要求するには、次の手順に従います。

1. Interactive SQL を通じて **SIS_CarDealer_LP_DBLSN_CONDB** データベースに接続します。
2. 次のスクリプトを入力します。

```
INSERT INTO non_sync_request(poll_key) VALUES ('%remote_id%.no_action');
COMMIT;
```

3. 同期が発生するまで数秒待ちます。

   Listener は、統合データベースをポーリングして Push 通知をダウンロードし、リモート・データベースの Dealer テーブルを更新します。

デバイスとのサーバ起動同期を停止するには、non_sync_request テーブルからポーリング・キー値を削除します。

♦ サーバ起動同期を停止するには、次の手順に従います。

1. Interactive SQL を通じて **SIS_CarDealer_LP_DBLSN_CONDB** データベースに接続します。
2. 次のスクリプトを入力します。

```
DELETE FROM non_sync_request WHERE poll_key = '%remote_id%.no_action';
COMMIT;
```

参照

- 「Push 要求の生成」 12 ページ
- 「INSERT 文」 『SQL Anywhere サーバ - SQL リファレンス』
- 「UPDATE 文」 『SQL Anywhere サーバ - SQL リファレンス』
- 「DELETE 文」 『SQL Anywhere サーバ - SQL リファレンス』

チュートリアルの削除

チュートリアルをコンピュータから削除します。

♦ **チュートリアルをコンピュータから削除するには、次の手順に従います。**

1. Interactive SQL を閉じます。
2. SQL Anywhere、Mobile Link、同期クライアントの各ウィンドウを閉じます。
3. チュートリアルに関連するすべての DSN を削除します。
   a. ODBC アドミニストレータを起動します。
      コマンド・プロンプトで次のコマンドを入力します。

      ```
      odbcad32
      ```

   b. **SIS_CarDealer_LP_DBLSN_CONDB** データ・ソースを削除します。
4. 統合データベースとリモート・データベースが保存されているディレクトリ *c:¥mlsis¥* に移動し、すべてのファイルを削除します。

チュートリアル：ゲートウェイを使用したサーバ起動同期

Mobile Link サーバ起動同期チュートリアルの概要

このチュートリアルでは、サーバ起動同期を使用できるように SQL Anywhere 統合データベースとリモート・データベースを設定する方法について説明します。このチュートリアルは、*samples-dir¥MobiLink¥SIS_CarDealer* に配置されているサンプル・コードに基づいています。

サーバ起動同期の実装サンプルのいくつかは、*samples-dir¥MobiLink* にあります。サーバ起動同期のすべてのサンプル・ディレクトリ名には、プレフィクスの SIS_ が付いています。

必要なソフトウェア

- SQL Anywhere 11

前提知識と経験

- Mobile Link イベント・スクリプトの基本的な知識

目的

- SQL Anywhere 統合データベースをサーバ起動同期用に設定する。
- サーバ側プロパティを設定する。
- サーバ起動同期を要求する Push 要求を発行する。

関連項目

- 「サーバ起動同期の概要」 2 ページ

レッスン 1：統合データベースの設定

このレッスンでは、dbinit ユーティリティを使用して、同期に必要なスクリプトで統合データベースを作成します。

♦ **新しい SQL Anywhere 統合データベースを作成して起動するには、次の手順に従います。**

1. 新しい SQL Anywhere 統合データベースを作成します。

   コマンド・プロンプトで、dbinit ユーティリティを使用し、データベースのファイル名とパスを指定します。たとえば、次のコマンドを実行します。

   ```
   dbinit c:¥MLsis¥MLconsolidated.db
   ```

2. 統合データベースを起動します。

   ```
   dbeng11 c:¥MLsis¥MLconsolidated.db
   ```

SQL Anywhere 11 ドライバを使用して、データベース用の ODBC データ・ソースを定義します。

♦ **統合データベース用の ODBC データ・ソースを定義するには、次の手順に従います。**

1. **[スタート] - [プログラム] - [SQL Anywhere 11] - [ODBC アドミニストレータ]** を選択します。
2. **[ユーザー DSN]** タブをクリックし、**[追加]** をクリックします。
3. **[名前]** リストで **[SQL Anywhere 11]** をクリックします。**[完了]** をクリックします。
4. **[SQL Anywhere 11 の ODBC 設定]** ウィンドウで、次の操作を行います。
   a. **[ODBC]** タブをクリックします。
   b. **[データ・ソース名]** フィールドに **sis_cons** と入力します。
   c. **[ログイン]** タブをクリックします。
   d. **[ユーザ ID]** フィールドに **DBA** と入力します。
   e. **[パスワード]** フィールドに **sql** と入力します。
   f. **[データベース]** タブをクリックします。
   g. **[サーバ名]** フィールドに、**MLconsolidated** と入力します。
   h. **[データベース・ファイル]** フィールドに *c:¥MLsis¥MLconsolidated.db* と入力します。
   i. **[OK]** をクリックします。
5. **[OK]** をクリックします。

参照

● 「ODBC データ・ソースの作成」『SQL Anywhere サーバ - データベース管理』

レッスン 2：データベース・スキーマの生成

統合データベース・スキーマには、Dealer テーブル、download_cursor 同期スクリプト、サーバ起動同期の Push 要求を生成するテーブルとストアド・プロシージャが含まれます。

♦ **Dealer テーブルと download_cursor 同期スクリプトを追加するには、次の手順に従います。**

1. 統合データベースに接続します。
   a. Sybase Central で、**[SQL Anywhere 11]** を右クリックし、**[接続]** を選択します。
   b. **[ID]** タブをクリックします。
   c. **[ODBC データ・ソース名]** をクリックし、**sis_cons** と入力します。**[OK]** をクリックします。
2. Interactive SQL を起動します。

   左ウィンドウ枠で、データベースを右クリックし、**[Interactive SQL を開く]** を選択します。
3. 次の文を実行して、Dealer テーブルと download_cursor 同期スクリプトをインストールします。

```
CREATE TABLE Dealer (
   name varchar(10) NOT NULL PRIMARY KEY,
```

```
    rating VARCHAR(5),
    last_modified TIMESTAMP DEFAULT TIMESTAMP
)
INSERT INTO Dealer(name, rating) VALUES ('Audi', 'a');
INSERT INTO Dealer(name, rating) VALUES ('Buick', 'b');
INSERT INTO Dealer(name, rating) VALUES ('Chrysler', 'c');
INSERT INTO Dealer(name, rating) VALUES ('Dodge', 'd');
INSERT INTO Dealer(name, rating) VALUES ('Eagle', 'e');
INSERT INTO Dealer(name, rating) VALUES ('Ford', 'f');
INSERT INTO Dealer(name, rating) VALUES ('Geo', 'g');
INSERT INTO Dealer(name, rating) VALUES ('Honda', 'h');
INSERT INTO Dealer(name, rating) VALUES ('Isuzu', 'I');
COMMIT;
```

4. 次のコマンドを実行して Mobile Link のシステム・テーブルとストアド・プロシージャを作成します。*c:¥Program Files¥SQL Anywhere 11¥* は、SQL Anywhere 11 のインストール環境のディレクトリ名に置き換えてください。

```
read "c:¥Program Files¥SQL Anywhere 11¥MobiLink¥setup¥syncsa.sql"
```

5. 次のコマンドを実行します。

```
CALL ml_add_table_script(
    'sis_ver1',
    'Dealer',
    'download_cursor',
    'SELECT * FROM Dealer WHERE last_modified >= ?'
)
```

参照

- 「SQL Anywhere データベース・サーバ」『SQL Anywhere サーバ - データベース管理』
- 「CREATE TABLE 文」『SQL Anywhere サーバ - SQL リファレンス』
- 「同期スクリプトの作成」『Mobile Link - サーバ管理』
- 「download_cursor テーブル・イベント」『Mobile Link - サーバ管理』

レッスン 3：Push 要求の構成

Notifier は、Push 要求を検出すると、デバイスにメッセージを送信します。

♦ **Push 要求を格納する簡単なテーブルを作成するには、次の手順に従います。**

1. Interactive SQL で次のコマンドを実行します。

```
CREATE TABLE PushRequest (
    req_id INTEGER DEFAULT AUTOINCREMENT PRIMARY KEY,
    mluser VARCHAR(128),
    subject VARCHAR(128),
    content VARCHAR(128),
    resend_interval VARCHAR(30) DEFAULT '20s',
    time_to_live VARCHAR(30) DEFAULT '1m',
    status VARCHAR(128) DEFAULT 'created'
)
COMMIT;
```

2. Interactive SQL を閉じます。

参照

- 「Push 要求」 10 ページ
- 「Mobile Link サーバ起動同期の概要」 1 ページ
- 「サーバ起動同期のコンポーネント」 4 ページ

レッスン 4：Notifier の設定

このレッスンでは、Notifier で Push 要求を作成し、要求をリモート Listener に送信し、有効期限が切れた要求を検出する方法に影響する、3 つの Notifier イベントを設定します。

♦ **Notifier プロパティとイベントを設定するには、次の手順に従います。**

1. Mobile Link 同期プラグインを使用して、統合データベースに接続します。
   a. Sybase Central を開きます。
   b. 左ウィンドウ枠で、**[Mobile Link 11]** をクリックします。
   c. **[モード]** - **[管理]** をクリックします。
   d. **[ファイル]** - **[接続]** をクリックします。
   e. **[ID]** タブをクリックします。
   f. **[ODBC データ・ソース名]** フィールドに **sis_cons** と入力します。
   g. **[OK]** をクリックします。
2. 左ウィンドウ枠で、**[通知]** フォルダをクリックします。
3. **[ファイル]** - **[新規]** - **[Notifier]** をクリックします。
4. [Notifier] フィールドで **CarDealerNotifier** を指定し、**[完了]** をクリックします。
5. begin_poll イベント・スクリプトを入力します。

   Notifier は、統合データベース内の変更を検出し、begin_poll イベントを使用して Push 要求を作成します。この場合、変更が Dealer テーブルで発生し、リモート・データベースが最新でない場合、begin_poll スクリプトは、PushRequest テーブルを設定します。
   a. 右ウィンドウ枠で、[CarDealerNotifier] を選択します。**[ファイル]** - **[プロパティ]** を選択します。
   b. **[イベント]** タブをクリックし、イベントの begin_poll を選択します。
   c. **[begin_poll]** スクリプトに次の内容を入力します。

```
--
-- Insert the last consolidated database
-- modification date into @last_modified
--
DECLARE @last_modified timestamp;
SELECT MAX(last_modified) INTO @last_modified FROM Dealer;

--
-- Delete processed requests if the mluser is up-to-date
--
DELETE FROM PushRequest
```

```
    FROM PushRequest AS p, ml_user AS u, ml_subscription AS s
    WHERE p.status = 'processed'
      AND u.name = p.mluser
      AND u.user_id = s.user_id
      AND @last_modified <= GREATER(s.last_upload_time, s.last_download_time);

--
-- Insert new requests when a device is not up-to-date
--
INSERT INTO PushRequest(mluser, subject, content)
SELECT u.name, 'sync', 'ignored'
    FROM ml_user as u, ml_subscription as s
    WHERE u.name IN (SELECT name FROM ml_listening WHERE listening = 'y')
      AND u.user_id = s.user_id
      AND @last_modified > greater(s.last_upload_time, s.last_download_time)
      AND u.name NOT LIKE '%-dblsn'
      AND NOT EXISTS(SELECT * FROM PushRequest
        WHERE PushRequest.mluser = u.name
          AND PushRequest.subject = 'sync')
```

begin_poll スクリプトの最初の主要セクションでは、デバイスが最新の状態でない場合は、PushRequest テーブルからの処理済み要求は削除されます。

```
@last_modified <= GREATER(s.last_upload_time, s.last_download_time)
```

@last_modified は、統合データベース Dealer テーブルで最大の変更が行われた日です。式 greater(s.last_upload_time, s.last_download_time) は、リモート・データベースの最後の同期時間を表します。

request_delete イベントを使用して、直接 Push 要求を削除することもできます。ただし、この場合に begin_poll イベントを使用すると、リモート・データベースが同期する前に、同期期限が切れた要求や暗黙的に除外された要求が削除されないようにすることができます。

コードの次のセクションは、Dealer テーブルの last_modified カラムに加えられた変更をチェックし、ml_listening テーブルにリストされた最新の状態にないすべてのアクティブ Listener に対して Push 要求を発行します。

```
@last_modified > GREATER(s.last_upload_time, s.last_download_time)
```

PushRequest テーブルが移植されると、begin_poll スクリプトが件名を 'sync' に設定します。

6. request_cursor スクリプトを入力します。

   request_cursor スクリプトは、Push 要求をフェッチします。各 Push 要求により、メッセージで送信される情報、情報を受信するリモート・データベースが決まります。

   a. イベントの **[request_cursor]** を選択します。

   b. 表示された領域で、request_cursor スクリプトの次のコードを入力します。

```
SELECT
    p.req_id,
    'Default-DeviceTracker',
    p.subject,
    p.content,
    p.mluser,
    p.resend_interval,
    p.time_to_live
    FROM PushRequest AS p
```

   PushRequest テーブルによって、request_cursor スクリプトにローが入力されます。

順序と request_cursor 結果セットの値は重要です。たとえば、2 番目のパラメータは、デフォルトのゲートウェイ Default-DeviceTracker を定義します。デバイス・トラッキング・ゲートウェイは、ユーザへのアクセス方法を追跡し、UDP または SMTP を自動的に選択してリモート・デバイスに接続します。

7. request_delete スクリプトを入力します。

   request_delete Notifier イベントはクリーンアップ処理を指定します。このスクリプトを使用すると、暗黙に除外された要求、期限が切れた要求が自動的に削除されます。

   a. イベントの **[request_delete]** を選択します。

   b. 表示された領域で、request_delete スクリプトに次のコードを入力します。

      ```
      UPDATE PushRequest SET status='processed' WHERE req_id = ?
      ```

   request_delete スクリプトは、ローを削除するのではなく、PushRequest テーブルのローのステータスを 'processed' に更新します。

8. **[OK]** をクリックして Notifier プロパティを保存します。

参照

- 「request_delete イベント」 48 ページ
- 「begin_poll イベント」 43 ページ
- 「ml_listening」『Mobile Link - サーバ管理』
- 「デバイス・トラッキング・ゲートウェイ」 33 ページ
- 「request_cursor イベント」 47 ページ
- 「request_delete イベント」 48 ページ

レッスン 5：ゲートウェイと Carrier の設定

ゲートウェイは、メッセージを送信するためのメカニズムです。UDP ゲートウェイと SMTP ゲートウェイを定義できます。また、デバイス・トラッキング・ゲートウェイも使用できます。デバイス追跡により、Mobile Link はユーザへのアクセス方法を追跡して、UDP または SMTP のどちらのゲートウェイを使用するかを自動的に決定します。

このチュートリアルでは、デフォルトのデバイス・トラッキング・ゲートウェイを使用するため、設定は必要ありません。

参照

- 「デバイス・トラッキング・ゲートウェイ・プロパティ」 57 ページ
- 「ゲートウェイと Carrier」 32 ページ

レッスン 6：Mobile Link サーバの起動

このレッスンでは、デバイスに Push 通知を送信できるように Notifier を使用して Mobile Link サーバを起動します。

♦ **Mobile Link サーバ (mlsrv11) を実行するには、次の手順に従います。**

● コマンド・プロンプトで、統合データベース・ディレクトリに移動して、次のように入力します。

```
mlsrv11 -notifier -c "dsn=sis_cons" -o ml.log -fr -v+ -zu+ -x tcpip
```

次の表は、mlsrv11 ユーティリティで使用できるオプションを示しています。オプション -o、-v、は、デバッグとトラブルシューティングの情報を提供します。これらのロギング・オプションは、開発環境での使用に適しています。パフォーマンス上の理由から、一般的に -v は運用環境では使用しません。

オプション	説明
-notifier	サーバ起動同期用に有効なすべての Notifier を起動します。 「-notifier オプション」 『Mobile Link - サーバ管理』を参照してください。
-c	接続文字列を指定します。 「-c オプション」 『Mobile Link - サーバ管理』を参照してください。
-o	メッセージ・ログ・ファイル *ml.log* を指定します。 「-o オプション」 『Mobile Link - サーバ管理』を参照してください。
-fr	データをアップロードするスクリプトとダウンロードするスクリプトが 1 つも同期に含まれない場合に、Mobile Link サーバがアボートしないようにします。このオプションは、ダウンロードのみの同期を使用するこのチュートリアルでは必須です。 「-fr オプション」 『Mobile Link - サーバ管理』を参照してください。
-v+	-v オプションは、ログを取る情報を指定します。-v+ を使用して、最大冗長ロギングをオンに設定します。 「-v オプション」 『Mobile Link - サーバ管理』を参照してください。
-zu+	自動的に新しいユーザを追加します。 「-zu オプション」 『Mobile Link - サーバ管理』を参照してください。
-x	Mobile Link クライアントの通信プロトコルとプロトコル・オプションを設定します。 「-x オプション」 『Mobile Link - サーバ管理』を参照してください。

Mobile Link サーバが要求を処理する準備ができたことを示すウィンドウが表示されます。また、Notifier も表示されます。

参照

このレッスンのトピックの詳細については、次の各項を参照してください。

- 「Mobile Link サーバ」『Mobile Link - サーバ管理』
- 「Mobile Link サーバ・オプション」『Mobile Link - サーバ管理』

レッスン 7：リモート・データベースの設定

このレッスンでは、SQL Anywhere リモート・データベースを作成し、同期パブリケーション、ユーザ、サブスクリプションを作成します。また、Listener のコマンド・ファイルを作成し、Listener を起動します。

♦ 新しい SQL Anywhere リモート・データベースを作成して起動するには、次の手順に従います。

1. コマンド・プロンプトで、データベースを作成するディレクトリに移動します。
2. 次のコマンドを入力して、データベースを作成します。

   ```
   dbinit c:¥MLsis¥rem1.db
   ```

3. 次のコマンドを入力して、データベースを起動します。

   ```
   dbeng11 c:¥MLsis¥rem1.db
   ```

4. リモート・データベース・スキーマを生成します。
   a. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Sybase Central]** をクリックします
   b. 左ウィンドウ枠で、**[SQL Anywhere 11]** をクリックします。
   c. **[ファイル] - [接続]** をクリックします。
   d. **[ID]** タブをクリックします。
   e. **[ユーザ ID]** フィールドに **DBA** と入力します。
   f. **[パスワード]** フィールドに **sql** と入力します。
   g. **[データベース]** タブをクリックします。
   h. **[サーバ名]** フィールドに、**rem1** と入力します。
   i. **[OK]** をクリックします。
   j. **[ファイル] - [Interactive SQL を開く]** をクリックします。
   k. 次のコマンドを実行して、Dealer テーブルを作成します。

      ```
      CREATE TABLE Dealer (
         name VARCHAR(10) NOT NULL PRIMARY KEY,
         rating VARCHAR(5),
         last_modified TIMESTAMP DEFAULT TIMESTAMP
      )
      COMMIT;
      ```

5. 次のコマンドを実行して、同期パブリケーション、同期ユーザ、同期サブスクリプションを作成します。

```
CREATE PUBLICATION car_dealer_pub (table Dealer);
CREATE SYNCHRONIZATION USER sis_user1;
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO car_dealer_pub
   FOR sis_user1
   OPTION scriptversion='sis_ver1';
```

参照

- 「Mobile Link クライアント」『Mobile Link - クライアント管理』
- 「CREATE TABLE 文」『SQL Anywhere サーバ - SQL リファレンス』
- 「データのパブリッシュ」『Mobile Link - クライアント管理』
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『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』
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「スクリプト・バージョン」『Mobile Link - サーバ管理』

レッスン 8：Listener の設定

Listener は、リモート・デバイスで実行されます。Listener は、Notifier からのメッセージを受信して、アクションを実行します。たとえば、次の dblsn オプションを指定した場合、Listener はメッセージ **sync** を受信すると dbmlsync を起動します。

```
-l "subject=sync;action='run dbmlsync.exe...'
```

Listener を設定するには、コマンド・ライン・オプションをテキスト・ファイルに格納すると便利です。たとえば、設定を *mydblsn.txt* に格納し、次のように入力して Listener を起動します。

```
dblsn @mydblsn.txt
```

また、パラメータを指定しないで dblsn と入力すると、デフォルトの引数ファイルとして *dblsn.txt* が使用されます。

♦ **Mobile Link Listener を作成して起動するには、次の手順に従います。**

1. 次の内容の *mydblsn.txt* というテキスト・ファイルを作成します。

```
#-----------------------------------
# Verbosity level
-v2

# Show notification messages in console and log
-m

# Polling interval, in seconds
-i 3

# Truncate, then write output to dblsn.log
-ot dblsn.log

# MobiLink address and connect parameter for dblsn
-x "host=localhost"
```



```
# Enable device tracking and specify the MobiLink user name.
-t+ sis_user1

# Message handlers
# Synchronize using dbmlsync
-l "subject=sync;
action='start dbmlsync.exe
 -c eng=rem1;uid=DBA;pwd=sql
 -ot dbmlsyncOut.txt -qc';"
```

2. *mydblsn.txt* ファイルを保存します。
3. Listener を起動します。
   a. コマンド・プロンプトで、Listener コマンド・ファイルのディレクトリに移動します。
   b. 次のように入力して Listener を起動します。

      ```
      dblsn @mydblsn.txt
      ```

   Listener が起動してデバイス・トラッキング情報を Mobile Link サーバにアップロードしたことを示すウィンドウが表示されます。

トラッキング情報が統合データベースにアップロードされると、Mobile Link サーバ・ウィンドウに新しいエントリが表示されます。この情報は、Listener と Mobile Link サーバ間の正常な初期通信をリレーします。

参照

- 「Listener」 19 ページ
- 「Windows デバイス用の Listener ユーティリティ」 64 ページ
- 「@ option」 67 ページ

レッスン 9：Push 要求の発行

サーバ起動同期では、直接 PushRequest テーブルを移植するか、Dealer テーブルで変更を加えることで、Push 要求を発行することができます。後者の場合、Notifier の begin_poll スクリプトは、Dealer テーブルの変更を検出して、PushRequest テーブルを移植します。

どちらの場合も、PushRequest テーブルが Notifier の request_cursor スクリプトにローを入力します。これによって、リモート・デバイスでメッセージを受信する方法が決まります。

♦ 直接 Push 要求を PushRequest テーブルに挿入してサーバ起動同期を要求するには、次の手順に従います。

1. Interactive SQL で、*MLconsolidated.db* データベースに接続して、次の文を入力します。

   ```
   INSERT INTO PushRequest(mluser, subject, content)
      VALUES ('sis_user1', 'sync', 'not used');
   COMMIT;
   ```

2. 同期が発生するまで数秒待ちます。

移植すると、PushRequest テーブルは Notifier の request_cursor スクリプトにローを入力します。request_cursor スクリプトは、メッセージで送信される情報と、情報を受信するリモート・デバイスを決定します。

♦ サーバ起動同期を要求するように、統合データベース Dealer で変更を加えるには、次の手順に従います。

1. Interactive SQL で、次のように入力します。

```
UPDATE Dealer
   SET RATING = 'B' WHERE name = 'Geo';
COMMIT;
```

2. 同期が発生するまで数秒待ちます。

この場合、Notifier の begin_poll スクリプトは Dealer テーブルの変更を検出し、PushRequest テーブルを適切に移植します。この場合も、PushRequest テーブルが移植されると、Notifier の request_cursor スクリプトは、メッセージで送信される情報と、情報を受信するリモート・デバイスを決定します。

参照

- 「Push 要求の生成」 12 ページ
- 「INSERT 文」 『SQL Anywhere サーバ - SQL リファレンス』
- 「UPDATE 文」 『SQL Anywhere サーバ - SQL リファレンス』

チュートリアルの削除

チュートリアルをコンピュータから削除します。

♦ チュートリアルをコンピュータから削除するには、次の手順に従います。

1. Interactive SQL を閉じます。
2. SQL Anywhere、Mobile Link、同期クライアントの各ウィンドウを閉じます。
3. チュートリアルに関連するすべての DSN を削除します。
   a. ODBC アドミニストレータを起動します。
      コマンド・プロンプトで次のコマンドを入力します。

```
odbcad32
```

   b. **ml_sis** データ・ソースを削除します。
4. 統合データベースとリモート・データベースが保存されているディレクトリ *c:¥mlsis¥* に移動し、すべてのファイルを削除します。

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」 160 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」 165 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」 169 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)」 169 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」 177 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」 157 ページ
- 「jConnect」 157 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」 157 ページ
- 「iAnywhere JDBC ドライバ」 156 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」 157 ページ
- 「iAnywhere JDBC ドライバ」 156 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」 165 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」 160 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」 184 ページ
- 「同期」 184 ページ
- 「Ultra Light」 161 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」 165 ページ
- 「Listener」 157 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」 159 ページ
- 「サーバ起動同期」 165 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」 165 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」 155 ページ。

Replication Agent

参照：「LTM」 157 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」 157 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」 167 ページ
- 「SQL」 160 ページ
- 「データベース管理システム (DBMS)」 169 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」 161 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」 177 ページ
- 「パブリケーション」 172 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」 171 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」 164 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」 185 ページ
- 「コード・ページ」 165 ページ
- 「照合」 182 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」 161 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」 164 ページ
- 「カーソル位置」 163 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」 163 ページ
- 「カーソル結果セット」 164 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」 163 ページ
- 「カーソル位置」 163 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」 160 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」 170 ページ
- 「ローカル・テンポラリ・テーブル」 178 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」 165 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」 185 ページ
- 「エンコード」 163 ページ
- 「照合」 182 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」 172 ページ
- 「Mobile Link ユーザ」 158 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」 166 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」 166 ページ
- 「生成されたジョイン条件」 183 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」 162 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」 185 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」 184 ページ
- 「SQL ベースの同期」 161 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」 179 ページ
- 「プライマリ・キー」 174 ページ
- 「データベース管理システム (DBMS)」 169 ページ
- 「リレーショナル・データベース管理システム (RDBMS)」 177 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」 155 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)」 177 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)」 169 ページ
- 「SYS」 161 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」 169 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」 170 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」 165 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」 178 ページ
- 「グローバル・テンポラリ・テーブル」 164 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」 170 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」 171 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」 178 ページ
- 「文レベルのトリガ」 185 ページ
- 「整合性」 182 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」 172 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」 177 ページ
- 「アーティクル」 162 ページ
- 「パブリケーションの更新」 172 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」 177 ページ
- 「パブリケーション」 172 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」 177 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」 182 ページ
- 「ユーザ定義データ型」 176 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」 173 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」 179 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」 182 ページ
- 「検査制約」 181 ページ
- 「外部キー制約」 180 ページ
- 「一意性制約」 179 ページ
- 「整合性」 182 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」 161 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」 162 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」 175 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」 170 ページ
- 「ビュー」 173 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」 165 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」 175 ページ
- 「ビュー」 173 ページ

ミラー・ログ

参照：「トランザクション・ログ・ミラー」 171 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」 167 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」 177 ページ
- 「FILE」 156 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」 164 ページ
- 「サーバ・メッセージ・ストア」 165 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」 177 ページ
- 「統合データベース」 184 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」 170 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」 165 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」 184 ページ
- 「統合データベース」 184 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)」 169 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」 177 ページ
- 「パブリケーションの更新」 172 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」 177 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」 171 ページ
- 「文レベルのトリガ」 185 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」 170 ページ
- 「グローバル・テンポラリ・テーブル」 164 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」 171 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」 179 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」 171 ページ
- 「トランザクション・ログ・ミラー」 171 ページ
- 「フル・バックアップ」 174 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」 185 ページ
- 「同時性 (同時実行性)」 185 ページ
- 「整合性」 182 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」 180 ページ
- 「プライマリ・キー制約」 174 ページ
- 「制約」 182 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」 174 ページ
- 「外部テーブル」 180 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」 182 ページ
- 「検査制約」 181 ページ
- 「プライマリ・キー制約」 174 ページ
- 「一意性制約」 179 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」 166 ページ
- 「内部ジョイン」 185 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」 179 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」 182 ページ
- 「外部キー制約」 180 ページ
- 「プライマリ・キー制約」 174 ページ
- 「一意性制約」 179 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」 179 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」 174 ページ
- 「外部キー」 179 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」 174 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」 185 ページ
- 「コード・ページ」 165 ページ
- 「エンコード」 163 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」 173 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」 181 ページ
- 「外部キー制約」 180 ページ
- 「プライマリ・キー制約」 174 ページ
- 「一意性制約」 179 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」 181 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」 166 ページ
- 「ジョイン条件」 167 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」 165 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」 177 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」 184 ページ
- 「レプリケーション」 177 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」 157 ページ
- 「SQL Remote」 160 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」 171 ページ
- 「独立性レベル」 185 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」 167 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」 166 ページ
- 「外部ジョイン」 180 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」 171 ページ
- 「ロー・レベルのトリガ」 178 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」 165 ページ
- 「エンコード」 163 ページ
- 「照合」 182 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

I

J

O

P

Q

R

S

U

W

あ

い

う

え

お

か

き

し

す

せ

そ

た

ち

つ

て

と

れ

ろ

わ

