SYBASE iAnywhere

Mobile Link
サーバ起動同期

改訂 2007 年 3 月

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 のサーバによって開始される同期について説明します。サーバによって開始される同期とは、統合データベースから同期またはその他のリモート・アクションの開始を可能にする Mobile Link の機能です。

対象読者

このマニュアルは、この高度な機能を使用する 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『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

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

SQL Anywhere のマニュアル

SQL Anywhere の完全なマニュアルは、各マニュアルをまとめたオンライン形式とマニュアル別の PDF ファイルで提供されます。いずれの形式のマニュアルも、同じ情報が含まれ、次のマニュアルから構成されます。

- **『SQL Anywhere 10 - 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 10 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 10 - 変更点とアップグレード』** このマニュアルでは、SQL Anywhere 10 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 データベースの実行、管理、設定について説明します。管理ユーティリティとオプションのほか、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について説明します。
- **『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。
- **『SQL Anywhere サーバ - SQL リファレンス』** このマニュアルは、SQL Anywhere で使用する SQL 言語の完全なリファレンスです。また、SQL Anywhere のシステム・ビューとシステム・プロシージャについても説明しています。
- **『SQL Anywhere サーバ - プログラミング』** このマニュアルでは、C、C++、Java プログラミング言語、Visual Studio .NET を使用してデータベース・アプリケーションを構築、配備する方法について説明します。Visual Basic や PowerBuilder などのツールのユーザは、それらのツールのプログラミング・インタフェースを使用できます。
- **『SQL Anywhere 10 - 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、その診断情報とともに説明します。
- **『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 - 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 - 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。
- **『Mobile Link - サーバ起動同期』** このマニュアルでは、Mobile Link のサーバによって開始される同期について説明します。サーバによって開始される同期とは、統合データベースから同期またはその他のリモート・アクションの開始を可能にする Mobile Link の機能です。
- **『QAnywhere』** このマニュアルでは QAnywhere について説明します。QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアント用のメッセージング・プラットフォームであるほか、モバイル・クライアントや無線クライアント用のメッセージング・プラットフォームでもあります。
- **『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。
- **『SQL Anywhere 10 - コンテキスト別ヘルプ』** このマニュアルには、[接続] ダイアログ、クエリ・エディタ、Mobile Link モニタ、SQL Anywhere コンソール・ユーティリティ、インデックス・コンサルタント、Interactive SQL のコンテキスト別のヘルプが収録されています。
- **『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 - AppForge プログラミング』** このマニュアルでは、Ultra Light for AppForge について説明します。Ultra Light for AppForge を使用すると、Palm OS、Symbian OS、または Windows CE を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対してデータベース・アプリケーションを開発、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、モバイル、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 CE、または Windows XP を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対して Web ベースのデータベース・アプリケーションを開発、配備できます。
- **『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、モバイル、埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。

マニュアルの形式

SQL Anywhere のマニュアルは、次の形式で提供されています。

- **オンライン・マニュアル** オンライン・マニュアルには、SQL Anywhere の完全なマニュアルがあり、SQL Anywhere ツールに関する印刷マニュアルとコンテキスト別のヘルプの両方が含

まれています。オンライン・マニュアルは、製品のメンテナンス・リリースごとに更新されます。これは、最新の情報を含む最も完全なマニュアルです。

Windows オペレーティング・システムでオンライン・マニュアルにアクセスするには、[スタート] − [プログラム] − [SQL Anywhere 10] − [オンライン・マニュアル] を選択します。オンライン・マニュアルをナビゲートするには、左ウィンドウ枠で HTML ヘルプの目次、索引、検索機能を使用し、右ウィンドウ枠でリンク情報とメニューを使用します。

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

- **PDF ファイル** SQL Anywhere の完全なマニュアル・セットは、Adobe Reader で表示できる Adobe Portable Document Format (pdf) 形式のファイルとして提供されています。

  Windows では、PDF 形式のマニュアルはオンライン・マニュアルの各ページ上部にある PDF のリンクから、または Windows の [スタート] メニュー ([スタート] − [プログラム] − [SQL Anywhere 10] − [オンライン・マニュアル - PDF フォーマット]) からアクセスできます。

  UNIX では、PDF 形式のマニュアルはインストール CD にあります。

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書体およびグラフィック表現の規則について説明します。

SQL 構文の表記規則

SQL 構文の表記には、次の規則が適用されます。

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

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

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

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

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

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

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

- **オプション部分**　文のオプション部分は角カッコで囲みます。

  RELEASE SAVEPOINT [*savepoint-name*]

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

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

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

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

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

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

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

オペレーティング・システムの表記規則

- **Windows**　デスクトップおよびラップトップ・コンピュータ用の 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 ファミリには Windows Vista や Windows XP も含まれます。
- **Windows CE**　Microsoft Windows CE モジュラ・オペレーティング・システムに基づいて構築されたプラットフォームです。Windows Mobile や Windows Embedded CE などのプラットフォームが含まれます。

  Windows Mobile は Windows CE 上に構築されています。これにより、Windows のユーザ・インタフェースや、Word や Excel といったアプリケーションの小規模バージョンなどの追加機能が実現されています。Windows Mobile は、モバイル・デバイスで最も広く使用されています。

  SQL Anywhere の制限事項や相違点は、基盤となっているオペレーティング・システム (Windows CE) に由来しており、使用しているプラットフォーム (Windows Mobile など) に依存していることはほとんどありません。
- **UNIX**　特に記述がないかぎり、UNIX は Linux プラットフォームと UNIX プラットフォームの両方のことです。

ファイルの命名規則

マニュアルでは、パス名やファイル名などのオペレーティング・システムに依存するタスクと機能を表すときは、通常 Windows の表記規則が使用されます。ほとんどの場合、他のオペレーティング・システムで使用される構文に簡単に変換できます。

- **ディレクトリ名とパス名**　マニュアルでは、ドライブを示すコロンや、ディレクトリの区切り文字として使用する円記号など、Windows の表記規則を使用して、ディレクトリ・パスのリストを示します。次に例を示します。

  ```
  MobiLink¥redirector
  ```

  UNIX、Linux、Mac OS X では、代わりにスラッシュを使用してください。次に例を示します。

  ```
  MobiLink/redirector
  ```

  SQL Anywhere がマルチプラットフォーム環境で使用されている場合、プラットフォーム間でのパス名の違いに注意する必要があります。
- **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、拡張子 *.exe* が付きます。UNIX、Linux、Mac OS X では、実行ファイルの名前には拡張子は付きません。NetWare では、実行ファイルの名前には、拡張子 *.nlm* が付きます。

  たとえば、Windows では、ネットワーク・データベース・サーバは *dbsrv10.exe* です。UNIX、Linux、Mac OS X では、*dbsrv10* になります。NetWare では、*dbsrv10.nlm* になります。
- **install-dir**　インストール・プロセスでは、SQL Anywhere をインストールするロケーションを選択できます。マニュアルでは、このロケーションは *install-dir* という表記で示されます。

インストールが完了すると、環境変数 SQLANY10 によって SQL Anywhere コンポーネントがあるインストール・ディレクトリのロケーション (*install-dir*) が指定されます。SQLANYSH10 は、SQL Anywhere が他の Sybase アプリケーションと共有しているコンポーネントがあるディレクトリのロケーションを指定します。

オペレーティング・システム別の *install-dir* のデフォルト・ロケーションの詳細については、「SQLANY10 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・プロセスでは、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できます。マニュアルでは、このロケーションは *samples-dir* という表記で示されます。

  インストールが完了すると、環境変数 SQLANYSAMP10 によってサンプルがあるディレクトリのロケーション (*samples-dir*) が指定されます。Windows の [スタート] メニューから、[プログラム] – [SQL Anywhere 10] – [サンプル・アプリケーションおよびプロジェクト] を選択すると、このディレクトリで [Windows エクスプローラ] ウィンドウが表示されます。

  オペレーティング・システム別の *samples-dir* のデフォルト・ロケーションの詳細については、「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。

- **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 では、環境変数を参照するのに、構文 *%envvar%* が使用されます。UNIX、Linux、Mac OS X では、環境変数を参照するのに、構文 *$envvar* または *${envvar}* が使用されます。

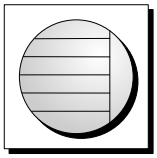
  UNIX、Linux、Mac OS X 環境変数は、*.cshrc* や *.tcshrc* などのシェルとログイン・スタートアップ・ファイルに格納されます。

グラフィック・アイコン

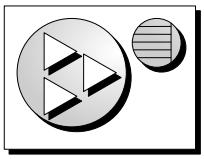
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

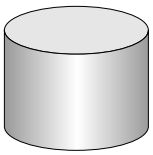
- SQL Anywhere などのデータベース・サーバ



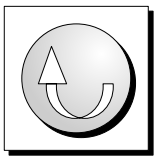
- Ultra Light アプリケーション



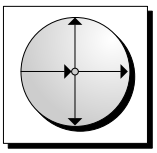
- データベース。高度な図では、データベースとデータベースを管理するデータ・サーバの両方をこのアイコンで表します。



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



- Sybase Replication Server



- プログラミング・インタフェース

詳細情報の検索／フィードバックの提供

詳細情報の検索

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

ご質問がある場合や支援が必要な場合は、次に示す 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。バージョン情報は、コマンド・プロンプトで **dbeng10 -v** と入力して確認できます。

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

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

ニュースグループに関するお断り
iAnywhere Solutions は、ニュースグループ上に解決策、情報、または意見を提供する義務を負うものではありません。また、システム・オペレータ以外のスタッフにこのサービスを監視させて、操作状況や可用性を保証する義務もありません。
iAnywhere のテクニカル・アドバイザとその他のスタッフは、時間のある場合にかぎりニュースグループでの支援を行います。こうした支援は基本的にボランティアで行われるため、解決策や情報を定期的に提供できるとはかぎりません。支援できるかどうかは、スタッフの仕事量に左右されます。

フィードバック

このマニュアルに関するご意見、ご提案、フィードバックをお寄せください。

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

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

第 1 章

サーバ起動同期の概要

目次

Mobile Link サーバ起動同期の概要

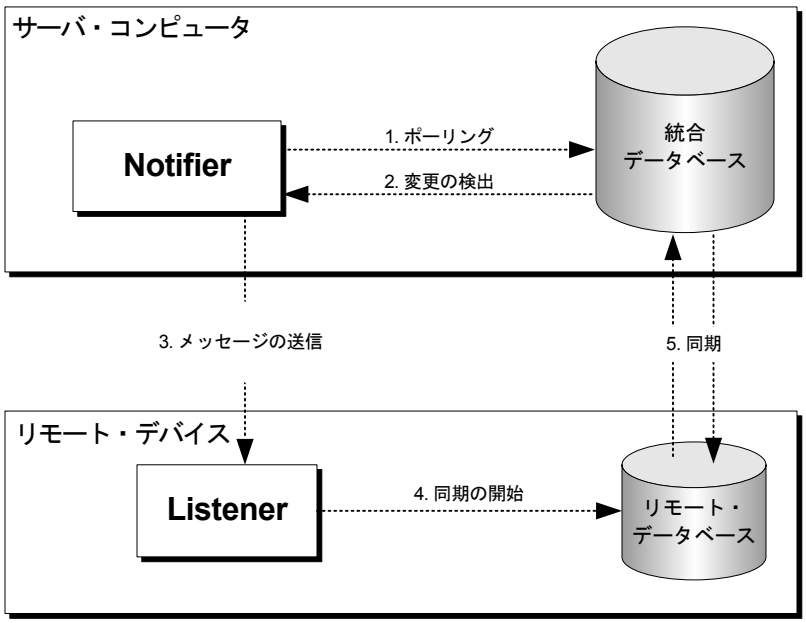
サーバ起動同期を使用すると、Mobile Link 同期を統合データベースから開始できます。これは、リモート・データベースが統合データベースにデータをアップロードできることに加え、データの更新をリモート・データベースにプッシュできることを意味しています。この Mobile Link コンポーネントには、どの変更内容が統合データベースで発生したら同期を開始するかを決定したり、プッシュするメッセージを受信するリモート・データベースを選択する方法やリモート・データベースの応答方法を決定したりするためのプログラム可能なオプションが用意されています。

例

たとえば、トラックの運転手全員がモバイル・データベースを使用して経路と配送先を調べるとします。ここで、1 人の運転手が道路が渋滞しているというレポートを同期します。Notifier というコンポーネントは統合データベースの変更を検出し、経路に影響するすべての運転手のリモート・デバイスにメッセージを自動的に送信します。この結果、運転手のリモート・データベースが同期され、運転手は別の経路を使用します。

通知プロセス

次の図では、Notifier は統合データベースをポーリングし、検索するよう設定された変更を検出します。この場合、Notifier は 1 つのリモート・デバイスにメッセージを送信します。この結果、リモート・データベースは同期によって更新されます。



この例で発生する手順を以下に示します。

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ポーリングし、リモート・データベースと同期する必要があるすべての変更を検出します。

2. 変更を検出すると、Notifier はリモート・デバイスにメッセージを送信する準備を行います。
3. Notifier はメッセージを送信します。デフォルトでは、同期と同じプロトコルを使用します。UDP または SMTP ゲートウェイを使用するように設定することもできます。
4. Listener は、メッセージの件名、内容、送信元をフィルタと照らし合わせてチェックします。
5. メッセージがフィルタと一致した場合、Listener はこのフィルタに関連付けられたプログラムを実行します。たとえば、dbmlsync を実行したり、Ultra Light アプリケーションを起動したりします。

接続起動同期

サーバから同期を起動するだけでなく、リモート・デバイス上の Listener によって生成された内部メッセージを使用して同期を起動することもできます。これらの内部メッセージは、接続状態が変化したことを示します。接続状態の変化とは、デバイスが Wi-Fi の受信可能範囲に入った、ユーザが RAS 接続を開始した、ユーザがデバイスをクレドールに置いた、といったことです。

「接続起動同期」 32 ページを参照してください。

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を使用します。

- **Push 要求** これにより、同期が発生します。Push 要求は、Mobile Link 統合データベース上のテーブルに挿入するデータのような形式を使用します。または、テンポラリ・テーブルに挿入されるデータや、SQL の結果セットのような形式の場合もあります。Push 要求は、テーブルにデータを挿入する任意の方法を使用して作成できます。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・トリガで Push 要求を作成することができます。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。

  「Push 要求」 10 ページを参照してください。

- **Notifier** Mobile Link サーバと同じコンピュータで実行されるプログラムです。これは、定期的に統合データベースをポーリングし、Push 要求を探します。Notifier がデータベースをポーリングする頻度は、制御することが可能です。通知されるリモート・デバイスを含め、Notifier が Push 要求の収集に使用するビジネス論理を指定することもできます。要求を検出すると、Notifier は、その要求に関連付けられたメッセージを 1 つまたは複数のリモート・デバイスの Listener に送信します。有効期限付きの繰り返し可能なメッセージを送信するオプションが用意されています。

  Notifier の詳細については、「Notifier」 17 ページ を参照してください。

- **Listener** 各リモート・デバイスにインストールされているプログラムです。Listener は、Notifier からのメッセージを受信して、アクションを開始します。このアクションは通常は同期ですが、その他の処理の場合もあります。Listener は、選択したソースからのメッセージである場合や、特定の内容のメッセージである場合にのみ動作するように設定できます。

  Windows または Windows CE では、Listener はコマンド・ライン・オプションで設定される実行プログラムです。メッセージを受信するには、リモート・デバイスを起動し、Listener を実行する必要があります。

  「Listener ユーティリティ」 37 ページを参照してください。

  Palm OS の場合、まず Windows デスクトップで Palm 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ます。次に、この設定ファイルを Palm デバイスにコピーして、Palm Listener を実行します。

  「Palm デバイス用 Listener」 49 ページを参照してください。

- **ゲートウェイ** Notifier から Listener にメッセージを送信するためのインタフェースです。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。SYNC ゲートウェイでは、Mobile Link 同期と同じプロトコルが使用されます。

  デバイス・トラッキング・ゲートウェイ リモート・デバイスを自動的にトラッキングする方法を提供します。デバイス・トラッキング機能を使用すると、リモート・デバイスのアドレスを知る必要がなくなります。デバイス・トラッカ・ゲートウェイのゲートウェイ名 (デフォ

ルトでは **Default-DeviceTracker**) と Mobile Link ユーザ名を指定すると、Mobile Link がメッセージを適切なゲートウェイを通じて適切なデバイスへルート指定します。

詳細については、「ゲートウェイと Carrier」 19 ページを参照してください。

サーバ起動同期がサポートされているプラットフォーム

次の項を参照してください。

- サポートされている Listener プラットフォーム
- Mobile Link 同期 PC プラットフォーム版
- Mobile Link 同期 UNIX プラットフォーム版

サーバ起動同期の配備

次に、サーバ起動同期を行うアプリケーションを配備する前に考慮すべきいくつかの事項について説明します。

UDP ゲートウェイを使用する場合の Listener の制限事項

- UDP ゲートウェイでは、Listener は受信用にソケットを開いたままにするため、Listener を IP ネットワークに接続して通知を受信できるようにする必要があります。
- リモート・デバイスの IP アドレスは、Mobile Link サーバからアクセスできるものでなければなりません。

Windows での Listener の制限事項 (Windows CE を含む)

- 現在サポートされている無線モデムでは、オペレーティング・システムが実行されている必要があります。この結果、バッテリーが消耗します。使用パターンに応じて、十分な電力を確保してください。

Palm Listener は自動的にデバイス・トラッキングを使用できない

- Palm では、デバイス・トラッキングが自動的に動作しません。ただし、これを可能にする方法があります。

  「デバイス・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」 23 ページを参照してください。

サーバ起動同期のクイック・スタート

次の手順では、Mobile Link 同期が設定済みであることを前提に、サーバ起動同期の設定に必要なタスクの概要について説明します。

この手順は、手動で実行します。Sybase Central でのサーバ起動同期の設定については、「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。

♦ サーバ起動同期の設定の概要

1. Push 要求を格納するためのテーブルを統合データベース上に作成します。

   「Push 要求」 10 ページを参照してください。

2. Push 要求を作成および管理する Notifier を設定します。

   「Notifier」 17 ページ を参照してください。

3. Notifier からの Push 要求をフィルタして処理する Listener を設定します。

   「Listener」 28 ページ を参照してください。

4. ゲートウェイ経由で転送します。

   - デフォルトの SYNC ゲートウェイを使用しており、そこで使用されているプロトコルが同期のプロトコルと同じ場合は、デフォルトのゲートウェイ設定に変更を加える必要があります。
   - UDP を使用する場合は、デフォルト設定を使用してメッセージを送信できます。
   - SMS 通知を送信する場合、ゲートウェイと Carrier を設定するとともに、SMS 受信ライブラリを指定する必要があります。

   「ゲートウェイと Carrier」 19 ページと「受信ライブラリ」 47 ページを参照してください。

クイック・スタートのためのその他の資料

- *samples-dir¥MobiLink¥SIS_** にサンプル・アプリケーションがあります。*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。

第 2 章

サーバ起動同期の設定

目次

Push 要求

Push 要求は、Mobile Link 統合データベース上のテーブルに挿入するデータと同じ形式を使用します。または、テンポラリ・テーブルに挿入されるデータや、SQL の結果セットと同じ形式を使用する場合もあります。Push 要求は、テーブルにデータを挿入するどの方法でも作成できます。

Notifier は、Push 要求を検出すると、リモート・データベースにメッセージを送信します。Push 要求は、メッセージの内容とともに、メッセージが送信された時期、方法、送信者を指定します。

Push 要求テーブルの作成

Push 要求は、統合データベースにある SQL 結果セット内のローです。このローには、次に示すカラムが以下の順序で格納されています。最初の 5 つのカラムは必須で、最後の 2 つのカラムはオプションです。Notifier は、request_cursor プロパティを使用して Push 要求をフェッチします。

通常の実装では、Push 要求テーブルを統合データベースに追加します。統合データベースで変更が検出された場合は、Push 要求テーブルを移植し、Notifier イベント request_cursor を使用して Push 要求をリモート Listener に送信します。Notifier イベント request_cursor は、次のカラムを次の順番で受信する必要があります。

カラム	説明
request id	INTEGER。Push 要求のユニークな ID です。
gateway	VARCHAR。メッセージを送信するゲートウェイです。これは、事前に定義されたゲートウェイまたはユーザ定義のゲートウェイです。事前に定義されたゲートウェイは **Default-DeviceTracker**、**Default-SMTP**、**Default-UDP** です。
subject	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。送信先アドレス。SYNC ゲートウェイまたはデバイス・トラッカ・ゲートウェイの場合は、Listener の Mobile Link ユーザ名、またはユーザが dblsn -t+ コマンドを使用して登録したその他の Mobile Link ユーザ名です。デバイス・トラッキングを使用しない SMTP ゲートウェイの場合は、電子メール・アドレスです。デバイス・トラッキングを使用しない UDP ゲートウェイの場合は、IP アドレスまたはホスト名です (ホスト名の後にコロンとポート番号が付くこともあります)。

カラム	説明
resend interval	VARCHAR。省略可。メッセージを再送する頻度です。デフォルトの単位は分です。秒、分、時間の単位には、**S**、**M**、**H** を使用します。また、1H 30M 10S のように単位を組み合わせることもできます。 再送間隔は、リモート・デバイスが、信頼性の低いネットワークを介して UDP メッセージの受信を待機する場合に特に便利です。Notifier は、再送可能な通知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再送可能な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。再送可能な通知を停止するには、request_cursor クエリを使用するか、要求テーブルから要求を削除します。デフォルトでは一度だけ送信され、再送は行われません。対象 Listener から受信確認が届くと、次の再送は停止されます。
time to live	VARCHAR。省略可。再送の有効期限が切れるまでの時間です。デフォルトの単位は分です。秒、分、時間の単位には、**S**、**M**、**H** を使用します。また、1H 30M 10S のように単位を組み合わせることもできます。 この値が 0 (ゼロ)、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。

注意 : Push 要求は、テンポラリ・テーブル内や、複数のテーブルにまたがって格納することもできます。

デバイス・トラッキングを使用している場合のアドレス指定通知の詳細については、「デバイス・トラッキング用の Listener オプション」 22 ページを参照してください。

例

次に示すのは、Push 要求テーブルを作成する SQL Anywhere の CREATE TABLE 文です。

```
create table PushRequest (
   req_id    integer default autoincrement primary key,
   gateway    varchar(128),
   subject    varchar(128),
   content    varchar(128),
   address    varchar(128),
   resend_minute varchar(30),
   minute_to_live varchar(30)
)
```

次のコードでは、ml_add_property ストアド・プロシージャを使用して、Push 要求を作成する request_cursor プロパティを作成しています。

```
call ml_add_property( 'SIS', 'Notifier(Simple)', 'request_cursor',
  'select req_id,
        gateway,
          subject,
          content,
          address,
          resend_minute,
        minute_to_live
   from PushRequest' );
```


Push 要求の作成

Push 要求は、テーブルにデータを挿入するどの方法でも作成できます。次に、Push 要求を作成する一般的な方法のリストを示します。

- Notifier プロパティの SQL 同期論理を指定する。Push 要求の作成で最も明瞭なプロパティは、begin_poll プロパティです。

  Notifier の内部で Push 要求を作成すると、Push 要求で 1 つのデータベース接続のみが使用されるので、競合を最小化できて便利です。

  「begin_poll プロパティ」 59 ページを参照してください。

- データベース・トリガを定義する。たとえば、価格の変更を検出し、Push 要求データを Push 要求テーブルに挿入するトリガを作成します。

  「トリガの概要」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

- Mobile Link 同期論理を使用して、他の Mobile Link ユーザに通知する Push 要求を作成する。たとえば、特定の変更がアップロードされたことを検出し、その後同じデータが必要な他のユーザを更新するように Push 要求を作成する end_upload スクリプトを作成します。

  「end_upload テーブル・イベント」 『Mobile Link - サーバ管理』を参照してください。

- データを Push 要求テーブルに直接挿入するデータベース・クライアント・アプリケーションを使用する。

- Interactive SQL ユーティリティを使用して Push 要求データを手動で挿入する。

Push 要求の送信

Notifier は、request_cursor プロパティで指定されている SQL クエリを実行して、複数の Push 要求を送信します。

統合データベースに対してクエリを発行する方法の詳細については、「request_cursor プロパティ」 62 ページを参照してください。

Push 要求の削除

Push 要求を削除すると、古いメッセージが再送されなくなります。適時に Push 要求を削除することで、送信されるメッセージ数が最小限に抑えられ、アプリケーションの効率が向上します。

Push 要求を削除する最も簡単な方法は、Notifier のプロパティの request_delete を使用することです。このプロパティは、パラメータとして要求 ID を持つ SQL 文です。この文を使用すると、Notifier は配信済みとして確認された要求や有効期限が切れた要求を削除します。

詳細については、「request_delete プロパティ」 62 ページを参照してください。

組み込み配信確認は、Palm デバイスでは使用できません。これは、すべてのデバイスで無効にできます。オプションで、独自の配信確認メカニズムを実装できます。たとえば、同期論理では特定の同期が発生した場合に、要求テーブルからの Push 要求を削除できます。

sa_send_udp による Listener への通知

SQL Anywhere データベースには、Listener に UDP 通知を送信するために使用するシステム・ストアド・プロシージャ sa_send_udp が格納されています。

Listener に通知する方法として sa_send_udp を使用する場合は、UDP パケットに 1 を追加してください。この数字は、サーバで開始された同期プロトコル番号です。Mobile Link の将来のバージョンでは、新しいプロトコル・バージョンによって Listener リスナの動作が変更される可能性があります。

「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。

例

あるデバイス上で、次のように Listener を起動します。*path* は、Internet Explorer が存在する場所です。

```
dblsn -v -l "message=TheMessage;action=start 'path¥iexplore.exe' http://www.sybase.com"
```

別のデバイス上で、SQL Anywhere データベースを起動します。InteractiveSQL を起動してデータベースに接続します。次の SQL を実行します(UDP パケットの末尾に 1 が追加されている点に注意してください)。

```
call SA_SEND_UDP('machine#1_ip_name',5001,'TheMessage1')
```

Internet Explorer が開き、Sybase のホームページが表示されます。

この例を 1 つのデバイス上で動作させるには、sa_send_udp の 1 番目のパラメータに sa_send_udp と指定します。

プロパティの設定

Notifier、ゲートウェイ、Carrier は、プロパティを介して設定します。これらのプロパティは、ml_property Mobile Link システム・テーブルまたは Notifier プロパティ・ファイルに格納できます。

データベースへのプロパティの格納

プロパティ設定を Mobile Link システム・テーブル ml_property に格納できます。このようにするには、次の 2 つの方法があります。

- Sybase Central 内の Mobile Link プラグインの [通知] フォルダを使用します。また、Sybase Central 内の [通知] フォルダを右クリックして Notifier プロパティ・ファイルに設定をエクスポートするか、Notifier プロパティ・ファイルから設定をインポートすることもできます。

  詳細については、Sybase Central [Notifier] ダイアログの [ヘルプ] をクリックしてください。

- Sybase Central の Mobile Link プラグインのモデル・モードでサーバ起動同期タブを使用します。

- ストアド・プロシージャ ml_add_property を使用します。

  「ml_add_property」『Mobile Link - サーバ管理』を参照してください。

プロパティ・ファイルへのプロパティの格納

別の方法として、オプションを Notifier のプロパティ・ファイルに格納できます。これはテキスト・ファイルで、テキスト・エディタで編集できます。プロパティ・ファイルを使用する場合は、デフォルト SYNC ゲートウェイは使用できません。次の項を参照してください。

- 「Notifier のプロパティ・ファイル」 15 ページ
- 「SYNC ゲートウェイ・プロパティ」 70 ページ

複数の場所でのプロパティの設定

ml_properties テーブルと Notifier プロパティ・ファイルの両方にプロパティを指定する場合、設定は次のように決定されます。

1. 統合データベースの ml_property テーブル内にあるサーバ起動同期プロパティがロードされます。
2. Notifier プロパティ・ファイルが -notifier オプションで指定されている場合、このファイルの設定がデータベースからの設定の先頭にロードされます。

   Notifier プロパティ・ファイルが設定されておらず、デフォルトの設定ファイル (*config.notifier*) が検索された場合、デフォルト・ファイルの設定がデータベースからの設定の先頭にロードされます。

プロパティの変更

プロパティは起動時に読み込まれます。プロパティを変更した場合、Mobile Link サーバを停止して再起動しないと、その変更が有効になりません。

プロパティ

設定できる Notification プロパティのリストの詳細については、次を参照してください。

- 「共通プロパティ」 54 ページ
- 「Notifier プロパティ」 55 ページ
- 「ゲートウェイ・プロパティ」 66 ページ
- 「Carrier プロパティ」 74 ページ

Notifier のプロパティ・ファイル

Notifier、ゲートウェイ、Carrier のプロパティは、ml_property Mobile Link システム・テーブルまたは Notifier のプロパティ・ファイルに格納できます。「プロパティの設定」 14 ページを参照してください。

Notifier のプロパティ・ファイルは、テキスト・ファイルです。これには、任意の名前を付けることができます。このファイルを作成する最も簡単な方法は、*samples-dir¥MobiLink¥* にあるテンプレート *template.notifier* を変更することです。*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。

ml_property テーブルから Notifier のプロパティ・ファイルへプロパティをエクスポートできます。エクスポートするには、Sybase Central 内の Mobile Link プラグインに接続し、[通知] フォルダを右クリックして、[設定のエクスポート] を選択します。エクスポートされたファイルは別の場所にコピーでき、そこで Notifier の設定を簡単に行えます。

Notifier のプロパティ・ファイルは、複数作成できます。使用するプロパティ・ファイルを特定するには、mlsrv10 を起動するときに、-notifier オプションを使用して名前とロケーションを指定します。次に、mlsrv10 コマンド・ラインの一部を示します。

```
mlsrv10 ... -notifier "c:¥CarDealer.notifier"
```

コマンド・ラインでプロパティ・ファイルを指定した場合に、どのようにプロパティが読み込まれるかの詳細については、「複数の場所でのプロパティの設定」 14 ページを参照してください。

Notifier のプロパティ・ファイルでは、複数の Notifier と複数のゲートウェイを設定および起動できます。この場合、定義する各 Notifier とゲートウェイの名前を指定します。

Notifier のプロパティは通常 1 行に入力しますが、行が次の行に続くことを表す文字として円記号 (¥) を使用できます。

円記号は、エスケープ文字でもあります。プロパティの設定では、次のエスケープ・シーケンスを使用できます。

エスケープ・シーケンス	説明
¥b	¥u0008: バックスペース (BS)
¥t	¥u0009: 水平タブ (HT)
¥n	¥u000a: ラインフィード (LF)
¥f	¥u000c: フォーム・フィード (FF)
¥r	¥u000d: キャリッジ・リターン (CR)
¥"	¥u0022: 二重引用符 (")
¥'	¥u0027: 一重引用符 (')
¥¥	¥u005c: 円記号 (¥)
¥uhhhh	ユニコード文字 (16 進数)
¥xhh	¥xhh: ASCII 文字 (16 進数)
¥e	¥u001b: エスケープ (ESC)

例

詳細なコメントが付いたサンプル Notifier プロパティ・ファイルは、*samples-dir¥MobiLink ¥template.notifier* 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。

注意
デフォルト SYNC ゲートウェイを使用する場合、Notifier 設定をこのプロパティ・ファイルには保存できません。データベースに格納する必要があります。「プロパティの設定」 14 ページを参照してください。

Notifier

Notifier は、Mobile Link サーバと同じコンピュータで実行されます。Notifier は、定期的に統合データベースをポーリングし、Push 要求を探します。Push 要求を検出すると、リモート・デバイスにメッセージを送信します。また、カスタム SQL スクリプトの実行、配信確認の処理、Push 要求の削除、データベース接続が失われた後の再接続などの機能も含まれています。カスタム SQL スクリプトを使用して、データのモニタと Push 要求の作成ができます。

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。各 Notifier は、常に 1 つのデータベース接続を開くようにしています。

複数の Notifier を使用した例については、*samples-dir¥MobiLink¥SIS_MultipleNotifier* にあるサンプル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。

Notifier の起動

Notifier は、mlsrv10 コマンド・ラインで起動します。Notifier を起動するには、mlsrv10 オプション **-notifier** を使用します。必要に応じて、Notifier のプロパティ・ファイルの名前があればそれを指定することも可能です。

次に、mlsrv10 コマンド・ラインの一部を示します。

```
mlsrv10 ... -notifier c:¥myfirst.notifier
```

コマンド・ラインでプロパティ・ファイルを指定した場合に、どのようにプロパティが読み込まれるかの詳細については、「複数の場所でのプロパティの設定」 14 ページを参照してください。

プロパティの適用方法については、「プロパティの設定」 14 ページを参照してください。

-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。

-notifier オプションを使用する場合、有効にしたすべての Notifier が起動されます。Notifier を有効にする方法の詳細については、「enable プロパティ」 63 ページを参照してください。

QAnywhere ユーザ向けの注意
mlsrv10 -m オプションを使用して、Mobile Link サーバを実行すると、Notifier が自動的に起動します。

通知を送信する別の方法

SQL Anywhere 統合データベースを使用している場合は、Notifier 以外の方法として、sa_send_udp ストアド・プロシージャを使用して簡単な通知を送信できます。

「sa_send_udp による Listener への通知」 13 ページを参照してください。

Notifier の設定

Notifier を使用することにより、カスタム SQL を作成して、サーバ起動同期処理をプログラムできます。これは、プロパティを設定することで行います。たとえば、次のようなタスクを実行するためにプロパティを設定します。

- poll_every プロパティを使用してポーリング間隔を設定する。
- 統合データベースの変化に応じて Push 要求を作成する。begin_poll プロパティは常にこのような方法で使用します。
- request_cursor プロパティを使用して、メッセージで送信する情報、送信先、送信場所、送信時間を決定します。Notifier は request_cursor から返された結果セットを使用して、Push 要求をリモート Listener に送信します。

  注意：request_cursor プロパティのみが必須プロパティです。「request_cursor プロパティ」 62 ページを参照してください。
- request_delete プロパティで Push 要求を削除する。

Notifier の完全なプロパティのリストについては、「Mobile Link 通知プロパティ」 53 ページを参照してください。

Notifier のプロパティの設定方法については、「プロパティの設定」 14 ページを参照してください。

Notifier プロパティの順序

次の擬似コードは、サーバ起動同期のプロパティが使用される順序を示しています。request_cursor を除いて、これらのプロパティはすべてオプションであることに注意してください。

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

ゲートウェイと Carrier

ゲートウェイは、メッセージを送信するメカニズムです。SYNC ゲートウェイ、UDP ゲートウェイ、SMTP ゲートウェイを定義できます。通常は、使用すべきゲートウェイの自動決定機能を備えたデバイス・トラッカ・ゲートウェイも使用できます。

- **デバイス・トラッカ・ゲートウェイ**　デバイス・トラッカ・ゲートウェイを使用する場合、Push 要求のアドレスは Listener の Mobile Link ユーザ名です。

  デバイス・トラッキングを使用すると、リモート・デバイスのアドレス変更を自動管理できます。デバイス・トラッキングを使用する場合、SYNC が最初に試行され、UDP ゲートウェイと SMTP ゲートウェイが有効の場合はこの順序でフォールバックされます。

  デバイス・トラッカ・ゲートウェイの使用をおすすめします。デバイス・トラッキングを使用しない場合、request_cursor には UDP または SMTP ゲートウェイの名前とアドレスを含める必要があります。Push 要求では、このゲートウェイだけが使用されます。

  「デバイス・トラッキング」 21 ページと「デバイス・トラッカ・ゲートウェイ・プロパティ」 66 ページを参照してください。

- **SYNC ゲートウェイ**　SYNC ゲートウェイは、同期と同じプロトコルを使用できます。接続は永続的で、すべての Listener 通信 (通知、確認、デバイス・トラッキング) に使用されます。SYNC を使用している場合、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」 70 ページを参照してください。

- **UDP ゲートウェイ**　このゲートウェイを使用すると、Push 要求をリモート・リスナに UDP を使用して送信できます。UDP を使用している場合、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。

  「UDP ゲートウェイ・プロパティ」 71 ページを参照してください。

- **SMTP ゲートウェイ**　SMTP ゲートウェイを使用すると、無線通信事業者が提供する電子メールから SMS メッセージへの変換サービスを通じて、SMS メッセージを SMS リスナに送信できます。SMTP の場合、SMTP ゲートウェイと Carrier を設定する必要があります。

  「SMTP ゲートウェイ・プロパティ」 67 ページを参照してください。

SMTP ゲートウェイを使用する場合は、**Carrier** を設定して、使用する公衆無線通信事業者に関する情報を格納します。Carrier 情報は、Listener から送信されるデバイス・トラック情報から SMS 電子メール・アドレスを作成するのに使用します。

ゲートウェイと Carrier の設定

ゲートウェイと Carrier の各プロパティの設定方法については、「プロパティの設定」 14 ページを参照してください。

ゲートウェイ・プロパティと Carrier プロパティのリストについては、次を参照してください。

- 「デバイス・トラッカ・ゲートウェイ・プロパティ」 66 ページ

- 「UDP ゲートウェイ・プロパティ」 71 ページ
- 「SMTP ゲートウェイ・プロパティ」 67 ページ
- 「Carrier プロパティ」 74 ページ

ゲートウェイ

デフォルトのゲートウェイには 3 種類あります。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

デバイス・トラッカ・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。デバイス・トラッカ・ゲートウェイは、Listener から送信されたデバイス・トラック情報に基づいて、いずれかの従属ゲートウェイへ自動的に各メッセージをルート指定します。「デバイス・トラッキング」 21 ページを参照してください。

Default-SYNC、Default-UDP、Default-SMTP は、特に SYNC と UDP について、問題なく動作するようにいくつかの設定を使用して事前に設定されています。ほとんどの場合、デフォルト・ゲートウェイを使用してください。必要に応じて設定をカスタマイズできます。

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。追加のゲートウェイを作成して、名前を割り当てることができます。

Carrier

SMTP ゲートウェイとともにデバイス・トラッキングを使用している場合、Carrier のみ設定する必要があります。Carrier の設定を行うことで、ネットワーク・プロバイダ、電子メールのプレフィクス、ネットワーク・プロバイダ ID などの情報を指定できます。この情報は、各無線通信事業者の電子メールから SMS への変換サービスで使用する電子メール・アドレスを作成するために、Notifier で必要です。

Carrier を設定するには、モデムがあり、サービス・プロバイダを設定しているデバイスで Listener を実行し、Listener コンソールまたはログを検査します。Lisner で -x オプションを使用して実行中の Mobile Link サーバに接続する場合、ml_device_address Mobile Link システム・テーブル内にある Carrier のデバイス・トラッキング情報も検索できます。

いったん Carrier を設定すれば、これ以上の設定は不要です。設定した Carrier を使用して、その公衆無線通信事業者を使用しているすべてのデバイスへ SMTP を介して SMS メッセージを送信できます。

「Carrier プロパティ」 74 ページを参照してください。

デバイス・トラッキング

デバイス・トラッキングを使用することにより、Push 要求内に Mobile Link ユーザ名を指定するだけで、リモート・データベースをアドレス指定できます。デバイス・トラッキングが有効の場合、Mobile Link はユーザのアクセス方法を追跡し続けます。たとえば、SMTP ゲートウェイの場合、デバイスの IP 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 Mobile Link システム・テーブル内のデバイス・トラッキング情報を更新します。デバイス・トラッカ・ゲートウェイは、まず SYNC ゲートウェイの使用を試み、配信に失敗した場合に、UDP ゲートウェイまたは SMTP ゲートウェイの使用を試みます。

ほとんどの場合、デバイス・トラッキングを使用してください。配備が容易に行えるようになるため、これを使用することをおすすめします。

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・トラッカ・ゲートウェイを使用することもできます。

「デバイス・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」 23 ページを参照してください。

デバイス・トラッキングを使用しない場合、request_cursor には特定の UDP または SMTP ゲートウェイの名前とアドレスを含める必要があります。各 Push 要求に対して、そのゲートウェイのみが使用され、他のゲートウェイは使用されません。

デバイス・トラッキングの設定

♦ **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、UDP ゲートウェイまたは SMTP ゲートウェイ (またはその両方) を設定します。*注意 :* 通常、UDP ゲートウェイは特別な設定なしで使用できるので、そのためにゲートウェイや Carrier を設定する必要はありません。ただし、電子メールから SMS への通知を使用する場合は、SMTP ゲートウェイのデフォルト設定を変更する必要があります。

   「ゲートウェイと Carrier の設定」 19 ページを参照してください。

2. request_cursor スクリプトを次のように設定します。

   - ゲートウェイ名は、デバイス・トラッカ・ゲートウェイの名前にしてください。デフォルトでは、Default-DeviceTracker です。
   - アドレスは、Mobile Link ユーザ名にしてください。デフォルトで、Listener ユーザ名を使用できます。または、dblsn の -t+ オプションを使用して、同期するリモート・データベースの Mobile Link ユーザ名を追加し、そのデータベースのアドレスを直接指定することもできます。

   「request_cursor プロパティ」 62 ページを参照してください。

3. Mobile Link の 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。デバイス名は、Listener コンソール内で検索できます。オプションで、dblsn -e オプションを使用してデバイス名を設定できます。別の Listener 名を指定するには、dblsn -u オプションを使用します。

デフォルト名を使用するかどうかにかかわらず、*Listener_name* を統合データベースにある ml_user Mobile Link システム・テーブルに追加する必要があります。これは、*Listener_name* が Mobile Link ユーザ名だからです。他の Mobile Link ユーザ名と同様に、Listener_name はユニークでなければならず、統合データベースの Mobile Link ml_user システム・テーブルに追加する必要があります。

「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。

「デバイス・トラッキング用の Listener オプション」 22 ページを参照してください。

デバイス・トラッキング用の Listener オプション

dblsn には、次のデバイス・トラッキング用オプションがあります。

Mobile Link サーバへの接続方法を指定するには、-x、-u、-w を使用します。これは、デバイス・トラッキングを使用している場合に必要です。これにより、アドレスが変更されたときにリモート・デバイスが統合データベースを更新できるようになります。また、配信確認を統合データベースに送信する場合にも必要です。

-t+ オプションは、指定することをおすすめします。このオプションを指定すると、リモート・データベースの Mobile Link ユーザ名を登録し、それを Listener データベースの Mobile Link ユーザ名の代わりに、通知アドレスとして使用できます。これは一度だけ操作すれば済みます。

- **-t+ *ml_user*** このオプションは、リモート・データベースに Mobile Link ユーザ名を登録するために使用します。これにより、登録したユーザ名を Push 要求のアドレスとして使用できます。

  -t+ を使用して、Mobile Link ユーザ名を複数登録できます。これは、複数のリモート・データベースにがある場合など、リモート・デバイス上の異なるアプリケーションに通知を送信する場合に便利です。

  このマッピングは、いったんトラッキング情報が正常にアップロードされるとサーバの ml_listening テーブルに保持されます。したがって、Mobile Link のユーザ名や場所を変更しないかぎり Mobile Link のユーザ名を 1 回だけ登録する必要があります。ただし、-t+ を複数回使用しても問題はありません。

- **-t- *ml_user_alias*** デバイス・トラッキング用に -t+ を使用して登録した Mobile Link ユーザ名を無効にするには、**-t-** オプションを使用します。

- **-u *ML_user_name*** -u を使用して、Listener 用の Mobile Link ユーザ名を作成します。デフォルトの *Listener_name* の *device_name*-**dblsn** があるので、-u オプションは任意です。ここで、

device_name はデバイス名です。デバイス名は、Listener コンソール内で検索できます。オプションで、-e オプションを使用してデバイス名を設定できます。

デフォルト名を使用するかどうかにかかわらず、*Listener_name* を統合データベースにある ml_user Mobile Link システム・テーブルに追加する必要があります。これは、*Listener_name* が Mobile Link ユーザ名だからです。他の Mobile Link ユーザ名と同様に、Listener_name はユニークでなければならず、統合データベースの Mobile Link ml_user システム・テーブルに追加する必要があります。

「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。

- **-w *password*** このオプションは、Listener 名用のパスワードを設定するものです。
- **-x *connection-parameters*** -x を使用して、Mobile Link サーバへの接続方法を指定します。これは、デバイス・トラッキングを使用している場合に必要です。これにより、アドレスが変更されたときにリモート・デバイスが統合データベースを更新できるようになります。またこのオプションは、配信確認を統合データベースに送信する場合にも必要です。
- **-y** このオプションは、Listener 名のパスワードを更新するものです。

Listener のオプションの詳細については、「Listener 構文」 38 ページを参照してください。

例

次の 1 行コマンドは、デバイス・トラッキングを使用して Listener を起動します。

```
dblsn -l "subject=sync;action='run dbmlsync.exe -c dsn=rem1'"
 -x tcpip(host=MLSERVER_MACHINE) -t+ user1 -u remoteuser1
```

デバイス・トラッキングの停止

次のような場合に、デバイス・トラッキングを停止する方が便利なことがあります。

- デバイスが静的 IP アドレスで UDP のみを受信する場合。
- デバイスが UDP のみを受信し、待ち時間が少ない DNS 更新の動的 IP を持つ場合。つまり、静的 IP 名を使用してデバイスに直接アドレス指定できます。

配信確認の使用を継続したままデバイス・トラッキングを停止するには、dblsn -g オプションを使用します。

dblsn オプションの詳細については、「Listener 構文」 38 ページを参照してください。

デバイス・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用

Listener に次のような特徴がある場合は、完全に自動化された形式のデバイス・トラッキングを使用できません。

- Adaptive Server Anywhere 9.0.1 以前の Listener または Palm Listener である

この場合にデバイス・トラッキングを設定する方法については、「手動でのデバイス・トラッキングの設定」 24 ページを参照してください。

- UDP で受信し、リモート IP アドレスが Mobile Link サーバ・マシンから到達不可能である

  この場合の対処方法については、「到達不可能アドレス」 26 ページを参照してください。

手動でのデバイス・トラッキングの設定

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場合に役立つストアド・プロシージャがいくつかあります。これらのストアド・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 を操作します。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せずに Mobile Link ユーザ名によって受信者をアドレス指定しますが、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難しくなります。IP アドレスではなくホスト名でアドレス指定することでこの問題を回避できますが、この場合、DNS サーバ・テーブルの更新が遅いとメッセージの誤配信が発生する可能性があります。また、Mobile Link システム・テーブルの更新をプログラミングすることで、次のストアド・プロシージャを設定して IP アドレスを変更できます。

♦ 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. 各リモート・デバイスに対して、ml_device Mobile Link 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、ここでは 1 に設定されていますが、SQL Anywhere 9.0.0 からの Listener 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 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、ここでは **y** に設定されていますが、デバイス・トラッキングを無視するよう指定します。これを **n** に設定すると、デバイス・トラッキングによってこのレコードが上書きされます。最後のパラメータには、このレコードのソースにあるオプションの注釈が含まれています。

   「ml_set_device」 82 ページを参照してください。

2. 追加した各デバイスに対して、ml_device Mobile Link 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',
   '3211234567',
   '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 の network_provider_id プロパティと一致している必要があります。詳細については、「network_provider_id プロパティ」 74 ページを参照してください。3 番目のパラメータは、UDP の IP アドレス、または SMS 対応デバイスの電話番号です。4 番目のパラメータは、**y** に設定されていて、通知を送信するためにこのレコードをアクティブにします。5 番目のパラメータは、ここでは **y** に設定されていますが、デバイス・トラッキングを無視するよう指定します。これを **n** に設定すると、デバイス・トラッキングによってこのレコードが上書きされます。最後のパラメータには、このレコードのソースにあるオプションの注釈が含まれています。

Carrier 情報の検索方法については、「デバイス・トラッキング」 21 ページを参照してください。

ml_set_device_address の使用については、「ml_set_device_address」 84 ページ を参照してください。

3. 各リモート・データベースに対して、追加したデバイスの ml_listening Mobile Link 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 ユーザ名にマップします。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、**y** に設定されていて、デバイス・トラッキングのアドレス指定用にこのレコードをアクティブにします。4 番目のパラメータは、ここでは **y** に設定されていますが、デバイス・トラッキングを無視するよう指定します。これを **n** に設定すると、デバイス・トラッキングによってこのレコードが上書きされます。最後のパラメータには、このレコードのソースにあるオプションの注釈が含まれています。

「ml_set_listening」 86 ページを参照してください。

ゲートウェイのトラブルシューティング

この項では、リモート・デバイスとサーバとの通信に関連する既知の問題とその解決法について説明します。

到達不可能アドレス

現象

Notifier が追跡 IP アドレスを持つデバイスに到達できません。

原因

いくつかまたはすべてのデバイスが、Mobile Link サーバに対してプライベートであるために、直接アドレス指定できません。たとえば、リモート・デバイスがプライベートなサブネットワーク上にあり、そのアドレスがネットワーク内部のものである場合です。

対応策

次のいずれかを試してみます。

- IP アドレスが公衆無線通信事業者または ISP によって割り当てられている場合、プライベート IP アドレスではなくパブリック IP アドレスを取得するために、Carrier プランをアップグレードしてください。
- Wi-Fi を使用している場合、組織の IP セキュリティ・ポリシーによってデバイスが到達できないように設定されている可能性があります。社内の IT 部門に相談してください。
- SMTP ゲートウェイを使用します。

  デバイスの IP アドレスが到達不可能の場合、Listener のデバイス・トラッキングを -g オプションで停止できます。-g オプションは、デバイス・トラッキングが不要で、配信確認が必要な場合に便利です。配信確認を使用している場合、まず UDP 経由で接続を行い、次の UDP 試行のときに確認ができるようになります。

  配信確認の詳細については、「confirmation_handler プロパティ」 56 ページを参照してください。

追跡アドレスが正しくない

現象

デバイス・トラッキングがデバイス用に最良の IP アドレスを選択しません。

原因

デバイスのルーティング・テーブルに問題がある可能性があります。

対応策

次のいずれかを試してみます。

- ルーティング・テーブルを修正します。
- ml_set_device_address ストアド・プロシージャを使用してデバイスのトラッキングを無視し、address パラメータを正しいアドレスに設定します。4 番目のパラメータが **y** に設定されていることを確認してください。さらに、問題のある Listener に -g を使用します。

「ml_set_device_address」 84 ページを参照してください。

Listener

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、作成したメッセージ・ハンドラに基づいてアクションを実行します。一般的なメッセージ・ハンドラには、フィルタ、アクション、オプションが含まれています。

たとえば、次の Listener コマンド・ラインでは、Listener が dbmlsync を起動するのは、件名が **FullSync** であるメッセージを受信した場合です。

```
dblsn -l "subject='FullSync';action='run dbmlsync.exe ...'"
```

呼び出せるアクションには、次のようなものがあります。通常、対象となるアクションは、dbmlsync または Ultra Light アプリケーションを介して起動される同期です。

- プロセスを開始する。
- 完了するまでプロセスを実行する。
- 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。
- オプションで確認が可能な、TCP/IP を介したローカルまたはリモート・アプリケーションとのテキスト・ベース通信を実行する。

アクションは、メッセージから得られる変数でパラメータ化できます。これにより、動的なオプションを実装する場合の柔軟性が大幅に増加します。

通常、1 つのデバイス上で起動する必要のある Listener は 1 つだけです。1 つの Listener で複数のチャネルを受信でき、同一デバイスで複数の Mobile Link ユーザに対応できます。実行中の Listener は、常に UDP で受信します (Palm Listener を除く)。

Listener は、デバイス・トラッキング情報をもとの統合データベースへ同期することもできます。詳細については、「デバイス・トラッキング」 21 ページを参照してください。

参照

- Listener の構文とオプションについては、「Listener 構文」 38 ページを参照してください。
- Palm デバイスの詳細については、「Palm デバイス用 Listener」 49 ページ を参照してください。
- dbmlsync のオプションについては、「Mobile Link SQL Anywhere クライアント・ユーティリティ [dbmlsync]」『Mobile Link - クライアント管理』を参照してください。
- メッセージ・ハンドラの詳細については、「メッセージ・ハンドラ」 29 ページを参照してください。
- dblsn オプションを毎回コマンド・プロンプトに入力する代わりに、テキスト・ファイルに格納しておくと便利です。詳細については、「Listener オプションの保存」 34 ページを参照してください。

例

次のコマンドは Listener ユーティリティを起動します。コマンドは、1 行に入力する必要があります。

```
dblsn -v2 -m -ot dblsn.log -x "host=localhost"
 -l "subject=sync;action='start dbmlsync.exe
    -c eng=rem1;uid=DBA;pwd=sql -ot dbmlsyncOut.txt -k';"
```

この例で使用しているオプションは次のとおりです。

オプション	説明
-v2	ログの冗長性をレベル 2 に設定する (Listener DLL メッセージとアクション・トレースを記録する)。
-m	通知メッセージを記録する。
-ot	ログ・ファイルをトランケートし、そのファイルに出力メッセージを送信する。この例では、出力ファイルは dblsn.log です。
-x	Mobile Link サーバへの接続方法を指定します。このオプションは、デバイス・トラッキングと配信確認に必要です。この例では、"host=localhost" というプロトコル・オプションだけを指定しています。プロトコル・オプションの完全なリストについては、「-x オプション」 『Mobile Link - クライアント管理』を参照してください。
-l	メッセージ・ハンドラを指定します。この例では、フィルタ条件はメッセージの件名に **sync** という文字列が含まれていること、アクションは dbmlsync を起動することです。dbmlsync の 3 つのコマンド・ライン・オプションも指定されています。-c には、同期を実行する Mobile Link サーバへの接続文字列を指定します。-ot には、出力ログ・ファイル名を指定します。-k を指定すると、同期完了時に dbmlsync が停止します。

メッセージ・ハンドラ

dblsn コマンド・ラインを使用して「**メッセージ・ハンドラ**」を作成すると、フィルタするメッセージの種類や、受け入れた各メッセージに対するアクションを Listener に通知できます。

「Listener ユーティリティ」 37 ページを参照してください。

メッセージの解釈

メッセージ (Push 要求) は、次のような構造を持つ単体のテキストで受信されます。

message control_information

control_information は内部使用のためのもので、メッセージ処理の前に削除されます。Listener は、出力できない文字をチルダに置き換え、次のような形式で *message* 部を解釈します。

message = sender subj-open subject subj-close content

subj-open = **(| [| { | < | ' | "**

subj-open 文字は、左から右にスキャンして最初に見つかった文字によって識別されます。*subj-open* の値は、*subj-close* の値を識別します。*subj-close* で考えられる値は、)、]、}、>、'、" です。

最初の *subj-close* 文字の位置が、*subject* の終わりと *content* の始まりを示します。

メッセージが *subj-open* で始まる場合、*sender* は空です。この場合、メッセージの *sender* は、配信パスに依存する方法で識別されます。たとえば、UDP ゲートウェイを通過したメッセージは [*subject*]*content* という形式で到着し、*sender* は IP アドレスです。SMTP ゲートウェイは、電子メールから SMS サービスへ変換された電子メール・メッセージを送信します。そのフォーマットは、公衆無線通信事業者によって異なります。

フィルタと action 変数

Listener は、受信メッセージ (Push 要求) の処理方法を決定するためのフィルタと action 変数を提供します。

- **フィルタ**　フィルタを使用すると、Listener で実行するアクションを、件名や内容など、メッセージ (Push 要求) の一部分の内容に応じて決定できます。フィルタの指定には dblsn -l オプションを使用します。

  「subject と content フィルタの使用」 30 ページと「フィルタ message、message_start、sender の使用」 31 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 (Push 要求) の一部を Listener アクションに含めることができます。

  「action 変数」 45 ページを参照してください。

subject と content フィルタの使用

フィルタ **subject** と **content** を使用すると、メッセージを Push 要求内で指定された件名と内容でフィルタできます。これらのフィルタを使用する場合、Listener は Carrier によって受信されるフォーマットと一致するよう自動的にフィルタを調整します。たとえば、Sync という件名と Orders という内容でメッセージをフィルタしたいとします。この場合、UDP では [Sync] Orders、電子メールから SMS への変換サービスでは Bob@mail.com[Sync]Orders となることをユーザが考慮する必要はありません。

ユーザは、この件名にそれを囲むための閉じカッコを含めることはできません。上記の例では、UDP によって件名 Sync が角カッコで囲まれます。つまり、UDP で受信される件名に閉じ角カッコを使用することはできません。SMTP メッセージの場合、件名を囲むのに使用される文字は Carrier 側が決定します。これは、)、]、}、>、'、" のいずれかになります。

> **注意：**
> Push 要求を作成するときに、件名には英数字のみを使用することが最良の方法です。

SMS メッセージの場合、Listener は、送信者名、件名、内容から前後のスペースと前後のチルダ (~) 文字を削除します。改行文字などの印字できない文字は、フィルタリングの前に Listener によって削除されます。

リモート ID によるフィルタリング

SQL Anywhere リモート・データベースの場合、初めて同期したときに、リモート・データベースで使用するリモート ID を含む「**リモート ID ファイル**」が作成されます。このファイルは、データベースと同じ名前で、拡張子は *.rid* です。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、リモート ID ファイルを指定するには、データベース名を使用します。

リモート ID をフィルタで使用するには、dblsn -r オプションを使用して、リモート ID ファイルの名前とロケーションを指定する必要があります。その上で、フィルタで $remote_id 変数を使用します。-r を複数使用した場合、$remote_id はその直前で使用されている -r オプションによって指定されたファイルを参照します。

リモート ID を直接フィルタで使用することもできます。ただし、リモート ID はデフォルトで GUID なので、わかりやすいリモート ID 名を付けないかぎり、直接参照するのは簡単ではありません。

例

たとえば、次のコードは dblsn コマンド・ファイルの一部です。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 データベースの 2 つ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。

```
-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

- 「Listener 構文」 38 ページの -r オプション
- 「action 変数」 45 ページの $remote_id 変数
- 「リモート ID」 『Mobile Link - クライアント管理』

フィルタ message、message_start、sender の使用

おすすめするフィルタは、**subject** と **content** です。ただし、使用できるフィルタはこのほかに 3 種類あります。

Listener は、印字できない文字をチルダ (~) に変換するため、印字できない文字がある場合は、フィルタもチルダを使用する必要があります。

- **message**
  指定したテキストとメッセージ全体とを比較します。一致させるには、このフィルタはメッセージと完全に同じ長さでなければなりません。メッセージ・ハンドラごとに指定できる message は 1 つだけです。

メッセージのフォーマットは Carrier に依存していて、**message**、**message_start**、または **sender** フィルタを使用する場合はそれを把握しておく必要があります。たとえば、送信者が Bob@mail.com、件名が Help、メッセージが Me というメッセージに一致させる必要があるとします。UDP では、これは [Help]Me と表示されます。Bell Mobility が使用する電子メールから SMS への変換サービスでは、Bob@mail.com[Help]Me となります。Fido が使用する電子メールから SMS への変換サービスでは、Bob@mail.com¥n(Help)¥nMe と送信されますが、Listener によって Bob@mail.com(Help)Me に変換されます。-v と -m オプションを使用して Carrier とともにテストを実行し、適切なフォーマットを調べてください。

- **message_start**
  指定したテキストとメッセージの一部とを (先頭から) 比較します。message_start を指定すると、Listener は action 変数 $message_start と $message_end を作成します。「action 変数」 45 ページを参照してください。メッセージ・ハンドラごとに使用できる message_start は 1 つだけです。

- **sender**
  メッセージの送信元です。メッセージ・ハンドラごとに指定できる sender は 1 つだけです。UDP ゲートウェイの場合、送信者はゲートウェイのホストの IP アドレスです。SMS 電子メールでは、SMS フォーマットがサーバ起動同期と互換性がある場合、送信者はメッセージの最初に埋め込まれた電子メール・アドレスです。それ以外の場合、送信者情報は使用できません。

複数のメッセージ・ハンドラが必要な場合

メッセージが互換性のあるフォーマットで受信される場合、subject と content がおすすめするフィルタです。ただし、メッセージのフォーマットに互換性がない場合、message、message_start、sender フィルタを使用できます。この場合、配信パスが (UDP と SMTP で) 変化する場合、複数のハンドラで異なるフィルタを使用する必要があります。

接続起動同期

Windows デバイス上では、サーバからの同期起動に加えて、接続が変更された場合にも同期を起動できます。これが可能なのは、Windows Listener が、接続が変更された場合には _IP_CHANGED_ という内容の内部メッセージを、新しい最善の IP 接続が確立された場合には _BEST_IP_CHANGED_ という内容の内部メッセージを、それぞれ作成するからです。

内部メッセージ _IP_CHANGED_ と _BEST_IP_CHANGED_ は、Windows デバイス (Windows CE を含む) 上だけで生成されます。

Mobile Link サーバへの最適パスの変更の識別

IP 接続は、dblsn -x オプションで指定された Mobile Link サーバに接続するときに使用することが最適の接続である場合に、「最善」とみなされます。「最善」の指定は Mobile Link サーバへのパスによって定義されますが、実際には、一般に使用される最善の IP 接続を示す傾向があります。

最善の IP 接続が変更されたことを利用するには、メッセージ・フィルタでキーワード **_BEST_IP_CHANGED_** を使用します。Mobile Link サーバは、ネットワークが最適なルートを決

定するために宛先として必要です。したがって、-x オプションを使用して、Mobile Link サーバの接続パラメータも指定する必要があります。メッセージ・フィルタの形式は次のとおりです。

```
-l "message='_BEST_IP_CHANGED_';action=..."
```

$best_ip アクション変数は、_BEST_IP_CHANGED_ フィルタで使用すると便利です。$best_ip 変数の値は、最善の IP 接続を表すローカルの IP アドレスです。IP 接続が存在しない場合、$best_ip の値は 0.0.0.0 になります。

_BEST_IP_CHANGED_ が使用できるのは、Listener が Mobile Link サーバと異なるマシン上で実行されている場合だけです。

次の例では、_BEST_IP_CHANGED_ フィルタを使用して、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'"
```

接続におけるすべての変更の識別

リモート・デバイス上の IP 接続が変更されたことを利用するには、メッセージ・フィルタでキーワード **_IP_CHANGED_** を使用します。_IP_CHANGED_ は、IP 接続が変更されたことを示すだけです。メッセージ・フィルタの形式は次のとおりです。

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

次の例は、dblsn コマンド・ラインで使用できるメッセージ・ハンドラを示しています。このフィルタは、_IP_CHANGED_ という内容を含むメッセージを取得します。アクションでは、アクション変数 $adapters と $network_names を使用しています。接続が失われると、エラーが生成されます。

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

参照

- 「Listener ユーティリティ」 37 ページ
- 「action 変数」 45 ページ

マルチ・チャネル受信

複数の媒体で受信するには、Listener を -d オプションで起動します。UDP 受信用のライブラリはデフォルトで常にロードされますが、ロードする方法にはほかにいくつかあります。「Listener 構文」 38 ページと「受信ライブラリ」 47 ページを参照してください。

Listener の詳細については、「Listener ユーティリティ」 37 ページを参照してください。

Listener オプションの保存

Listener を設定するには、コマンド・ライン・オプションを設定ファイルに格納し、@ 記号を使用してアクセスすると便利です。設定ファイルはテキスト・ファイルです。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
```

設定ファイルへのパスは、完全に修飾されたパスでなければなりません。

「設定ファイルの使用」 『SQL Anywhere サーバ - データベース管理』を参照してください。

設定ファイル内のパスワードなどの情報を保護する場合は、ファイル非表示ユーティリティを使用して、設定ファイルの内容を難読化できます。

「ファイル非表示ユーティリティ (dbfhide)」 『SQL Anywhere サーバ - データベース管理』を参照してください。

コマンド・ライン・オプションを環境変数に格納して、dblsn コマンド・ラインで呼び出すこともできます。それには、**dblsn @dblsnoptions** のように、@ の後に環境変数名を入力します。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。

デフォルト・パラメータ・ファイル dblsn.txt

引数を付けずに dblsn と入力すると、dblsn はデフォルトの引数ファイルとして *dblsn.txt* を使用します。この機能は、対象が CE デバイスの場合に特に便利です。

次に、パラメータ・ファイルの例を示します。

```
#---- SIS_SimpleListener¥dblsn.txt ----------------------------------------
#
# This is the default argument file for dblsn.exe
#

#-------------------------------------------------------------------------
# Device name
#
-e device1

#-------------------------------------------------------------------------
# MobiLink connection parameters
#
-x host=localhost


#-------------------------------------------------------------------------
# Verbosity level 2
#
-v2

#-------------------------------------------------------------------------
# Show notification messages in console and log
#
-m

#-------------------------------------------------------------------------
# Polling interval of 1 seconds
#
-i 1
```

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

#-------------------------------------------------------------------------------
# First message handler
#    - No filter, so it applies to all messages
#    - Try to send the message to the beeper utility
#    - If that fails, start the beeper utility with the message
#    - Message handling continues with the next handler
#
-l "action='socket port=12345;
            sendText=$sender:$message;
            recvText=beeperAck;
            timeout=5';
    altaction='start java.exe Beeper 12345 $sender:$message';
    continue=yes;"


#-------------------------------------------------------------------------------
# Second message handler
#    - Only applies to messages with subject equals 'shutdown'
#    - The action is to send "shutdown" to the beeper utility
#    - Message handling continues with the next handler
#
-l "subject='shutdown';
    action='socket port=12345;
            sendText=shutdown;
            recvText=beeperAck;
            timeout=5';
    continue=yes;"

#-------------------------------------------------------------------------------
# Third handler
#    - Only applies to messages with subject equals 'shutdown'
#    - The action is to shut down the MobiLink Listener
#
-l "subject='shutdown';
    action='DBLSN FULL SHUTDOWN';"
```

第 3 章

Listener ユーティリティ

目次

Listener 構文

Listener ユーティリティの dblsn は、Windows CE などの Windows デバイスで Listener の設定と起動を行います。

この項は、Listener ユーティリティの詳細なリファレンスです。Listener ユーティリティの使用法については、「Listener」 28 ページ を参照してください。

Palm デバイスの詳細については、「Palm デバイス用 Listener」 49 ページ を参照してください。

構文

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

message-handler :
[*filter*;...]*action*
[**;continue = yes**]
[**;maydial = no**]
[**;confirm_delivery = 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*

パラメータ

オプション 次のオプションは、Listener の設定に使用できます。これらはすべてオプションです。

dblsn オプション	説明
@*data*	指定された環境変数または設定ファイルからオプションを読み込みます。環境変数と設定ファイルが両方とも存在する場合は、環境変数が使用されます。「Listener オプションの保存」 34 ページを参照してください。
-a *option*	Listener の DLL オプションを指定します。複数の -d オプションを指定する場合、後続の各 -a が -d オプション用になります。 複数のオプションを指定するには、たとえば -a port=2439 -a ShowSenderPort のように、-a を繰り返します。 DLL 用のオプションを確認するには、dblsn -d *filename.dll* -a ? と入力するか、「受信ライブラリ」 47 ページを参照してください。
-d *filename*	使用する Listener の DLL を指定します。デフォルトの DLL は *lsn_udp.dll* です。 SMTP ゲートウェイには、指定可能な DLL がいくつかあります。DLL のリストについては、「受信ライブラリ」 47 ページを参照してください。 マルチ・チャネル受信を可能にするには、-d を繰り返して複数の DLL を指定します。-d オプションの後に、DLL に関連する -a オプションと -i オプションを指定します。次に例を示します。 dblsn.exe -d lsn_udp.dll -i 10 -d maac750.dll -i 60
-e *device-name*	デバイス名を指定します。デフォルトでは、デバイス名は自動的にシステムから抽出されます。-e を使用しない場合、すべてのデバイスがユニークな名前であることを確認する必要があります。
-f *string*	デバイスに関する追加の情報を指定します。デフォルトでは、この情報はオペレーティング・システムのバージョンです。このオプションを使用すると、デフォルト値が上書きされます。
-i *seconds*	SMTP 接続のポーリング間隔を秒単位で設定します。これは、Listener がメッセージをチェックする頻度です。複数の -d オプションを指定する場合、後続の各 -i 設定が -d オプション用になります。SMTP 接続の場合、デフォルト値は 30 秒です。UDP 接続の場合、Listener はすぐに接続を試みます。
-m	メッセージのロギングを有効にします。デフォルトはオフです。
-ni	-x を使用する場合に UDP アドレスのトラッキングを停止します。これは、デバイス・トラッキングが不要で、配信確認が必要な場合に便利です。「confirmation_handler プロパティ」 56 ページを参照してください。

dblsn オプション	説明
-ns	Windows Mobile 2003 とその Phone Edition では、Listener は UDP とともに SMS を受信します。受信にはシステム・プロセスとして実行されるフィルタリング・メカニズムが使用されるので、Listener が実行されていないときでもフィルタリングは続行されます。-ns オプションは、この動作を無効にします。-ns が使用されると、Listener はデフォルトで UDP だけを受信します。SMS 受信を使用する場合は、受信ライブラリに -d オプションを指定できます。
-nu	デフォルトの UDP 受信を無効にします。
-o *filename*	ファイルに出力のログを取ります。-o が使用されない場合、出力はコンソール・ウィンドウに記録されます。
-os *bytes*	ログ・ファイルの最大サイズをバイト単位で指定します。最小サイズは 10000 で、デフォルトは無制限です。
-ot *filename*	ファイルに出力のログを取りますが、先にファイルの内容を削除します。
-p	アイドル状態になると自動的に電源を切ります。このオプションは、CE デバイスでのみ有効です。このオプションを使用すると、アイドル状態になったときにデバイスを停止できます。デフォルトでは、Listener がデバイス自体による停止を防止するので、受信が継続できます。
-pc {**+**|**-**}	-pc- を使用すると、受信通知用の永続的接続が無効になりますが、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引き続き使用されます。デフォルトでは、永続的接続は有効 (-pc+) です。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。
-q	最小化ウィンドウで実行します。
-r *remote-id-file*	メッセージ・ハンドラの応答アクションに関わる Mobile Link リモート・データベースを指定します。-r を使用すると、$remote_id 変数をメッセージ・ハンドラで使用して、*remote-id-file* に含まれているリモート ID を参照できます。このオプションは、リモート ID (デフォルトは GUID) の参照を簡素化します。 デバイスに複数のデータベースがある場合は、このオプションを複数回使用できます。 *remote-id-file* はリモート ID を含むファイルのフル・パス/名前です。このファイルは、最初の同期後に、dbmlsync によって自動的に作成されます。ファイルのロケーションと名前はデータベース・ファイルと同じで、拡張子は .rid です。Ultra Light データベースの場合は、Ultra Light データベース名をリモート ID ファイルと同じにしてください。 「リモート ID によるフィルタリング」 31 ページを参照してください。

dblsn オプション	説明
-t {**+**|**-**} *ml-user-alias*	通知用のリモート・データベースを登録して、デバイス・トラッキングを使用するとき、リモート・データベースを名前でアドレス指定できるようにします。$remote_id 変数を使用して、データベースを指定することもできます。 「デバイス・トラッキング用の Listener オプション」 22 ページと「action 変数」 45 ページを参照してください。
-u *Listener-name*	Mobile Link ユーザ名を指定します。この名前は、Listener が Mobile Link サーバに接続する必要がある場合、たとえばデバイス・トラッキング、配信確認、永続的接続に使用されます。 デフォルトの Mobile Link 名は、*device-name*-**dblsn** です。 Mobile Link ユーザ名は、Mobile Link サーバに登録する必要があります。「統合データベースへの Mobile Link ユーザ名の追加」『Mobile Link - クライアント管理』を参照してください。
-v [*level*]	dblsn のログとコンソールの冗長レベルを設定します。*level* は、**0**、**1**、**2**、**3** に設定できます。 ◆ **0** - 情報メッセージを表示しない (デフォルト)。 ◆ **1** - Listener の dll のメッセージ、基本的なアクション・トレース段階、コマンド・ライン・オプションを表示する。 ◆ **2** - レベル 1 に加え、詳細なアクションのトレース段階を表示する。 ◆ **3** - レベル 2 に加え、ポーリングと受信のステータスを表示する。 通知メッセージを出力するには、-m (上記参照) も指定する必要があります。
-w *password*	*Listener-name* のパスワードを指定します。 「デバイス・トラッキング用の Listener オプション」 22 ページを参照してください。
-x {**http**|**https**|**tcpip**} [**(***keyword***=***value*;...**)**]	Mobile Link サーバとの通信に使用するネットワーク・プロトコルとプロトコル・オプションを指定します。プロトコル・オプションのリストについては、「Mobile Link クライアントのネットワーク・プロトコル・オプション」『Mobile Link - クライアント管理』を参照してください。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したり、SYNC ゲートウェイを使用したりするのに必要です。 「デバイス・トラッキング用の Listener オプション」 22 ページを参照してください。

dblsn オプション	説明
-y *new-password*	Listener 名用の新規 Mobile Link パスワードを指定します。使用している認証システムでリモート・デバイスのパスワードを変更できる場合、このオプションによって新規パスワードを送信できます。 「デバイス・トラッキング用の Listener オプション」 22 ページを参照してください。

メッセージ・ハンドラ
-l オプションを使用すると、フィルタとアクションのペアであるメッセージ・ハンドラを指定できます。このフィルタは、処理するメッセージを判断します。アクションは、フィルタがメッセージと一致したときに呼び出されます。

-l のインスタンスは複数指定できます。-l の各インスタンスは、入力メッセージごとの異なるメッセージ・ハンドラを指定します。メッセージ・ハンドラは、指定した順序で処理されます。

また、メッセージ・ハンドラ用に次のオプションも指定できます。

- **continue=yes**　最初の一致を検出した後に Listener が処理を継続するかどうかを指定します。これは、複数の -l 句を指定して、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ようにするときに便利です。デフォルトは no です。
- **maydial=no**　アクションがモデムにダイヤル接続できないように指定します。このオプションは、アクションの前にモデムを解放するかどうかを決定するための情報を Listener に提供します。このオプションが役立つのは、action または altaction が、Listener によって使用されるモデムに排他的にアクセスする必要がある場合です。デフォルトは yes です。
- **confirm_delivery=no**　ハンドラが配信を確認しないように指定します。メッセージを送信するゲートウェイで confirm_delivery プロパティが [yes] に設定されている場合、メッセージの確認が必要になります。メッセージの確認が必要で、ハンドラがメッセージを受け入れた場合のみ、配信が確認可能です。デフォルトは yes です。

  通常、このオプションを指定する必要はありません。デフォルトでは、メッセージを最初に受け入れたハンドラが、必要に応じて配信確認を送信します。このオプションは、複数のハンドラが同一メッセージを受け入れ可能な場合に、どのハンドラが配信確認をするかを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 プロパティ」 56 ページを参照してください。

filter　受信メッセージとの比較を行うフィルタを指定します。フィルタが一致すると、指定したアクションが呼び出されます。

このフィルタはオプションです。フィルタを指定しない場合は、メッセージの受信時にアクションが行われます。これは、デバッグ時またはキャッチ・オール・メッセージ・ハンドラを最終メッセージ・ハンドラとする場合に便利です。

「subject と content フィルタの使用」 30 ページと「フィルタ message、message_start、sender の使用」 31 ページを参照してください。

action と altaction
各フィルタは、アクションに関連付けられています。また、必要に応じて altaction という代替アクションに関連付けられています。メッセージがフィルタの条件を満たしている場合に、アクションが呼び出されます。アクションの指定は必須です。altaction を指定すると、アクションが失敗した場合にのみ altaction が呼び出されます。

各 action と altaction には、1 つのコマンドを指定できます。指定可能なコマンドは、**start**、**run**、**post**、**socket**、または **DBLSN FULL SHUTDOWN** のいずれかです。

- **start**
  プロセスを生成します。プログラムを起動すると、Listener はメッセージの受信を再開します。

  start でプログラムを起動すると、Listener はリターン・コードを待ちません。このため、プログラムを検索または起動できない場合に、action が失敗したことだけを判定できます。

  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、action 変数 $content を使用してメッセージから取得されます。

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

- **run**
  プログラムを実行し、完了まで待機します。Listener は、処理が完了してから受信を再開します。

  run でプログラムを実行すると、プログラムの検索や起動ができない場合またはゼロ以外のリターン・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と判定します。

  次の例では、一部をメッセージから取得したコマンド・ライン・オプションをいくつか使用して dbmlsync を実行します。

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

- **post**
  ウィンドウ・クラスに Windows メッセージを送信します。スケジュールがオンになっているときは、dbmlsync に post が必要です。また、post は Windows のメッセージを使用するアプリケーションに通知するときにも使用されます。

  Windows メッセージは、メッセージの内容または Windows メッセージの ID 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のタイトルによって識別できます。ウィンドウ・クラスを名前で識別する場合は、dbmlsync -wc オプションを使用してウィンドウ・クラス名を指定します。ウィンドウ・クラスをタイトルで識別する場合は、トップ・レベル・ウィンドウのタイトルだけを使用してウィンドウ・クラスを識別します。

  Windows 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、そのメッセージまたは名前を一重引用符で囲みます。一重引用符自体を文字列に含めるには、一重引用符を 2 つ続けて書きます。たとえば、post my'message to my'class というメッセージの場合は、次の構文を使用します。


```
... -l "action='post my"message to my"class';"
```

または

```
... -l "action='post ""my""message" to ""my""class"";"
```

次の例では、dbas_synchronize と登録された Windows メッセージを、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。

```
"action='post dbas_synchronize to dmblsync_FullSync';"
```

「-wc オプション」 『Mobile Link - クライアント管理』を参照してください。

- **socket**
  TCP/IP 接続を確立して、アプリケーションに通知します。これは、特に動的情報を実行中のアプリケーションに渡す場合に便利です。また、Java と Visual Basic はカスタム・ウィンドウ・メッセージ機能をサポートしておらず、eVB はコマンド・ライン・パラメータをサポートしていないため、Java と Visual Basic アプリケーションを統合する場合に便利です。ローカルのソケットに接続するには、ポートのみを指定します。また、リモートのソケットに接続するには、ポートとともにホストを指定します。sendText を使用すると、文字列を送信できます。必要に応じて、recvText を使用し、予期された応答であることを確認することが可能です。recvText を使用するときは、タイムアウトを指定できます。こうすると、アプリケーションまたはネットワークの問題が発生している場合でもハングすることがなくなります。

  action に **socket** を実行すると、タイムアウトになる前に接続、送信、または予期される確認の受信に失敗した場合に、Listener は action が失敗したと判定します。

  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。この場合、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。

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

- **DBLSN FULL SHUTDOWN**
  Listener ユーティリティを停止します。停止すると、Listener は受信メッセージの処理を停止し、デバイス・トラッキング情報の同期を停止します。サーバからの同期を続行するには、リモート・ユーザは Listener を再度起動する必要があります。この機能は、テスト中に特に便利です。

  たとえば、action='DBLSN FULL SHUTDOWN' と指定します。

-l の各インスタンスでは、action と altaction はそれぞれ 1 つしか指定できません。1 つの action で複数のタスクを実行する場合、複数の action を含むカバー・プログラムまたはバッチ・ファイルを作成し、それを単一の action として実行できます。

次に、altaction の例を示します。この例では、$content が Mobile Link への接続用プロトコル・オプションです。プライマリの action は、Windows メッセージ dbas_synchronize を dbmlsync_FullSync ウィンドウに送信することです。この例では、プライマリの action が失敗した場合に、altaction を使用してウィンドウ・クラス名 dbmlsync_FullSync で dbmlsync を起動します (実行するのでは

ありません)。これは、dbmlsync のスケジュールと Listener を連動させるための標準的な方法です。

```
-l "subject=sync;
   action='post dbas_synchronize to dbmlsync_FullSync';
   altaction='start dbmlsync.exe
                  @dbmlsync.txt
              -wc dbmlsync_FullSync
              -e adr=$content;sch=INFINITE'"
```

参照

- 「Listener」 28 ページ

action 変数

次に示す Windows Listener の action 変数は、action または altaction 内の任意の位置で使用できます。

action 変数は、action または altaction が実行される直前に置き換えられます。

Listener の 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、"$$" と入力します。たとえば、$message_start が置き換えられないようにするときは、$$message_start と入力します。

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、配信パス固有のフォーマットを含むメッセージ全体。
$message_start	-l message_start で指定された、メッセージ・テキストの冒頭の一部。この変数を使用できるのは、-l message_start を指定した場合だけです。
$message_end	-l message_start で指定された部分が削除された後に残ったメッセージ部分。この変数を使用できるのは、-l message_start を指定した場合だけです。
$ml_connect	mlsrv10 -x オプションによって指定される Mobile Link 接続パラメータ。デフォルトは、空の文字列です。
$ml_user	dblsn -u によって指定される Mobile Link ユーザ名、またはデフォルト名 (*device-name*-**dblsn**)。
$ml_password	dblsn -w によって指定される Mobile Link ユーザ名のパスワード、または -y が使用された場合は新しい Mobile Link ユーザ名のパスワード。
$priority	この変数の意味は、carrier ライブラリに依存します。
$request_id	Push 要求用に指定された要求 ID。「Push 要求」 10 ページを参照してください。

action 変数	説明
$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 によるフィルタリング」 31 ページを参照してください。
$sender	メッセージの送信側。
$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 ー 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 ー 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 ー 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 ー 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 ー 59 までです。
$best_adapter_mac	dblsn コマンド・ラインに -x オプションで指定された 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 コマンド・ラインに -x オプションで指定された 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 コマンド・ラインに -x オプションで指定された 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 コマンド・ラインに -x オプションで指定された Mobile Link サーバに到達するための最善のプロファイルの RAS 名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、それぞれ縦線 (|) で分割します。
$network_names	接続 RAS エントリ名のリストで、それぞれ縦線 (|) 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。

例

たとえば、メッセージが message_start *pub-name* という形式で届いた場合、次の $message_end action 変数を使用して、どのパブリケーションを同期するかを決定します。

```
-l "message_start=message_start;action='dbmlsync.exe -c ... -n $message_end'"
```

受信ライブラリ

Windows Listener を実行すると、デフォルトで、受信ライブラリ *lsn_udp.dll* が使用されます。SMTP 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。

受信ライブラリを指定するには dblsn -d オプションを使用します。受信ライブラリのオプションを指定するには、-a オプションを使用します。マルチ・チャネル受信を可能にするには、-d を繰り返して複数の DLL を指定します。-d オプションの後に、DLL に関連する -a オプションと -i オプションを指定します。次に例を示します。

```
dblsn.exe -d lsn_udp.dll -i 10 -d maac750.dll -i 60
```

複数のオプションを指定するには、-a を繰り返します。次に例を示します。

```
-d maac750.dll -a port=2439 -a ShowSenderPort
```

DLL 用のオプションを確認するには、dblsn -d *filename.dll* -a ? と入力します。

サポートされている受信ライブラリとオプションのリストは、次のとおりです。

UDP (lsn_udp.dll)

オプション	説明
Port=*port_number*	デフォルトは 5001 です。
Timeout=*seconds*	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	:*port* を送信側に追加します。
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*	CE では、このコード・ページ番号に基づいてマルチバイト文字をユニコードに変換します。

AirCard510 用 SMS (lsn_swi510.dll)

オプション	説明
MessageStoreSize=*number*	このサイズは、ライブラリが冗長メッセージを圧縮する方法に影響します。メッセージの格納領域が満杯になると、ライブラリは同一メッセージの圧縮を停止し、メッセージが消費されるまで待ちます。デフォルトは 20 です。
NetworkProviderId=*name*	一致する Carrier(*name*).network_provider_id です。この情報は、デバイス・トラッキング同期中に送信されます。このオプションはデバイス・トラッキングで必要です。

オプション	説明
PhoneNumber=*number*	10 桁の電話番号。この情報は、デバイス・トラッキング同期中に送信されます。このオプションはデバイス・トラッキングで必要です。

AirCard555 用 SMS (maac555.dll)

オプション	説明
MessageStoreSize=*number*	このサイズは、ライブラリが冗長メッセージを圧縮する方法に影響します。メッセージの格納領域が満杯になると、ライブラリは同一メッセージの圧縮を停止し、メッセージが消費されるまで待ちます。デフォルトは 20 です。
PreserveMessage	他の SMS アプリケーション用に、クエリ内にメッセージを残すように指定します。デフォルトは Listener 用で、メッセージが処理されるときに消去するようにします。

ファームウェア R2 を使用する AirCard710 と AirCard750 用の SMS (maac750.dll)

オプション	説明
MessageStoreSize=*number*	このサイズは、ライブラリが冗長メッセージを圧縮する方法に影響します。メッセージの格納領域が満杯になると、ライブラリは同一メッセージの圧縮を停止し、メッセージが消費されるまで待ちます。デフォルトは 20 です。
PreserveMessage	他の SMS アプリケーション用に、キュー内にメッセージを残すように指定します。デフォルトは Listener 用で、メッセージが処理されるときに消去するようにします。

ファームウェア R3 を使用する AirCard710 と AirCard750 用の SMS (maac750r3.dll)

オプション	説明
MessageStoreSize=*number*	このサイズは、ライブラリが冗長メッセージを圧縮する方法に影響します。メッセージの格納領域が満杯になると、ライブラリは同一メッセージの圧縮を停止し、メッセージが消費されるまで待ちます。デフォルトは 20 です。
PreserveMessage	他の SMS アプリケーション用に、キュー内にメッセージを残すように指定します。デフォルトは Listener 用で、メッセージが処理されるときに消去するようにします。

第 4 章

Palm デバイス用 Listener

目次

Palm Listener ユーティリティ

Palm デバイスでのサーバ起動同期を実行するには、次の 2 つのユーティリティを使用します。

- Palm Listener 設定ユーティリティ (dblsncfg)
- Palm Listener (LsnT600.prc または LsnT650.prc)

まず Windows デスクトップで Palm 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ます。この設定ファイルは、後で HotSync を通じて Palm デバイスに転送する必要があります。

Listener とメッセージ・ハンドラの概要については、「Listener」 28 ページ を参照してください。

Palm Listener 設定ユーティリティ

Palm Listener 設定ユーティリティは、Windows デスクトップ上で実行し、Palm 用の設定ファイルを作成します。「Palm Listener ユーティリティ」 52 ページを参照してください。

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*]

オプションとパラメータ

@data　指定された環境変数または設定ファイルからオプションを読み込みます。環境変数と設定ファイルが両方とも存在する場合は、環境変数が使用されます。「Listener オプションの保存」 34 ページを参照してください。

-n [filename]　-n オプションは、Palm Listener 用の設定ファイルを作成するのに使用します。*filename* は、*lsncfg.pdb* にしてください。

-l message-handler　-l オプションを使用すると、フィルタとアクションのペアであるメッセージ・ハンドラを指定できます。このフィルタは、処理するメッセージを判断します。アクションは、フィルタがメッセージと一致したときに呼び出されます。-l のインスタンスは複数指定できます。-l の各インスタンスは、異なるメッセージ・ハンドラを指定します。

filter　受信メッセージとの比較を行うフィルタを指定します。フィルタが一致すると、指定したアクションが呼び出されます。

「subject と content フィルタの使用」 30 ページと「フィルタ message、message_start、sender の使用」 31 ページを参照してください。

このフィルタはオプションです。フィルタを指定しない場合は、メッセージの受信時にアクションが行われます。

action
action は、指定したアプリケーションを完全に起動します。構文は、**run** *application-name* [*arguments*] です。*arguments* はアプリケーションによって異なる文字列で、action 変数を含む場合があります。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。「Palm action 変数」 51 ページを参照してください。

注意 : Windows デスクトップ上で Palm Listener 設定ユーティリティを実行して Palm 用の設定ファイルを生成する場合は、**run** アクションを指定してください。ただし、Palm デバイスでは、Palm 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、アクションを発生させずにメッセージを消費できます。

Palm action 変数

次に示す Palm action 変数は、run 句内の引数で使用できます。

action 変数は、action が実行される直前に置き換えられます。

Listener の 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、**$$** と入力します。たとえば、$message_start が置き換えられないようにするときは、$$message_start と入力します。

action 変数	説明
$subject	メッセージの件名。
$object	メッセージのオブジェクト。
$message	メッセージ文字列全体。
$message_start	-l message_start で指定された、メッセージ・テキストの冒頭の一部。この変数を使用できるのは、-l message_start を指定した場合だけです。
$message_end	-l message_start で指定された部分が削除された後に残ったメッセージ部分。この変数を使用できるのは、-l message_start を指定した場合だけです。
$sender	メッセージの送信側。
$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。

Palm Listener ユーティリティ

サーバ起動同期を使用する Palm アプリケーションの場合、各クライアントには Palm Listener がインストールされている必要があります。Listener ファイルは次のとおりです。

- **LsnT600.prc**　Treo 600 用の Listener
- **LsnT650.prc**　Treo 650 用の Listener

現在、Palm Listener は設定ファイル *lsncfg.pdb* だけを読み込むことができます。

また、Palm Listener を使用すると、3 つのオプションを設定することもできます。これらのオプションは、明示的に変更されるか、リセットを実行するまで有効です。

- **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、リスニングがオンになっているときにのみ使用できます。無効にする場合、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ダイアログがポップアップ表示されます。

SMS メッセージを受信したときに自動的に電源がオンになる場合は、デバイスの電源をオンにしておく必要はありません。Treo のデバイスは、Listener を動作させるために電源をオンにしておく必要はありません。

Listener SDK は、他の Palm デバイスのサポートを作成するのに使用できます。「Palm 用 Mobile Link Listener SDK」 89 ページを参照してください。

第 5 章

Mobile Link 通知プロパティ

目次

共通プロパティ

verbosity という共通プロパティが 1 つあります。

プロパティの設定方法の詳細については、「プロパティの設定」 14 ページを参照してください。

verbosity プロパティ

verbosity 設定は、すべての Notifier、ゲートウェイ、Carrier に適用されます。verbosity は次のレベルに設定できます。

レベル	説明
0	トレーシングなし (デフォルト)
1	起動、シャットダウン、プロパティのトレーシング
2	通知メッセージを表示
3	ポーリング・レベルのトレーシング

参照

- 「プロパティの設定」 14 ページ.

Notifier プロパティ

Notifier プロパティは、Notifier プロパティ・ファイルに設定するか、Mobile Link システム・テーブルに格納できます。enable プロパティと request_cursor プロパティは必須です。その他の Notifier のプロパティは、すべてオプションです。

Notifier プロパティには、次の 2 種類があります。

Notifier イベント

confirmation_handler イベントは、その他の Notifier イベントとは非同期で発生します。これらのイベントはすべてオプションです。

- 「begin_connection プロパティ」 56 ページ
- 「confirmation_handler プロパティ」 56 ページ
- 「end_connection プロパティ」 59 ページ

Notifier ポーリング・イベント

Notifier は、統合データベースをポーリングするときに、ポーリング・イベントを次の順序で起動します。request_cursor を除き、これらのイベントはすべてオプションです。

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

- 「begin_poll プロパティ」 59 ページ
- 「end_poll プロパティ」 60 ページ
- 「error_handler プロパティ」 60 ページ
- 「request_cursor プロパティ」 62 ページ
- 「request_delete プロパティ」 62 ページ
- 「shutdown_query プロパティ」 63 ページ

Notifier 動作プロパティ

さまざまな設定を使用するように Notifier を設定することもできます。これらのプロパティはすべてオプションです。

- 「connect_string プロパティ」 63 ページ
- 「enable プロパティ」 63 ページ
- 「gui プロパティ」 64 ページ
- 「isolation プロパティ」 64 ページ
- 「poll_every プロパティ」 64 ページ

参照

- 「Notifier」 17 ページ
- 「プロパティの設定」 14 ページ

Notifier イベント

begin_connection プロパティ

これは、Notifier がデータベースに接続してから最初のポーリングを行うまでに、別のトランザクションで実行される SQL 文です。たとえば、このプロパティはテンポラリ・テーブルや変数の作成に使用できます。

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、このトランザクションを再度実行します。

このプロパティを使用して、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。

参照

- 「プロパティの設定」 14 ページ
- 「isolation プロパティ」 64 ページ

confirmation_handler プロパティ

このプロパティを実装して、リモートの Listener がアップロードした配信確認情報をプログラムで処理できます。ステータス・パラメータが 0 の場合、request_id により識別された Push 要求は、remote_device パラメータにより識別された Listener によって正常に受信されています。

出力パラメータ request_option を使用すると、配信確認への応答として適切なアクションを実行できます。request_option が 0 の場合、confirmation_handler はデフォルトの Notifier アクションを実行します。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。一方、配信確認を送信する Listener デバイスが、request_id により識別された Listener 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 要求はセカンダリ・ゲートウェイに送信されます。

注意： リモート 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。配信確認を実行し、IP 追跡を実行しないようにする場合は、dblsn -ni オプションを使用します。

「Listener 構文」 38 ページにある -x と -ni の説明を参照してください。

次に confirmation_handler パラメータを示します。パラメータは、すべて使用しても一部だけ使用してもかまいません。このプロパティにはストアド・プロシージャの使用が必要です。

スクリプト・パラメータ	説明
request_option (out)	整数。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。次のいずれかです。 ◆ **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。status が示す応答デバイスがターゲット 1 の場合、Notifier は要求を削除します。それ以外の場合は、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)	整数。状況の概要です。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特定に使用できます。ステータスは次のいずれかです。 ◆ **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したが、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)	整数。要求を識別します。
remote_code (in)	整数。リモート Listener からレポートされた概要です。次のいずれかです。 ◆ **1**：メッセージは受け入れられます。 ◆ **2**：メッセージは拒否されます。 ◆ **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す。 ◆ **4**：メッセージは受け入れられ、アクションは失敗します。
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。
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

- デバイス・トラッカ・ゲートウェイ：「confirm_delivery プロパティ」 66 ページ
- SMTP ゲートウェイ：「confirm_delivery プロパティ」 68 ページ
- UDP ゲートウェイ：「confirm_delivery プロパティ」 72 ページ
- 「ml_add_property」 『Mobile Link - サーバ管理』

例

次の例では、CustomConfirmation というテーブルを作成し、CustomConfirmationHandler というストアド・プロシージャを使用して確認をログ記録します。この例では、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
```

end_connection プロパティ

これは、Notifier データベース接続が終了する直前に、独立したトランザクションとして実行される SQL 文です。たとえば、このプロパティは SQL 変数やテンポラリ・テーブルなどの、一時的な記憶領域の削除に使用できます。

この文は、スタンドアロン・トランザクションで実行されます。

Notifier ポーリング・イベント

begin_poll プロパティ

これは、各 Notifier ポーリングの前に実行される SQL 文です。通常は、データベースでのデータの変更を検出し、後で request_cursor でフェッチされる Push 要求を作成するのに使用します。

この文は、スタンドアロン・トランザクションで実行されます。

このプロパティはオプションです。デフォルト値は NULL です。

参照

- 「プロパティの設定」 14 ページ.

例

この例では、Notifier A という 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。この WHERE 句は、PushRequest テーブルに挿入される Push 要求を決定します。

ストアド・プロシージャ ml_add_property を SQL Anywhere 統合データベースで使用するには、次のコマンドを使用します。

```
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 プロパティ

これは、各ポーリングの後に実行される SQL 文です。通常は、カスタマイズされたクリーンアップまたはポーリングの追跡を実行するために使用されます。

この文は、スタンドアロン・トランザクションで実行されます。

このプロパティはオプションです。デフォルト値は NULL です。

error_handler プロパティ

このプロパティを実装すると、転送が失敗したり確認されなかったりした状況を捕捉できます。たとえば、転送が失敗した場合に、監査テーブルに行を挿入したり、通知を特定の宛先に送信したりできます。

捕捉できる情報を次に示します。パラメータは、すべて使用しても一部だけ使用してもかまいません。このプロパティにはストアド・プロシージャの使用が必要です。

スクリプト・パラメータ	説明
request_option (out)	整数。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。次のいずれかです。 ◆ 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：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。
error_code (in)	整数。次のいずれかです。 ◆ -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)	整数。要求を識別します。
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

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

例

次の例では、CustomError という名前のテーブルを作成します。ここでは、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。この例では、出力パラメータ notifier_opcode は常に 0 で、デフォルト 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
```

ストアド・プロシージャ ml_add_property を SQL Anywhere 統合データベースで使用するには、次のコマンドを使用します。

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

request_cursor プロパティ

このプロパティには、Notifier が Push 要求をフェッチするのに使用する SQL が含まれています。各ローは Push 要求で、どの情報をメッセージで送信するか、誰が、いつ、どこで情報を受信するかを決定します。このプロパティの設定は必須です。

この文の結果セットには、少なくとも 5 つのカラムが含まれていなければなりません。また、必要に応じて、その他の 2 つのカラムが含まれていることもあります。これらのカラムには名前を付けることもできますが、結果セットでは次の順序になっている必要があります。

- request id
- gateway
- subject
- content
- address
- resend interval (オプション)
- time to live (オプション)

カラムの詳細については、「Push 要求テーブルの作成」 10 ページを参照してください。

条件に一致する要求を除外するために、WHERE 句を request_cursor に追加することもできます。たとえば、要求を挿入した時刻を追跡するカラムを Push 要求テーブルに追加すると、前回の同期よりも前に挿入された要求を WHERE 句を使用して除外できます。

この文は、スタンドアロン・トランザクションで実行されます。

request_delete プロパティ

これは、クリーンアップ操作を指定する SQL 文です。この文は、パラメータとして要求 ID のみを取ります。パラメータは、名前付きパラメータか疑問符 (?) を使用して参照できます。

DELETE 文を使用すると、Notifier はこれらの古い要求を自動的に削除できます。

- **暗黙的に除外された要求**　以前発生したが、request_cursor で取得された現在の要求セットにはない要求。
- **確認済み要求**　配信時に確認されたメッセージ。
- **有効期限が切れた要求**　resend 属性と現在の時刻に基づき、有効期限が切れている要求。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 文は、削除要求が検出されたときに、スタンドアロン・トランザクションで要求 ID ごとに実行されます。クリーンアップを行う別のプロセスを提供している場合、この処理はオプションです。

request_delete スクリプトは、有効期限が切れた要求や暗黙に除外された要求を削除しないように記述できます。たとえば、CarDealer サンプルでは、request_delete 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 = ?
```

CarDealer サンプルの begin_poll スクリプトでは、最後の同期時間を利用して、処理済み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ことをチェックしています。

詳細については、*samples-dir¥MobiLink¥SIS_CarDealer* にある Car Dealer サンプル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。

shutdown_query プロパティ

これは、begin_poll の直後に実行される SQL 文です。結果には、yes (または 1) か no (または 0) の値のみが含まれています。Notifier を停止するには、yes または 1 を指定します。この文は、スタンドアロン・トランザクションとして実行されます。

テーブルに停止のステータスを格納している場合は、end_connection プロパティを使用して、Notifier によって切断される前にステータスをリセットできます。

Notifier 動作プロパティ

connect_string プロパティ

デフォルトでは、Notifier は ianywhere.ml.script.ServerContext を使用して統合データベースに接続します。つまり、Notifier は現在の mlsrv10 セッションのコマンド・ラインで指定された接続文字列を使用するということです。

これは、デフォルトの接続動作を無効にするために使用可能なオプションのプロパティです。このプロパティを使用すると、統合データベースを含め、任意のデータベースに接続できます。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。

「ServerContext インタフェース」『Mobile Link - サーバ管理』を参照してください。

参照

- 「プロパティの設定」 14 ページ.

enable プロパティ

既存の Notifier を有効または無効にできます。複数の Notifier を有効にしている場合、-notifier オプションで Mobile Link サーバを起動するとすべてが開始されます。

参照

- 「プロパティの設定」 14 ページ

gui プロパティ

このプロパティでは、Notifier が実行されているコンピュータで Notifier ファイル・ダイアログを表示するかどうかを指定します。このユーザ・インタフェースを使用すると、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)。

このプロパティはオプションです。デフォルトは ON です。

参照

- 「プロパティの設定」 14 ページ

isolation プロパティ

isolation は、Notifier のデータベース接続の独立性レベルを指定するオプションのプロパティです。デフォルト値は 1 です。次の値を使用できます。

値	独立性レベル
0	コミットされない読み込み
1	コミットされた読み込み (デフォルト)
2	繰り返し可能読み出し
3	直列可能

戻り値

独立性レベルの設定結果に注意してください。レベルが高くなると競合が増加し、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 0 に設定すると、コミットされていないデータ (最終的にロールバックされるデータ) を読み込むことができます。

参照

- 「プロパティの設定」 14 ページ

poll_every プロパティ

このプロパティでは、Notifier のポーリング間隔を指定します。秒、分、時間の単位として S、M、H を指定できます。また、1H 30M 10S のように単位を組み合わせることも可能です。単位が指定されていない場合、間隔は秒単位になります。

データベース接続が失われた場合、Notifier はデータベースが再び使用可能になった後に、この最初のポーリング間隔で自動的にリカバリを行います。

このプロパティはオプションです。デフォルトは 30 秒です。

shared_database_connection プロパティ

このオプションは、Notifiers で接続を共有する場合にオンに設定します。

Notifier 間で接続を共有すると、使用システム・リソースが減ります。パフォーマンスへの悪影響はありません。ただし、状況によっては、接続を共有できません。たとえば、アプリケーションが使用している SQL 変数名が Notifier 間で一意でない場合が該当します。

Notifier が接続を共有できるのは、独立性レベルが同じ場合だけです。

デフォルトはオフです。

参照

- 「isolation プロパティ」 64 ページ

ゲートウェイ・プロパティ

デバイス・トラッカ・ゲートウェイ・プロパティ

デフォルトのデバイス・トラッカ・ゲートウェイを使用する場合、**Default-Device Tracker** という名前を request_cursor の結果セットの 2 番目のカラムに入力します。

デバイス・トラッカ・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 UDP または SMTP ゲートウェイを介してメッセージを配信します。設定では、デバイス・トラッカ・ゲートウェイで使用する UDP ゲートウェイと SMTP ゲートウェイを定義します。また、このゲートウェイを介して送信されるメッセージのトラッキング要件も制御できます。

confirm_delivery プロパティ

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。これを実行するには、-x オプションで Listener を起動してください。confirm_delivery のデフォルトの設定は yes です。

サーバでの配信確認の処理の詳細については、「confirmation_handler プロパティ」 56 ページを参照してください。

次に例を示します。

```
DeviceTracker(Default-DeviceTracker).confirm_delivery = yes
```

参照

- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

enable プロパティ

デバイス・トラッカ・ゲートウェイを使用するには、**enable=yes** と指定します。デバイス・トラッカ・ゲートウェイを無効にするには、**enable=no** と指定します。複数のデバイス・トラッカ・ゲートウェイを定義して使用することもできます。

参照

- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

smtp_gateway プロパティ

これは、デバイス・トラッカが使用できる SMTP ゲートウェイを命名します。このゲートウェイは有効にしておく必要があります。デバイス・トラッカ・ゲートウェイが使用できる SMTP ゲートウェイは、1 つだけです。デフォルトは Default-SMTP です。

参照

- 「SMTP ゲートウェイ・プロパティ」 67 ページ
- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

sync_gateway プロパティ

これは、デバイス・トラッカが使用できる SYNC ゲートウェイを指定します。このゲートウェイは有効にしておく必要があります。デバイス・トラッカ・ゲートウェイが使用できる SYNC ゲートウェイは、1 つだけです。

SYNC ゲートウェイは TCP/IP ベースのゲートウェイで、永続的接続経由の通知をサポートします。これが推奨されるゲートウェイです。

SYNC ゲートウェイには、Default-SYNC という名前の事前に定義されたインスタンスが存在します。このインスタンスは、Default-DeviceTracker という名前の事前に定義されたデバイス・トラッカ・ゲートウェイで動作するよう設定されています。Default-SYNC ゲートウェイが Default-DeviceTracker ゲートウェイ経由で使用されている場合、配信確認はデフォルトでオンになります。

参照

- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

udp_gateway プロパティ

これは、デバイス・トラッカが使用できる UDP ゲートウェイを識別します。このゲートウェイは有効にしておく必要があります。デバイス・トラッカ・ゲートウェイが使用できる UDP ゲートウェイは、1 つだけです。デフォルトは Default-UDP です。

参照

- 「UDP ゲートウェイ・プロパティ」 71 ページ
- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

SMTP ゲートウェイ・プロパティ

SMTP ゲートウェイの設定が必要なのは、SMTP を介して SMS メッセージを送信する必要がある場合のみです。

SMTP ゲートウェイは、電子メール・メッセージの送信に使用できます。特に、無線通信事業者が提供する電子メールから SMS メッセージへの変換サービスを通じて、SMS メッセージを SMS リスナに送信できます。

以下のプロパティのリストでは、enable プロパティと server プロパティは必須です。server プロパティと sender プロパティは、必須の場合があります。user プロパティと password プロパティは、SMTP サーバの設定によっては必須の場合があります。その他の SMTP ゲートウェイ・プロパティは、すべてオプションです。

複数の SMTP ゲートウェイを使用できます。追加の SMTP ゲートウェイを設定するには、あるゲートウェイのプロパティをコピーし、別のゲートウェイ名とプロパティ値を指定します。

ゲートウェイの詳細については、「ゲートウェイと Carrier」 19 ページを参照してください。

confirm_delivery プロパティ

配信を確認する場合は、yes を指定します。デフォルトは no です。このプロパティは、このゲートウェイを介して直接送信する場合にのみ影響します (デバイス・トラッキング・ゲートウェイを介して間接的に送信する場合には効果はありません)。

参照

- 「confirmation_handler プロパティ」 56 ページ
- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

confirm_timeout プロパティ

確認がタイムアウトになるまでの時間を指定します。秒、分、時間の単位として s、m、h で指定します。s、m、h を指定しない場合、デフォルトは秒になります。

デフォルトの確認タイムアウト値は、10 m です。

参照

- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

description プロパティ

このプロパティはオプションで、ゲートウェイに関する説明の記述に使用します。

参照

- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

enable プロパティ

SMTP ゲートウェイを使用するには、enable=yes と設定します。複数の SMTP ゲートウェイを定義して使用することができます。

参照

- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

listeners_are_900 プロパティ

Listener が Adaptive Server Anywhere バージョン 9.0.0 クライアントの場合は、yes を指定します。バージョン 9.0.1 以降の場合は、no を指定します。デフォルトは no です。

参照

- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

password プロパティ

これは、使用している SMTP サービスのパスワードです。SMTP サービスには、パスワードを必要としないものもあります。

参照

- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

sender プロパティ

これは、電子メール (SMTP 要求) の送信元のアドレスです。デフォルトは anonymous です。

送信元は、受信メッセージが Mobile Link のメッセージ解釈と互換性のないフォーマットの場合、Listener の action 変数として使用できません。

参照

- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

server プロパティ

これは、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です。デフォルトは **mail** です。

参照

- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

user プロパティ

これは、使用している SMTP サービスのユーザ名です。SMTP サービスには、ユーザ名を必要としないものもあります。

参照

- 「デバイス・トラッキング」 21 ページ
- 「プロパティの設定」 14 ページ

SYNC ゲートウェイ・プロパティ

SYNC ゲートウェイの設定が必要なのは、同期と同じプロトコルで通知を送信する必要がある場合だけです。

デフォルトの SYNC ゲートウェイが用意されており、デフォルトで有効になっています。デフォルト・ゲートウェイを使用できるのは、Notifier 設定が統合データベースに保存されている (かつ Notifier プロパティ・ファイルにない) 場合だけです。

SYNC ゲートウェイは永続的接続を使用します。同じ接続が通知、確認、デバイス・トラッキングで使用されます。推奨されるのは SYNC ゲートウェイです。デバイス・トラッキングを使用する場合、通知の試行は SYNC ゲートウェイから始まり、UDP ゲートウェイ、SMTP ゲートウェイの順にフォールバックします。

以下のプロパティのリストでは、enable プロパティのみが必須です。その他の SYNC ゲートウェイ・プロパティは、すべてオプションです。

複数の SYNC ゲートウェイを使用できます。追加の SYNC ゲートウェイを設定するには、あるゲートウェイのプロパティをコピーし、別のゲートウェイ名とプロパティ値を指定します。

ゲートウェイの詳細については、「ゲートウェイと Carrier」 19 ページを参照してください。

プロパティの設定方法の詳細については、「プロパティの設定」 14 ページを参照してください。

confirm_action プロパティ

発信確認が必要な場合に yes を指定します。このプロパティは、このゲートウェイを介して直接送信する場合にのみ影響します。デバイス・トラッキング・ゲートウェイを介して間接的に送信する場合には影響しません。

デフォルトは no です。

参照

- 「confirmation_handler プロパティ」 56 ページ

- 「プロパティの設定」 14 ページ

confirm_delivery プロパティ

配信を確認する場合は、yes を指定します。デフォルトは no です。このプロパティは、このゲートウェイを介して直接送信する場合にのみ影響します (デバイス・トラッキング・ゲートウェイを介して間接的に送信する場合には効果はありません)。

参照

- 「confirmation_handler プロパティ」 56 ページ
- 「プロパティの設定」 14 ページ

confirm_timeout プロパティ

確認がタイムアウトになるまでの時間を指定します。秒、分、時間の単位として s、m、h で指定します。s、m、h を指定しない場合、デフォルトは秒になります。

デフォルトの確認タイムアウト値は、1 m です。

参照

- 「プロパティの設定」 14 ページ

description プロパティ

このプロパティはオプションで、ゲートウェイに関する説明の記述に使用します。

参照

- 「プロパティの設定」 14 ページ

enable プロパティ

SYNC ゲートウェイを使用するには、**enable=yes** と設定します。

参照

- 「プロパティの設定」 14 ページ

UDP ゲートウェイ・プロパティ

UDP ゲートウェイの設定が必要なのは、UDP メッセージを送信する必要がある場合のみです。

UDP メッセージのフォーマットは、[*subject*] *content* です。この *subject* と *content* は、Notifier のプロパティ request_cursor の subject カラムと content カラムから取得されます。

以下のプロパティのリストでは、enable プロパティのみが必須です。その他の UDP ゲートウェイ・プロパティは、すべてオプションです。

複数の UDP ゲートウェイを使用できます。追加の UDP ゲートウェイを設定するには、あるゲートウェイのプロパティをコピーし、別のゲートウェイ名とプロパティ値を指定します。

ゲートウェイの詳細については、「ゲートウェイと Carrier」 19 ページを参照してください。

プロパティの設定方法の詳細については、「プロパティの設定」 14 ページを参照してください。

confirm_delivery プロパティ

配信を確認する場合は、yes を指定します。デフォルトは yes です。このプロパティは、このゲートウェイを介して直接送信する場合のみ影響します (デバイス・トラッキング・ゲートウェイを介して間接的に送信する場合は含まれません)。

参照

- 「confirmation_handler プロパティ」 56 ページ
- 「プロパティの設定」 14 ページ

confirm_timeout プロパティ

確認がタイムアウトになるまでの時間を指定します。秒、分、時間の単位として s、m、h で指定します。s、m、h を指定しない場合、デフォルトは秒になります。

デフォルトの確認タイムアウト値は、1 m です。

参照

- 「プロパティの設定」 14 ページ

description プロパティ

このプロパティはオプションで、ゲートウェイに関する説明の記述に使用します。

参照

- 「プロパティの設定」 14 ページ

enable プロパティ

UDP ゲートウェイを使用するには、**enable=yes** と設定します。複数の UDP ゲートウェイを定義して使用することができます。

参照

- 「プロパティの設定」 14 ページ

listeners_are_900 プロパティ

Listener が Adaptive Server Anywhere バージョン 9.0.0 クライアントの場合は、yes を指定します。バージョン 9.0.1 以降の場合は、no を指定します。デフォルトは no です。

参照

- 「プロパティの設定」 14 ページ

listener_port プロパティ

これは、ゲートウェイが UDP パケットを送信するリモート・デバイスのポートです。このプロパティはオプションです。デフォルトは、UDP Listener (5001) のデフォルト受信ポートです。

参照

- 「プロパティの設定」 14 ページ

sender プロパティ

これは、送信元の IP アドレスまたはホスト名です。このプロパティはオプションですが、マルチホームのホストの場合にのみ有効です。デフォルトは localhost です。

参照

- 「プロパティの設定」 14 ページ

sender_port プロパティ

これは、UDP パケットの送信に使用されるポートです。このプロパティはオプションです。ファイアウォールで出力側のトラフィックを制限している場合は、このプロパティを設定しなければならないことがあります。設定しない場合、オペレーティング・システムによってフリー・ポートが割り当てられます。

参照

- 「プロパティの設定」 14 ページ

Carrier プロパティ

Carrier は、SMTP ゲートウェイを使用する場合にのみ必要です。

Carrier プロパティは、公衆無線通信事業者の設定を行うもので、電話番号自動追跡のマッピングやネットワーク・プロバイダに対する SMS 電子メール・アドレスのマッピング方法などの、Carrier 固有の情報を提供します。

Carrier 情報は、デバイス・トラッカ・ゲートウェイが自動追跡デバイス・アドレスから SMS 電子メール・アドレスを生成する必要がある場合に使用されます。アドレスは、次の形式で生成されます。

email-address =
sms_email_user_prefix*phone-number****@sms_email_domain***

文中の各項目を次に説明します。

- *sms_email_user_prefix* は、sms_email_user_prefix プロパティの値です。
- 電話番号は、ml_device_address.address カラムから生成されます。
- *sms_email_domain* は、sms_email_domain プロパティの値です。

参照

- 「sms_email_domain プロパティ」 75 ページ
- 「sms_email_user_prefix プロパティ」 75 ページ
- 「ml_device_address」 『Mobile Link - サーバ管理』

Carrier の詳細については、「ゲートウェイと Carrier」 19 ページを参照してください。

プロパティの設定方法の詳細については、「プロパティの設定」 14 ページを参照してください。

enable プロパティ

Carrier マッピングを使用するには、**enable=yes** と設定します。複数の Carrier マッピングを 1 つのファイルで定義して使用できます。

参照

- 「プロパティの設定」 14 ページ

network_provider_id プロパティ

ネットワーク・プロバイダ ID を指定します。

SMS を CE Phone Edition で使用するには、ネットワーク・プロバイダ ID を _generic_ に設定します。次に例を示します。

```
network_provider_id=_generic_
```

参照

- 「プロパティの設定」 14 ページ

sms_email_domain プロパティ

Carrier のドメイン名を指定します。

Carrier 情報は、デバイス・トラッカ・ゲートウェイが自動追跡デバイス・アドレスから SMS 電子メール・アドレスを生成する必要がある場合に使用されます。アドレスは、次の形式で生成されます。

email-address =
sms_email_user_prefix*phone-number*@***sms_email_domain***

文中の各項目を次に説明します。

- ***sms_email_user_prefix*** は、sms_email_user_prefix プロパティの値です。
- 電話番号は、ml_device_address.address カラムから生成されます。
- ***sms_email_domain*** は、sms_email_domain プロパティの値です。

参照

- 「sms_email_user_prefix プロパティ」 75 ページ
- 「ml_device_address」 『Mobile Link - サーバ管理』
- 「プロパティの設定」 14 ページ

sms_email_user_prefix プロパティ

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。

Carrier 情報は、デバイス・トラッカ・ゲートウェイが自動追跡デバイス・アドレスから SMS 電子メール・アドレスを生成する必要がある場合に使用されます。アドレスは、次の形式で生成されます。

email-address =
sms_email_user_prefix*phone-number*@***sms_email_domain***

文中の各項目を次に説明します。

- ***sms_email_user_prefix*** は、sms_email_user_prefix プロパティの値です。
- 電話番号は、ml_device_address.address カラムから生成されます。
- ***sms_email_domain*** は、sms_email_domain プロパティの値です。

参照

- 「sms_email_domain プロパティ」 75 ページ
- 「ml_device_address」 『Mobile Link - サーバ管理』

- 「プロパティの設定」 14 ページ

第 6 章

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・プロシージャの概要

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。

詳細については、「デバイス・トラッキング」 21 ページと「デバイス・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」 23 ページを参照してください。

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

その他の Mobile Link システム・プロシージャの詳細については、「Mobile Link サーバ・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。

ml_delete_device

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。

説明

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・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」 23 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

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)

説明

このシステム・プロシージャが役立つのは、デバイス・トラッキングを手動で設定する場合のみです。

「デバイス・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」 23 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

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)

説明

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・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」 23 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

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)。リスナ・バージョンのオプションの注釈。
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のトラッキングによる上書きを停止させる場合、**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。

説明

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・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」 23 ページを参照してください。

参照

- 「ml_set_device_address」 84 ページ
- 「ml_set_listening」 86 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

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)。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、**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のトラッキングによる上書きを停止させる場合、**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。

説明

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・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」 23 ページを参照してください。

参照

- 「ml_set_device」 82 ページ
- 「ml_set_listening」 86 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

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。
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のトラッキングによる上書きを停止させる場合、**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。

説明

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・トラッキングをサポートしない場合のデバイス・トラッキングと Listener の使用」 23 ページを参照してください。

参照

- 「ml_set_device」 82 ページ
- 「ml_set_device_address」 84 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 ユーザ名にマップします。

```
CALL ml_set_listening(
        'myULDB',
        'myFirstTreo180',
        'y',
        'y',
        'manually entered by administrator' );
```


第 7 章

Palm 用 Mobile Link Listener SDK

目次

Palm 用 Mobile Link Listener SDK の概要

Palm 用 Listener SDK を使用すると、新しい Palm デバイス用の Listener を作成できます。Listener SDK は、Listener ユーティリティを拡張できるようにするために用意された単純な API です。このプログラミング・インタフェースには、メッセージ処理用インタフェースとデバイス依存関数があります。Listener SDK を使用すると、新しい Palm デバイス用の Listener を作成したり、新しい無線ネットワーク・アダプタ用の Listener を作成したりできます。

Palm 用 Listener SDK ファイル

Mobile Link Listener SDK と実装サンプルは、次のファイルにあります。

Palm 用ファイル	説明
MobiLink¥ListenerSDK¥Palm¥68k¥cw¥lib¥PalmLsn.lib	Palm Listener のランタイム・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 を提供します。
MobiLink¥ListenerSDK¥Palm¥68k¥cw¥rsc¥	Palm Listener の UI リソースを含む。
MobiLink¥ListenerSDK¥Palm¥src¥PalmLsn.h	ランタイム・ライブラリのヘッダと Palm Listener API
MobiLink¥ListenerSDK¥Palm¥src¥Treo600.c	Treo 600 の実装
MobiLink¥ListenerSDK¥Palm¥src¥Treo650.c	Treo 650 の実装

メッセージ処理用インタフェース

メッセージ処理用インタフェースは、Palm Listener ライブラリ *PalmLsn.lib* に含まれています。

PalmLsn.lib の詳細については、「Palm 用 Listener SDK ファイル」 90 ページを参照してください。

a_palm_msg 構造体

Palm Listener SDK では、a_palm_msg 構造体を使用して Palm Listener メッセージを表現します。SDK のメッセージ処理インタフェースには、a_palm_msg 構造体インスタンスを割り当て、処理するための関数が用意されています。

概要

以下の関数を使用して、a_palm_msg の割り当て、メッセージ・フィールドの初期化、メッセージの処理を実行できます。

- **メッセージの割り当て**　メッセージの割り当てと割り当て解除には、次の関数を使用します。

  「PalmLsnAllocate 関数」 91 ページ

  「PalmLsnFree 関数」 92 ページ

- **メッセージ・フィールドの初期化**　a_palm_msg インスタンスのメッセージ、送信元、時刻の各フィールドに値を割り当てるには、次の関数を使用します。

  「PalmLsnDupMessage 関数」 92 ページ

  「PalmLsnDupSender 関数」 93 ページ

  「PalmLsnDupTime 関数」 94 ページ

- **メッセージの処理**　メッセージの各フィールドを処理し、アプリケーションを起動するには、PalmLsnProcess 関数を使用します。

  「PalmLsnProcess 関数」 95 ページを参照してください。

PalmLsnAllocate 関数

新しい a_palm_msg インスタンスを返します。

プロトタイプ

struct a_palm_msg * **PalmLsnAllocate( )**

戻り値

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

参照

- 「PalmLsnFree 関数」 92 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();
```

PalmLsnFree 関数

メッセージ用のメモリ領域を開放します。

プロトタイプ

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。

参照

- 「概要」 91 ページ

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してい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 );
```

PalmLsnDupMessage 関数

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。

プロトタイプ

```
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。

説明

PalmLsnDupMessage 関数は、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、それらの値を 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 関数」 93 ページ
- 「PalmLsnDupTime 関数」 94 ページ
- 「a_palm_msg 構造体」 91 ページ

例

次に示す例は、Treo 600 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 関数

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。

プロトタイプ

```
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。

説明

PalmLsnDupSender 関数は、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 関数」 92 ページ
- 「PalmLsnDupTime 関数」 94 ページ
- 「a_palm_msg 構造体」 91 ページ

PalmLsnDupTime 関数

a_palm_msg インスタンスの時刻フィールドを初期化します。

プロトタイプ

```
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。

説明

PalmLsnDupTime 関数は、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 関数」 92 ページ

- 「PalmLsnDupSender 関数」 93 ページ
- 「a_palm_msg 構造体」 91 ページ

PalmLsnProcess 関数

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。

プロトタイプ

```
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を取得するには、PalmLsnGetConfigFileName 関数を使用します。

  「PalmLsnGetConfigFileName」 101 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 列挙に定義されているリターン・コード。

「palm_lsn_ret 列挙」 97 ページを参照してください。

説明

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。

Palm Listener 設定データベースの作成方法の詳細については、「Palm Listener 設定ユーティリティ」 50 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 Listener 設定ユーティリティ」 50 ページ
- 「メッセージ・ハンドラ」 29 ページ
- 「action 変数」 45 ページ
- 「PalmLsnCheckConfigDB 関数」 96 ページ
- 「a_palm_msg 構造体」 91 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。このコード例では、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 );
```

PalmLsnCheckConfigDB 関数

Palm Listener 設定データベース内のエラーをレポートします。

プロトタイプ

```
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 関数を使用します。

  「PalmLsnGetConfigFileName」 101 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 列挙に定義されているリターン・コード。

「palm_lsn_ret 列挙」 97 ページを参照してください。

説明

この関数を使用して、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発生したエラーを検出できます。

参照

- 「PalmLsnProcess 関数」 95 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
}
```

palm_lsn_ret 列挙

palm_lsn_ret 列挙は、メッセージ処理のリターン・コードを指定します。

プロトタイプ

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

パラメータ

値	説明
PalmLsnOk	関数呼び出しが正常終了した。このフィールドには、errNone (エラーなしを表す Palm エラー・コード) と同じ値が格納されます。
PalmLsnMissingConfig	Palm Listener 設定データベースが見つからない。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 Palm エラー・コード appErrorClass と同じ値が格納されます。
PalmLsnProblemReadingConfig	Palm Listener 設定データベースの読み取りエラー。
PalmLsnProblemParsingCmd	Palm Listener 設定データベースに格納されているコマンドが処理できない。
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、関数の実行を継続できない。
PalmLsnUnrecognizedAction	Listener が、Palm Listener 設定データベースに指定されたアクションをサポートしていない。
PalmLsnRunMissingApp	Listener が、run アクションに指定されたアプリケーションを起動できない。

参照

- 「PalmLsnProcess 関数」 95 ページ

LsnMain 関数

Palm Listener ライブラリ *PalmLsn.lib* のメイン・エントリ・ポイントとなる関数です。

プロトタイプ

```
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。Palm 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。

説明

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 関数」 95 ページ
- 「Palm 用 Listener SDK ファイル」 90 ページ

例

次に示す例は、Treo 600 smartphone の実装での使用例です。Listener 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 ) );
}
```


デバイス依存関数

デバイス依存機能を指定するには、Palm Listener SDK に定義された関数群を使用します。これらの関数は次の機能を提供します。

- **識別情報**　Listener と設定データベースの識別情報を提供するには次の関数を使用します。

  「PalmLsnTargetCompanyID」 100 ページ

  「PalmLsnTargetDeviceID」 101 ページ

  「PalmLsnGetConfigFileName」 101 ページ

- **登録または初期化**　Listener を登録または登録解除するには、次の関数を使用します。

  「PalmLsnNormalStart」 102 ページ

  「PalmLsnNormalStop」 102 ページ

- **イベント処理**　アプリケーションのイベントを処理するには、次の関数を使用します。

  「PalmLsnNormalHandleEvent」 103 ページ

  デバイス依存の起動コードに応答するには、次の関数を使用します。

  「PalmLsnSpecialLaunch」 103 ページ

PalmLsnTargetCompanyID

デバイスの企業 ID を返します。

プロトタイプ

UInt32 **PalmLsnTargetCompanyID( )**

戻り値

デバイスの企業 ID または製造元 ID を表す値。

説明

PalmLsnTargetCompanyID と PalmLsnTargetDeviceID を使用すると、デバイスの互換性をチェックできます。

参照

- 「PalmLsnTargetDeviceID」 101 ページ

例

次に示す例は、Treo 600 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

ターゲット・デバイス ID を返します。

プロトタイプ

UInt32 **PalmLsnTargetDeviceID( )**

戻り値

デバイス ID を表す正の整数。

説明

PalmLsnTargetCompanyID と PalmLsnTargetDeviceID を使用すると、デバイスの互換性をチェックできます。

参照

- 「PalmLsnTargetCompanyID」 100 ページ

例

次の例では、Treo 600 シミュレータのデバイス ID を返しています。

```
UInt32 PalmLsnTargetDeviceID( void )
/*********************************/
{
    // Simulator device ID is hsDeviceIDOs5Device1Sim
    return( hsDeviceIDOs5Device1 );
}
```

PalmLsnGetConfigFileName

Palm Listener 設定データベースの名前を含む文字列を返します。

プロトタイプ

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

パラメータ

- **configPDBName**　Palm 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。

説明

この関数を使用すると、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 関数」 95 ページ

例

次に示す例は、Treo 600 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 );
}
```

PalmLsnNormalStart

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。

プロトタイプ

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。

説明

PalmLsnNormalStart は、Listener デバイスを登録する手段を提供します。

参照

- 「PalmLsnNormalStop」 102 ページ
- 「PalmLsnSpecialLaunch」 103 ページ

PalmLsnNormalStop

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。

プロトタイプ

void **PalmLsnNormalStop( )**

説明

受信を継続する場合は、PalmLsnNormalStop でデバイスを登録解除しないでください。この関数は、現在のアプリケーション設定を取得または設定する場合にも使用できます。

参照

- 「PalmLsnNormalStart」 102 ページ

PalmLsnNormalHandleEvent

アプリケーション・イベントを処理します。

プロトタイプ

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。

説明

この関数を使用して、アプリケーション・イベントを処理できます。

PalmLsnSpecialLaunch

デバイス依存の起動コードに応答します。

プロトタイプ

```
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。

説明

この関数は、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 600 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 phnLibLaunchCmdRegister:
   break;


case phnLibLaunchCmdEvent: {
   if( !IsFeatureOn( PalmLsnGetFeature(), Listening ) ) {
      return( errNone );
   }


 PhnEventPtr phoneEventP = (PhnEventPtr)cmdPBP;


 if( phoneEventP->eventType == phnEvtMessageInd ) {
      // handle the message
      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 );
}
```

索引

記号

A

B

C

D

E

F

G

I

L

M

N

P

R

S

と

に

は

ひ

ふ

へ

ま

め

ゆ

ら

り

