iAnywhere SOLUTIONS™
A SYBASE COMPANY

Mobile Link サーバ起動同期
ユーザーズ・ガイド

パート番号：DC20053-01-0902-01
改訂：2005 年 3 月

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 管理ガイド』> 「Mobile Link 同期について」を参照してください。

SQL Anywhere Studio のマニュアル

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

SQL Anywhere Studio のマニュアル

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

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

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

- **『Adaptive Server Anywhere SNMP Extension Agent 拡張エージェント ユーザーズ・ガイド』** このマニュアルでは、SNMP 管理アプリケーションとともに使用するように Adaptive Server Anywhere SNMP Extension Agent を設定して、Adaptive Server Anywhere データベースを管理する方法を説明します。

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

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

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

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

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

- **『Mobile Link チュートリアル』** このマニュアルには、Mobile Link アプリケーションの設定と実行を行う方法を説明するチュートリアルがいくつか用意されています。
- **『QAnywhere ユーザーズ・ガイド』** このマニュアルでは、Mobile Link QAnywhere について説明します。Mobile Link QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアントだけでなく、モバイル・クライアントや無線クライアント用のメッセージング・アプリケーションの開発と展開を可能にするメッセージング・プラットフォームです。
- **『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』** このマニュアルでは、Mobile Link 同期サーバから、または Adaptive Server Anywhere リモート・データ・アクセスによって、Adaptive Server Anywhere 以外の統合データベースにアクセスするための ODBC ドライバの設定方法について説明します。
- **『SQL Remote ユーザーズ・ガイド』** このマニュアルでは、モバイル・コンピューティング用の SQL Remote データ・レプリケーション・システムについて、あらゆる角度から説明します。このシステムによって、Adaptive Server Anywhere または Adaptive Server Enterprise の単一データベースと Adaptive Server Anywhere の複数データベースの間で、電子メールやファイル転送などの間接的リンクを使用したデータ共有が可能になります。
- **『SQL Anywhere Studio ヘルプ』** このマニュアルには、Sybase Central や Interactive SQL、その他のグラフィカル・ツールに関するコンテキスト別のヘルプが含まれています。これは、製本版マニュアル・セットには含まれていません。
- **『Ultra Light データベース・ユーザーズ・ガイド』** このマニュアルは、Ultra Light 開発者を対象としています。ここでは、Ultra Light データベース・システムの概要について説明します。また、すべての Ultra Light プログラミング・インタフェースに共通する情報を提供します。
- **Ultra Light のインタフェースに関するマニュアル** 各 Ultra Light プログラミング・インタフェースには、それぞれに対応するマニュアルを用意しています。これらのインタフェースは、RAD (

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

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

マニュアルの形式

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

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

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

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

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

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

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

表記の規則

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

SQL 構文の表記規則

SQL 構文の表記には、次の規則が適用されます。

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

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

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

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

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

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

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

- **オプション部分** 文のオプション部分は角カッコで囲みます。

  RELEASE SAVEPOINT [*savepoint-name*]

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

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

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

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

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

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

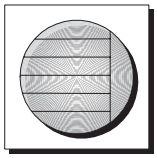
  QUOTES オプションを使用する場合は、ON または OFF のどちらかを選択する必要があります。角カッコと中カッコは入力しないでください。

グラフィック・アイコン

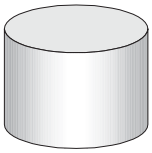
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

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



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



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

- プログラミング・インタフェース

詳細情報の検索／フィードバックの提供

詳細情報の検索

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

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

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

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

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

ニュースグループに関するお断り

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

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

フィードバック

このマニュアルに関するご意見、ご提案、フィードバックをお寄せください。

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

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

第 1 章

サーバ起動同期の概要

この章の内容　この章では、サーバ起動同期の概要を説明します。

Mobile Link サーバ起動同期の概要

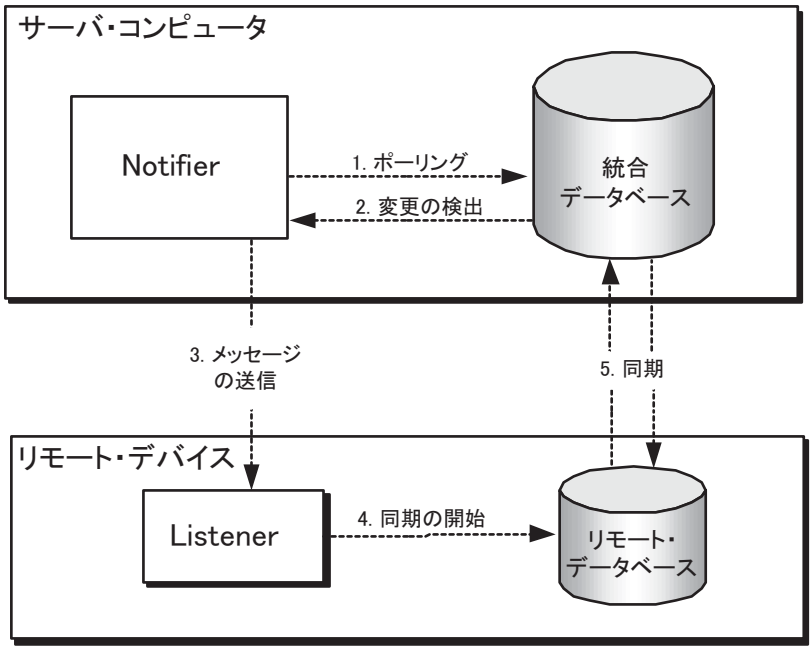
サーバ起動同期を使用すると、Mobile Link 同期を統合データベースから開始できます。これは、リモート・データベースが統合データベースにデータをアップロードできることに加え、データの更新をリモート・データベースにプッシュできることを意味しています。この Mobile Link コンポーネントには、どの変更内容が統合データベースで発生したら同期を開始するかを決定したり、プッシュするメッセージを受信するリモート・データベースを選択する方法やリモート・データベースの応答方法を決定したりするためのプログラム可能なオプションが用意されています。

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、UDP または SMTP を使用してメッセージを送信します。
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。

詳細については、「接続起動同期」42 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。

  詳細については、「Push 要求」12 ページを参照してください。

- **Notifier**　Mobile Link 同期サーバと同じコンピュータで実行される Java 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 SMTP または UDP を通じて、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」21 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。

詳細については、「Listener」49 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。

詳細については、「Palm デバイス用 Listener」65 ページ を参照してください。

- **ゲートウェイ** Notifier から Listener へメッセージを送信するためのインタフェースを提供します。メッセージは、SMTP ゲートウェイまたは UDP ゲートウェイ経由で送信できます。SMTP ゲートウェイを使用すると、送信した電子メール・メッセージは Listener が受信する前に Carrier によって SMS に変換されます。ほとんどの Carrier は、電子メールを SMS に変換するサービスを提供しています。

  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」24 ページを参照してください。

サポートされるプラットフォーム

Notifier を使用するには、Mobile Link 要件に加えて、コンピュータに JRE 1.4.1 以降がインストールされている必要があります。

Mobile Link 要件の詳細については、『SQL Anywhere Studio の紹介』>「SQL Anywhere Studio がサポートしているプラットフォーム」を参照してください。

Listener は、Windows 95 または Windows NT 4 ではサポートされていません。

対象のリモート・デバイスが Palm の場合は、Windows デスクトップ・デバイス上で Palm Listener 設定ユーティリティを使用して、設定ファイルを作成する必要があります。

- **SMS メッセージ**　SMTP ゲートウェイを通じて転送でき、無線通信事業者が提供する電子メールから SMS への変換が行われます。SMS メッセージは、次のプラットフォームでテスト済みです。
    - Sierra Wireless AirCard 510、555、710、または 750 を装備した Pocket PC 2002
    - Sierra Wireless AirCard 510、555、710、または 750 を装備した Windows 2000 と Windows XP
    - Kyocera 7135 上の Palm 4.1 および Treo 600 上の Palm 5.2
- **UDP メッセージ**　次のプラットフォームでテスト済みです。
    - Pocket PC 2002
    - Windows 2000 と Windows XP

各種 AirCard でサポートされているファームウェアとドライバのバージョンは次のとおりです (710 は 750 と互換性があります)。

AirCard	ファームウェアのバージョン	ドライバのバージョン
510	R1-3-4	該当なし

AirCard	ファームウェアのバージョン	ドライバのバージョン
555	R1_1_2_10AC_GEN	R1_0_0_9ac_1xRTT
750	R1_1_2_10AC_GEN	R1_0_7_ac_gprs
750	R3_1_17ACAP	R1_0_9_ac_gprs

配備に関する考慮事項

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、受信できるように IP ネットワークに接続されていなければなりません。
- リモート・デバイスの IP アドレスは、Mobile Link 同期サーバからアクセスできるものでなければなりません。

CE または PC に関する Listener の制限事項

- 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。

  詳細については、「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」31 ページを参照してください。

クイック・スタート

サーバ起動同期を設定するには、次の手順を実行します。この手順では、Mobile Link 同期がすでに設定されているものとします。

❖ 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」12 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」21 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」37 ページ を参照してください。

4 SMS 通知を送信する場合は、ゲートウェイと Carrier を設定して、メッセージの送信に使用するインタフェースを決定します (UDP を使用する場合は、この手順を省略して、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 通知を送信するには、Listener を起動する際に SMTP 受信ライブラリも指定する必要があります。

「ゲートウェイと Carrier」24 ページと「受信ライブラリ」61 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

- 「チュートリアル：サーバ起動同期」161 ページ
- SQL Anywhere Studio インストール・ディレクトリ内の *Samples¥MobiLink¥SIS_** に、サンプル・アプリケーションがあります。

第 2 章

サーバ起動同期の設定

この章の内容　この章では、サーバ起動同期の設定と使用方法について説明します。

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共に、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、次のカラムを使用してテーブルを統合データベースに追加します。ただし、Push 要求は、テンポラリ・テーブル内、および複数のテーブルにまたがって格納することもできます。

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。

カラム	説明
address	VARCHAR。送信先アドレス。アドレスのフォーマットは、ゲートウェイによって異なります。DeviceTracker ゲートウェイ (SMTP または UDP を使用) の場合は、Listener データベースの 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。SMTP ゲートウェイの場合、電子メール・アドレスです。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)。
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。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。

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」29 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 Adaptive Server 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。

  詳細については、「begin_poll プロパティ」74 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。

トリガの詳細については、『ASA SQL ユーザーズ・ガイド』> 「トリガの概要」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。

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

- 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」82 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」83 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

Adaptive Server 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。

詳細については、『ASA SQL リファレンス・マニュアル』> 「sa_send_udp システム・プロシージャ」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.yahoo.com"
```

別のデバイス上では、Adaptive Server 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、Yahoo 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 番目の引数に localhost 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

Mobile Link システム・テーブルにプロパティを設定するには、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。これは、統合データベースの ml_property テーブルにプロパティ設定を格納します。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。

- ストアド・プロシージャ ml_add_property を使用します。これも、統合データベースの ml_property テーブルにプロパティ設定を格納します。

  詳細については、『Mobile Link 管理ガイド』> 「ml_add_property」を参照してください。

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。

詳細については、「Notifier のプロパティ・ファイル」18 ページを参照してください。

プロパティ

設定できる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。

- 「Mobile Link 通知プロパティ」71 ページ
- 「デバイス・トラッカ・ゲートウェイ・プロパティ」87 ページ

- 「SMTP ゲートウェイ・プロパティ」90 ページ
- 「UDP ゲートウェイ・プロパティ」93 ページ
- 「Carrier プロパティ」96 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。プロパティを変更する場合、変更を有効にするには Mobile Link 同期サーバを停止し、再起動します。

Notifier のプロパティ・ファイル

Notifier、ゲートウェイ、および 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。詳細については、「プロパティの設定」17 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、テンプレート *%asany9%¥samples¥MobiLink¥template.notifier* を変更することです。

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、dbmlsrv9 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。次に示すのは、dbmlsrv9 コマンド・ラインの一部です。

```
dbmlsrv9 ... -notifier "c:¥samples¥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」18 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)

エスケープ・シーケンス	説明
¥e	¥u001b：エスケープ（ESC）

Notifier

Notifier は、Mobile Link 同期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 同期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、SQL Anywhere インストール環境の *Samples¥MobiLink¥SIS_MultipleNotifier* サブディレクトリにあるサンプルを参照してください。

Notifier の起動

Notifier は、dbmlsrv9 コマンド・ラインで起動します。Notifier を起動するには、dbmlsrv9 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。

次に示すのは、dbmlsrv9 コマンド・ラインの一部です。

```
dbmlsrv9 ... -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」18 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」17 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、『Mobile Link 管理ガイド』>「-notifier オプション」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」78 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。

  注意 : request_cursor プロパティのみが必須プロパティです。詳細については、「request_cursor プロパティ」82 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」71 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」17 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
  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。UDP ゲートウェイと SMTP ゲートウェイを定義できます。さらに、デバイス・トラッカ・ゲートウェイを使用して、どの UDP または SMTP ゲートウェイを使用するかを自動的に決定できます。

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・トラッキングを使用する場合は、Mobile Link ユーザ名だけを指定します。あるゲートウェイで障害が発生した場合は、フェールオーバにより別のゲートウェイが使用されることがあります。

詳細については、「デバイス・トラッキング」27 ページを参照してください。

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。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。

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」17 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」87 ページ
- 「UDP ゲートウェイ・プロパティ」93 ページ
- 「SMTP ゲートウェイ・プロパティ」90 ページ

- 「Carrier プロパティ」96 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-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 2 つの従属ゲートウェイ (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。詳細については、「デバイス・トラッキング」27 ページを参照してください。

Default-UDP と Default-SMTP は、特に 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 ゲートウェイと共にデバイス・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 同期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 のプロパティのリストについては、「Carrier プロパティ」96 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、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 (割り当てられている場合は) UDP ゲートウェイを使用し、配信に失敗すると (割り当てられている場合は) SMTP ゲートウェイを使用します。

デバイス・トラッキングは、頻繁に変更される UDP アドレスで特に有用です。UDP アドレスが変更されると、Listener は最新のアドレスを統合データベースに自動的に送信します。

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。

詳細については、「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」31 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 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ため、この手順は省略してかまいません。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」24 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」82 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 クライアント』> 「Mobile Link ユーザの作成」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」29 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、および -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、Listener データベースをアドレス指定する代わりに、登録したユーザ名を直接アドレスとして指定できます。

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限り 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*** -t+ オプションで作成した 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** ***Listener_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 クライアント』> 「Mobile Link ユーザの作成」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 同期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 ユーティリティ」 50 ページを参照してください。

例

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

```
dblsn -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ほう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 ユーティリティ」50 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」32 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」34 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、および 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。

❖ 手動でデバイス・トラッキングを設定するには、次の手順に従います。

1 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',
     '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 Studio 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 の使用については、「ml_set_device」103 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」97 ページを参照してください)。3 番目のパラメータは、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」27 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」105 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。

詳細については、「ml_set_listening」107 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。
- SMS 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。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。

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。

詳細については、「ml_set_device_address」105 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・ラインの場合、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ときだけです。

```
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」27 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 ユーティリティ」 50 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」65 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 クライアント』> 「Adaptive Server Anywhere クライアントの同期パラメータ」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」 39 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」44 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。

オプション	説明
-x	Mobile Link 同期サーバへの接続方法を指定する。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、『Mobile Link クライアント』> 「-x オプション」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 同期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。

dblsn の詳細については、「Listener」49 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。

Listener の詳細については、「Listener」49 ページ を参照してください。

subject および 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。

Listener は、送信者名、件名、および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 によってチルダに変換されます。

Listener の詳細については、「Listener」49 ページ を参照してください。

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 と共にテストを実行し、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。詳細については、「action 変数」59 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。

たとえば、`sub, content` という形式で UDP ゲートウェイを介してメッセージを送信する場合、`[sub]content` という形で受信されます。しかし、これを SMTP ゲートウェイで送信する場合、`mySender@mySite.com \n sub \n content` となります。件名が `sub` であるメッセージを受信するには、次のフィルタを持つ 2 つのメッセージ・ハンドラが必要です。

```
-l "subject='sub';action=..."

-l "message_start='mySender@mySite.com ~ sub ~
';action=..."
```

Listener の詳細については、「Listener」49 ページ を参照してください。

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 同期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 同期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 同期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」49 ページ
- 「action 変数」59 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他にいくつかあります。詳細については、「Listener ユーティリティ」50 ページと「受信ライブラリ」61 ページを参照してください。

Listener の詳細については、「Listener」49 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・オプションをテキスト・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。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
```

パラメータ・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。

設定ファイルの詳細については、『ASA データベース管理ガイド』> 「設定ファイルの使用」を参照してください。

設定ファイル内のパスワード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。

『ASA データベース管理ガイド』> 「dbfhide コマンド・ライン・ユーティリティを使用してファイル内容を隠す」を参照してください。

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

この章の内容

この章は、Listener ユーティリティの詳細なリファレンスです。Listener ユーティリティは、Windows デバイス (Windows CE を含む) で実行されます。

Listener ユーティリティの使用法については、「Listener」37 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」65 ページ を参照してください。

Listener ユーティリティ

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」37 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」65 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」44 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」61 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」61 ページを参照してください。 カスタムの Listener ライブラリを作成することもできます。「Mobile Link Listener SDK」109 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`

dblsn オプション	説明
-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。
-g	-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。
-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。
-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 10 000 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、先にファイルをトランケートします。
-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。
-q	最小化ウィンドウで実行します。

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。 「デバイス・トラッキング用の Listener オプション」29 ページを参照してください。
-u *Listener_name*	Listener のユニークな名前を指定します。この名前は、トラッキング情報や配信確認のアップロードに使用されます。また、DeviceTracker ゲートウェイ用の通知アドレスとしても使用できます。 *Listener_name* は、Mobile Link ユーザ名です。他の Mobile Link ユーザ名と同様に、これはユニークでなければならず、統合データベースの ml_user Mobile Link システムに追加する必要があります。詳細については、『Mobile Link クライアント』> 「Mobile Link ユーザの作成」を参照してください。 デフォルトの Listener 名は、*device-name*-**dblsn** です。 「デバイス・トラッキング用の Listener オプション」29 ページ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 のメッセージと、基本的なアクションのトレース段階を表示する。 • **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。

dblsn オプション	説明
-w *password*	*Listener_name* 用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」29 ページを参照してください。
-x {**http**|**tcpip**} [**(***keyword=value;...***)**]	Mobile Link 同期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 管理ガイド』> 「-x オプション」を参照してください。この情報は、デバイス・トラッキング情報や配信確認を統合データベースに送信するために Listener で必要です。 「デバイス・トラッキング用の Listener オプション」29 ページを参照してください。
-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」29 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。

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 または **content** フィルタの使用の詳細については、「subject および content フィルタの使用」40 ページを参照してください。

message、**message_start**、**sender** フィルタの使用の詳細については、「フィルタ message、message_start、sender の使用」41 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、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 を起動します。

  ```
  "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・クラスにウィンドウ・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。

  ウィンドウ・メッセージは、メッセージの内容またはウィンドウ・メッセージの 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。

ウィンドウ・メッセージまたはウィンドウ・クラス名にスペースや句読点などの英数字以外の文字が含まれる場合は、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';"
```

詳細については、『Mobile Link クライアント』> 「-wc オプション」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」37 ページ

action 変数

次に示す 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、および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。
$sender	メッセージの送信側。

action 変数	説明
$type	この変数の意味は、carrier ライブラリに依存します。
$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。詳細については、「Push 要求」12 ページを参照してください。
$year	この変数の意味は、carrier ライブラリに依存します。
$month	この変数の意味は、carrier ライブラリに依存します。値は 1 ～ 12 までです。
$day	この変数の意味は、carrier ライブラリに依存します。値は 1 ～ 31 までです。
$hour	この変数の意味は、carrier ライブラリに依存します。値は 0 ～ 23 までです。
$minute	この変数の意味は、carrier ライブラリに依存します。値は 0 ～ 59 までです。
$second	この変数の意味は、carrier ライブラリに依存します。値は 0 ～ 59 までです。
$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。

action 変数	説明
$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

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。

カスタムの Listener ライブラリを作成することもできます。詳細については、「Mobile Link Listener SDK」109 ページ を参照してください。

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、このコード・ページ番号に基づいてマルチバイト文字を Unicode 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*	Carrier(*name*).network_provider_id を照合します。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、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 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、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。

第 4 章

Palm デバイス用 Listener

この章の内容

この章では、Palm デバイスでサーバ起動同期を設定し、実行する方法について説明します。Palm Listener は UDP をサポートしていません。

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 (lsnK7135.prc または lsnT600.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」37 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 の詳細については、「Palm Listener ユーティリティ」68 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」44 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 または **content** フィルタの使用については、「subject および content フィルタの使用」40 ページを参照してください。

message、**message_start**、**sender** フィルタの使用については、「フィルタ message、message_start、sender の使用」41 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。詳細については、「action 変数」67 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。

action 変数

次に示す 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。サポートされている Palm Listener は、Kyocera 7135 と Treo 600 の 2 つです。Listener ファイルは次のとおりです。

- **lsnK7135.prc** Kyocera 7135 用の Listener
- **lsnT600.prc** Treo 600 用の Listener

現在、この 2 種類の Palm Listener のみが設定ファイル *lsncfg.pdb* を読み込めます。

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。Kyocera と Treo のデバイスでは、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。詳細については、「Palm 用 Listener SDK」141 ページを参照してください。

第 5 章

Mobile Link 通知プロパティ

この章の内容

この章では、Notifier、ゲートウェイ、Carrier をカスタマイズするのに使用するプロパティについて説明します。

プロパティの設定方法については、「プロパティの設定」17 ページを参照してください。

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」17 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、および 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	トレースなし (デフォルト)
1	起動、停止、プロパティのトレース
2	通知メッセージを表示
3	ポーリング・レベルのトレース

参照

「プロパティの設定」17 ページ

例

次の例は、verbosity をレベル 2 に設定する方法です。

Sybase Central 内の Mobile Link プラグインを使用して verbosity プロパティを変更している場合、[通知] フォルダを右クリックして、[プロパティ] を選択します。

Notifier プロパティ・ファイルを使用してプロパティを設定する場合、次の行を挿入します。

```
verbosity=2
```

ストアド・プロシージャ ml_add_property を使用して verbosity レベルを変更する場合、次のように入力します。

```
ml_add_property( 'SIS','global','verbosity','2' );
```

Notifier プロパティ

以下のプロパティは、Notifier のプロパティ・ファイルで設定できます。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。

複数の Notifier を 1 つの Mobile Link サーバで実行することが可能です。追加の Notifier を設定するには、ある Notifier のプロパティをコピーし、別の Notifier 名とプロパティ値を指定します。

Notifier の詳細については、「Notifier」21 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」17 ページを参照してください。

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」17 ページ
- 「isolation プロパティ」80 ページ

例

Sybase Central 内の Mobile Link プラグインを使用している場合、Notifier ファイルを右クリックして、[プロパティ] を選択します。[論理] タブを開いて、ドロップダウン・リストから [begin_connection] を選択します。次のコードを [この SQL 文を実行] というボックスに貼り付けます。

```
set temporary option blocking = 'off'
```

Notifier プロパティ・ファイルを使用してプロパティを設定している場合、次の行で Car Dealer という Notifier に対して begin_connection が定義されます。円記号は、行が次の行に続くことを示す文字です。

```
Notifier(Car Dealer).begin_connection = ¥
    set temporary option blocking = 'off'
```

ストアド・プロシージャ ml_add_property を使用して begin_connection プロパティを変更する場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property(
   'SIS',
   'Notifier(Car Dealer)',
   'begin_connection',
   'set temporary option blocking = ''off'''
 );
```

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」17 ページ

例 たとえば、次の SQL 文は、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。

```
INSERT INTO PushRequest
 ( gateway, mluser, subject, content )
         SELECT 'MyGateway', DISTINCT mluser,
        'sync', stream_param
         FROM MLUserExtra, Dealer
         WHERE
```

```
     MLUserExtra.mluser.push_sync_status = "waiting for
request"
        AND Dealer.last_modified >
MLUserExtra.last_sync_time
```

Sybase Central 内の Mobile Link プラグインを使用している場合、Notifier ファイルを右クリックして、[プロパティ] を選択します。[論理] タブを開いて、ドロップダウン・リストから [begin_poll] を選択します。上記のコードを [この SQL 文を実行] というボックスに貼り付けます。

Notifier プロパティ・ファイルを使用してプロパティを設定する場合、NotifierA. という Notifier に対して次の行を挿入します。円記号は、行が次の行に続くことを示す文字です。

```
Notifier(NotifierA).begin_poll = ¥
 INSERT INTO PushRequest ¥
 ( gateway, mluser, subject, content ) ¥
         SELECT 'MyGateway', DISTINCT mluser, ¥
        'sync', stream_param ¥
         FROM MLUserExtra, Dealer ¥
         WHERE ¥
    MLUserExtra.mluser.push_sync_status = "waiting for
request" ¥
        AND Dealer.last_modified >
MLUserExtra.last_sync_time ¥
 );
```

ストアド・プロシージャ ml_add_property を使用して begin_connection プロパティを変更する場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property( 'SIS',
   'Notifier(Car Dealer)',
   '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'
 );
```

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 dbmlsrv9 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。

詳細については、『Mobile Link 管理ガイド』> 「ServerContext インタフェース」を参照してください。

参照

「プロパティの設定」17 ページ

例

Sybase Central 内の Mobile Link プラグインを使用している場合、Notifier ファイルを右クリックして、[プロパティ] を選択します。接続文字列が [接続] タブに設定されます。

Notifier プロパティ・ファイルを使用してプロパティを設定する場合、Simple という Notifier を設定して DSN を次の行で使用します。円記号は、行が次の行に続くことを示す文字です。

```
Notifier(Simple).connect_string = dsn=SIS_DB ¥
    ;uid=user;pwd=myPwd
```

ストアド・プロシージャ ml_add_property を使用して接続文字列を変更する場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property( 'SIS',
   'Notifier(Simple)',
   'connect_string',
   'dsn=SIS_DB;uid=user;pwd=myPwd' );
```

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 同期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 同期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」17 ページ

例

Sybase Central 内の Mobile Link プラグインを使用している場合、Notifier ファイルを右クリックして、[プロパティ] を選択します。[一般] タブを開いて、[実行時にコントロール・ウィンドウを表示] を変更します。

Notifier プロパティ・ファイルを使用してプロパティを設定している場合、次の行を使用して Simple という Notifier のダイアログを無効にします。

```
Notifier(Simple).gui=off
```

ストアド・プロシージャ ml_add_property を使用してこのプロパティを変更する場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property( 'SIS','Notifier(Simple)','gui','off'
);
```

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 同期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」17 ページ

例

Sybase Central 内の Mobile Link プラグインを使用している場合、[Notifiers] フォルダを開いて [Notifier を追加] をダブルクリックし、新しい Notifier を追加します。新しい Notifier が自動的に有効になります。既存の Notifier を無効にするには、Notifier ファイルを右クリックして [プロパティ] を選択し、[一般] タブを開いて [この Notifier を有効にする] のチェックをはずします。

Notifier プロパティ・ファイルを使用してプロパティを設定する場合、次の行を使用して NotifierA という Notifier を有効にします。

```
Notifier(NotifierA).enable=yes
```

ストアド・プロシージャ ml_add_property を使用して NotifierA を有効にする場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property(
'SIS','Notifier(NotifierA)','enable','yes' );
```

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。

例

Sybase Central 内の Mobile Link プラグインを使用している場合、Simple という Notifier ファイルを右クリックして、[プロパティ] を選択します。[論理] タブを開いて、ドロップダウン・リストから [end_connection] を選択します。次のコードを [この SQL 文を実行] というボックスに貼り付けます。

```
DELETE FROM NotifierShutdown WHERE name = 'Simple'
```

Notifier プロパティ・ファイルを使用してプロパティを設定している場合、次の行で Simple という Notifier に対して end_connection が定義されます。円記号は、行が次の行に続くことを示す文字です。

```
Notifier(Simple).end_connection =   ¥
    DELETE FROM NotifierShutdown WHERE name = 'Simple'
```

ストアド・プロシージャ ml_add_property を使用している場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property( 'SIS',
   'Notifier(Simple)',
   'end_connection',
   'DELETE FROM NotifierShutdown WHERE name =
''Simple''');
```

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。

例

Sybase Central 内の Mobile Link プラグインを使用している場合、Simple という Notifier ファイルを右クリックして、[プロパティ] を選択します。[論理] タブを開いて、ドロップダウン・リストから [end_poll] を選択します。次のコードを [この SQL 文を実行] というボックスに貼り付けます。

```
call reportAliveRequests( )
```

Notifier プロパティ・ファイルを使用してプロパティを設定している場合、次の行で Simple という Notifier に対して end_poll が定義されます。

```
Notifier(Simple).end_poll = call reportAliveRequests( )
```

ストアド・プロシージャ ml_add_property を使用している場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property( 'SIS',
   'Notifier(Simple)',
   'end_poll',
   'call reportAliveRequests( )');
```

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	コミットされる読み込み (デフォルト)
2	繰り返し可能読み出し
3	直列化可能

説明

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」17 ページ

例

Sybase Central 内の Mobile Link プラグインを使用している場合、Notifier ファイルを右クリックして、[プロパティ] を選択します。独立性レベルが [接続] タブに設定されます。

Notifier プロパティ・ファイルを使用してプロパティを設定する場合、次の行を使用して NotifierA という Notifier の独立性レベルを設定します。

```
Notifier(NotifierA).isolation=2
```

ストアド・プロシージャ ml_add_property を使用して独立性レベルを変更する場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property(
'SIS','Notifier(NotifierA)','isolation','2' );
```

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。

例

Sybase Central 内の Mobile Link プラグインを使用している場合、Notifier ファイルを右クリックして、[プロパティ] を選択します。[ポーリング] タブを開いて、ポーリング間隔を選択します。

Notifier プロパティ・ファイルを使用してプロパティを設定する場合、次の行を使用して Simple という Notifier を設定して 3 時間ごとにポーリングするようにします。

```
Notifier(Simple).poll_every = 3H
```

ストアド・プロシージャ ml_add_property を使用してポーリング間隔を変更する場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property(
'SIS','Notifier(Simple)','poll_every','3H' );
```

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」12 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。

例

Sybase Central 内の Mobile Link プラグインを使用している場合、Notifier ファイルを右クリックして、[プロパティ] を選択します。[論理] タブを開いて、ドロップダウン・リストから [request_cursor] を選択します。次のコードを [この SQL 文を実行] というボックスに貼り付けます。

```
SELECT req_id, gateway, subject, content, address
   FROM PushRequest
```

Notifier プロパティ・ファイルを使用してプロパティを設定している場合、次の行で Simple という Notifier に対して request_cursor が定義されます。円記号は、行が次の行に続くことを示す文字です。

```
Notifier(Simple).request_cursor = ¥
 SELECT req_id, gateway, subject, content, address ¥
   FROM PushRequest
```

ストアド・プロシージャ ml_add_property を使用している場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property(
   'SIS',
   'Notifier(Simple)',
   'request_cursor',
   'SELECT req_id, gateway, subject, content, address
      FROM PushRequest'
 );
```

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。パラメータのプレースホルダは、疑問符 (?) です。

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、SQL Anywhere Studio インストール環境の *Samples¥MobiLink¥SIS_CarDealer* サブディレクトリにある Car Dealer サンプルを参照してください。

例

たとえば、次の例では、Simple という Notifier は request_cursor から以前に取得した req_id に、疑問符 (?) を置き換えるように設定されます。

Sybase Central 内の Mobile Link プラグインを使用している場合、Simple という Notifier を右クリックして、[プロパティ] を選択します。[論理] タブを開いて、ドロップダウン・リストから [request_delete] を選択します。次のコードを [この SQL 文を実行] というボックスに貼り付けます。

```
DELETE FROM PushRequest WHERE req_id = ?
```

Notifier プロパティ・ファイルを使用してプロパティを設定している場合、次の行で Simple という Notifier に対して request_delete が定義されます。円記号は、行が次の行に続くことを示す文字です。

```
Notifier(Simple).request_delete = ¥
    DELETE FROM PushRequest WHERE req_id = ?
```

ストアド・プロシージャ ml_add_property を使用している場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property( 'SIS',
   'Notifier(Simple)',
   'request_delete',
   'delete = DELETE FROM PushRequest WHERE req_id = ?');
```

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。

例

Sybase Central 内の Mobile Link プラグインを使用している場合、Notifier ファイルを右クリックして、[プロパティ] を選択します。[論理] タブを開いて、ドロップダウン・リストから [shutdown_query] を選択します。次のコードを [この SQL 文を実行] というボックスに貼り付けます。

```
SELECT COUNT(*) FROM NotifierShutdown
   WHERE name='Simple'
```

Notifier プロパティ・ファイルを使用してプロパティを設定している場合、次の行で Simple という Notifier に対して shutdown_query が定義されます。円記号は、行が次の行に続くことを示す文字です。

```
Notifier(Simple).shutdown_query = ¥
   SELECT COUNT(*) FROM NotifierShutdown ¥
   WHERE name='Simple'
```

ストアド・プロシージャ ml_add_property を使用している場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property(
'SIS','Notifier(Simple)','shutdown_query',
  'SELECT COUNT(*) FROM NotifierShutdown
   WHERE name=''Simple'''

 );
```

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 の結果セットにある 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、および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。

デバイス・トラッキングの詳細については、「デバイス・トラッキング」27 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」17 ページを参照してください。

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。

次に例を示します。

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

例

Sybase Central 内の Mobile Link プラグインを使用している場合、[ゲートウェイ] フォルダを開いて [Default-DeviceTracker] を右クリックし、[プロパティ] を選択します。[配信] タブを開いて、[メッセージ配信の確認] を選択します。

Notifier プロパティ・ファイルを使用してプロパティを設定する場合、次の行を使用して Default-DeviceTracker というデバイス・トラッカ・ゲートウェイが配信確認を行うようにします。

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

ストアド・プロシージャ ml_add_property を使用して Default-DeviceTracker に confirm_delivery を設定する場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property( 'SIS',
   'DeviceTracker(Default-DeviceTracker)',
   'confirm_delivery',
   'yes' );
```

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。

例

Sybase Central 内の Mobile Link プラグインを使用している場合、[ゲートウェイ] フォルダを開いて、[ゲートウェイを追加] をダブルクリックします。新しいゲートウェイが自動的に有効になります。

Notifier プロパティ・ファイルを使用してプロパティを設定する場合、次の行を使用して Default-DeviceTracker というデバイス・トラッカ・ゲートウェイを有効にします。

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

ストアド・プロシージャ ml_add_property を使用して Default-DeviceTracker を有効にする場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property( 'SIS',
   'DeviceTracker(Default-DeviceTracker)',
   'enable',
   'yes' );
```

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。

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。

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 リスナに送信することも可能です。

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」24 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」17 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)。

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。

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。

例

Sybase Central 内の Mobile Link プラグインを使用している場合、[ゲートウェイ] フォルダを開いて、[ゲートウェイを追加] をダブルクリックします。新しいゲートウェイが自動的に有効になります。

Notifier プロパティ・ファイルを使用してプロパティを設定する場合、次の行を使用して Gate3 という SMTP ゲートウェイを有効にします。

```
SMTP(Gate3).enable=yes
```

ストアド・プロシージャ ml_add_property を使用してゲートウェイ Gate3 を有効にする場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property( 'SIS','SMTP(Gate3)','enable','yes' );
```

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。

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。

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。

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。

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。

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 カラムから取り出されます。

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」24 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」17 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)。

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。

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。

例

Sybase Central 内の Mobile Link プラグインを使用している場合、[ゲートウェイ] フォルダを開いて、[ゲートウェイを追加] をダブルクリックします。新しいゲートウェイが自動的に有効になります。

Notifier プロパティ・ファイルを使用してプロパティを設定する場合、次の行を使用して Gate3 という UDP ゲートウェイを有効にします。

```
UDP(Gate3).enable=yes
```

ストアド・プロシージャ ml_add_property を使用してゲートウェイ Gate3 を有効にする場合、次のように入力します。

```
ml_add_property( 'SIS','UDP(Gate3)','enable','yes' );
```

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。

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。

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。

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。

Carrier プロパティ

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」97 ページ
- 「sms_email_user_prefix プロパティ」98 ページ
- 『Mobile Link 管理ガイド』> 「ml_device_address」

Carrier の詳細については、「ゲートウェイと Carrier」24 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」17 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。

例

Sybase Central 内の Mobile Link プラグインを使用している場合、[Carrier] フォルダを開いて、[Carrier マッピングの追加] をダブルクリックします。新しい Carrier マッピングが自動的に有効になります。

Notifier プロパティ・ファイルを使用してプロパティを設定する場合、次の行を使用して Bell Mobility 1x という Carrier マッピングを有効にします。

```
Carrier(Bell Mobility 1x).enable=yes
```

ストアド・プロシージャ ml_add_property を使用して Carrier Bell Mobility 1x を有効にする場合、次のように入力します (Adaptive Server Anywhere 統合データベースを想定します)。

```
ml_add_property( 'SIS',
   'Carrier(Bell Mobility 1x)',
   'enable',
   'yes' );
```

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_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」98 ページ
- 『Mobile Link 管理ガイド』> 「ml_device_address」

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」97 ページ
- 『Mobile Link 管理ガイド』> 「ml_device_address」

第 6 章

サーバ起動同期のストアド・プロシージャ

この章の内容

この章では、サーバ起動同期で使用されるストアド・プロシージャについて説明します。これらのストアド・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：これらのストアド・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、これらのストアド・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、これらのストアド・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」27 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」31 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 管理ガイド』>「Mobile Link システム・テーブル」を参照してください。

その他の Mobile Link ストアド・プロシージャの詳細については、『Mobile Link 管理ガイド』>「ストアド・プロシージャ」を参照してください。

ml_delete_device

機能

手動でデバイス・トラッキングを設定している場合、このストアド・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

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

機能

手動でデバイス・トラッキングを設定している場合、このストアド・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

詳細については、「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」31 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

機能

手動でデバイス・トラッキングを設定している場合、このストアド・プロシージャを使用して、Moblie 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

詳細については、「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」31 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

機能

手動でデバイス・トラッキングを設定している場合、このストアド・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

ストアド・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。

詳細については、「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」31 ページを参照してください。

参照

- 「ml_set_device_address」105 ページ

- 「ml_set_listening」107 ページ
- 『Mobile Link 管理ガイド』>「ml_device」
- 『Mobile Link 管理ガイド』>「ml_device_address」
- 『Mobile Link 管理ガイド』>「ml_listening」

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

機能

手動でデバイス・トラッキングを設定している場合、このストアド・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

ストアド・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。

詳細については、「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」31 ページを参照してください。

参照

- 「ml_set_device」103 ページ

- 「ml_set_listening」107 ページ
- 『Mobile Link 管理ガイド』>「ml_device」
- 『Mobile Link 管理ガイド』>「ml_device_address」
- 『Mobile Link 管理ガイド』>「ml_listening」

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

機能

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

パラメータ

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

説明

ストアド・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。

詳細については、「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」31 ページを参照してください。

参照

- 「ml_set_device」103 ページ
- 「ml_set_device_address」105 ページ
- 『Mobile Link 管理ガイド』>「ml_device」
- 『Mobile Link 管理ガイド』>「ml_device_address」
- 『Mobile Link 管理ガイド』>「ml_listening」

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

Mobile Link Listener SDK

この章の内容

この章では、Listener ソフトウェア開発キットについて説明します。これは、未サポートのリモート・デバイスで Listener をサポートできるように支援するものです。

概要

Mobile Link のサーバからの同期を現在サポートしていないリモート・デバイスを使用する場合は、Listener ソフトウェア開発キットを使用して、そのデバイス用の 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プログラム API です。

たとえば、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。この SDK は、Windows (32 ビットと CE) と Palm の両方のオペレーティング・システムに対応した開発キットです。

Palm 用の Listener SDK の詳細については、「Palm 用 Listener SDK」141 ページ を参照してください。

Windows および Windows CE 用の Listener SDK の詳細については、「Windows 用 Listener SDK」112 ページ を参照してください。

Mobile Link Listener SDK と実装サンプルは、次のファイルにあります。いずれのファイルも、SQL Anywhere Studio インストール先の *MobiLink¥ListenerSDK* サブディレクトリにあります。

Windows 用 Listener SDK ファイル

Windows 用ファイル	説明
¥Win32andCE¥Win32_VC¥lsn.def	Listener ライブラリの Visual C++ モジュール定義
¥Win32andCE¥CE_EVC¥lsn.def	Listener ライブラリの Visual C モジュール定義
¥Win32andCE¥src¥lsn.h	Win32 と CE の Listener ライブラリ API
¥Win32andCE¥src¥swi510.c	Sierra Wireless AirCard 510 の実装
Win32andCE¥src¥udp.c	UDP の実装

Palm 用 Listener SDK ファイル

Palm 用ファイル	説明
¥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。
¥Palm¥68k¥cw¥rsc¥	Palm Listener の UI リソースを含む。
¥Palm¥src¥PalmLsn.h	ランタイム・ライブラリのヘッダと Palm Listener API
¥Palm¥src¥Kyocera7135.c	Kyocera 7135 の実装
¥Palm¥src¥Treo600.c	Treo 600 の実装

Windows 用 Listener SDK

Windows (32 ビット版および CE) 用 Listener SDK を使用すると、新しい無線ネットワーク・アダプタや新しい通信プロトコル用の Listener 共有ライブラリを作成できます。

このライブラリは、サーバ起動同期メッセージを取り出すための共通インタフェースに準拠しており、次の要素で構成されています。

- Listener メッセージ・データを表すクラス。
- 戻り値と Listener SDK のバージョン番号を定義する列挙。
- Listener の状態を格納する構造体。
- Listener のメッセージまたは状態情報を管理する関数。

Listener ライブラリの詳細については、「受信ライブラリ」61 ページを参照してください。

Listener ユーティリティ (dblsn.exe) に -d コマンド・ライン・オプションを付けて起動すると、ライブラリがロードされます。

dblsn オプションの詳細については、「Listener ユーティリティ」50 ページを参照してください。

データ型

Windows 用 Listener SDK には、次のデータ型が定義されています。これらのデータ型は、よく使われる値の型を、SDK インタフェースで使用する各 Listener 型にマッピングしたものです。

名前	説明
lsn_bool	unsigned char
lsn_ulong	unsigned long
lsn_long	long

メッセージ・クラス

メッセージ・クラス (a_msg) は、Listener メッセージを管理するための以下のインタフェースを提供します。

- メッセージ・データを格納するための public フィールド。
- メッセージの割り当て、比較、基本操作を行う各種関数。

a_msg public フィールド

機能

メッセージ・クラスには、送信元、メッセージの内容、およびメッセージの送信時刻、タイプ、優先度 (この 3 つはオプション) を表すためのフィールドが含まれています。

プロトタイプ

```
class a_msg {

 // message class functions...
 ...

 // message class fields

    public:
     lsn_long year;
     lsn_long month;      // 1 to 12
     lsn_long day;     // 1 to 31
     lsn_long hour;     // 0 to 23
     lsn_long minute;     // 0 to 59
     lsn_long second;     // 0 to 59
     lsn_long type;
     lsn_long priority;
     TCHAR * sender;
     TCHAR * message;
 ...

 };
```

パラメータ

名前	説明
sender	メッセージの送信元を表す文字列。

名前	説明
message	メッセージの内容を表す文字列。
year	メッセージの年フィールドを表す整数。
month	メッセージの月フィールドを表す整数。
day	メッセージの日フィールドを表す整数。
hour	メッセージの時フィールドを表す整数。
minute	メッセージの分フィールドを表す整数。
second	メッセージの秒フィールドを表す整数。
type	メッセージのタイプを表す整数。
priority	メッセージの優先度を表す整数。

説明

実装する際に、上記の各フィールドに意味を与えます。

時刻、タイプ、優先度の各フィールドを有効にするかどうかを指定するには、lsn_info 構造体を使用します。たとえば、lsn_info の isMsgPriorityApplicable フィールドに FALSE を設定すると、a_msg の priority フィールドを無効にできます。

lsn_info 構造体の詳細については、「lsn_info 構造体」120 ページを参照してください。

例

次の例では、a_msg クラスのインスタンスを宣言し、priority フィールドに 1 が設定されているかどうかを判定しています。

```
a_msg * msgA;

 //...

 if( msgA -> priority == 1)
 {
         MessageBox( NULL,
       TEXT("Message Priority is 1."),
          TEXT( "Message from lsn_udp.dll" ),
MB_ICONEXCLAMATION );
 }
```

a_msg allocateSize 関数

機能

指定されたサイズの sender フィールドと message フィールドを持つ新しい a_msg インスタンスを返します。

プロトタイプ

static a_msg * **allocateSize (**
long *senderTChars*,
long *messageTChars*
)

パラメータ

- **senderTChars** sender フィールドの文字数。
- **messageTChars** message フィールドの文字数。

戻り値

すべてのフィールドがゼロに初期化されたメッセージ・クラスの新しいインスタンス。

参照

- 「メッセージ・クラス」113 ページ

例

次の例では、allocateSize 関数を使用して新しい a_msg インスタンスを作成しています。この実装では、sender フィールドは 16 文字、message フィールドは 200 文字です。

```
#define MAX_SENDER_TCHARS        16
 #define MAX_MESSAGE_TCHARS        200

 //...

 a_msg* msgA;
 msgA = a_msg::allocateSize( MAX_SENDER_TCHARS,
MAX_MESSAGE_TCHARS );
```

a_msg copy 関数

機能

a_msg パラメータを使用して呼び出し側インスタンスのフィールドを上書きします。

プロトタイプ

void **copy (** a_msg * *source-msg* **)**

パラメータ

- **source-msg** 呼び出し側のインスタンスに含まれる情報を置換するために使用される a_msg インスタンス。

参照

- 「メッセージ・クラス」113 ページ

例

次の例では、copy 関数を使用して msgA を初期化しています。この場合のコピー元は、lsn_state のインスタンス stateB の copyOfLastReceivedMsg フィールドです。

lsn_state 構造体の詳細については、「lsn_state 構造体」122 ページを参照してください。

```
a_msg* msgA;
 msgA -> copy( stateB -> copyOfLastReceivedMsg );
```

a_msg equals 関数

機能

呼び出し側インスタンスの各 public フィールドが、a_msg パラメータ内の各フィールドと等しいかどうかを判定します。

プロトタイプ

bool **equals (** a_msg * *source-msg* **)**

パラメータ

- **source-msg** 呼び出し側のインスタンスと比較するために使用される a_msg インスタンス。

戻り値

呼び出し側インスタンスの各 public フィールドが、a_msg パラメータ内の各フィールドと等しい場合は TRUE、それ以外の場合は FALSE。

参照

- 「メッセージ・クラス」113 ページ

例

次の例では、equals 関数を使用して、msgA が、lsn_state のインスタンス stateB の copyOfLastReceivedMsg フィールドと等しいかどうかを判定しています。

lsn_state 構造体の詳細については、「lsn_state 構造体」122 ページを参照してください。

```
a_msg* msgA;

 //...

 if(msgA -> equals( stateB -> copyOfLastReceivedMsg ))
 {
         MessageBox( NULL,
```

```
        TEXT("msgA equals stateB-
>copyOfLastReceivedMsg."),
          TEXT( "Message from lsn_sw786.dll" ),
MB_ICONEXCLAMATION );
 }
```

a_msg makeEmpty 関数

機能　a_msg インスタンスのフィールドをゼロ・クリアします。

プロトタイプ　void **makeEmpty ( )**

参照

- 「メッセージ・クラス」113 ページ

例　次の例では、makeEmpty 関数を使用して a_msg インスタンスの内容をクリアしています。

```
a_msg* msgA;
 msgA -> makeEmpty();
```

a_msg reallocBuffers 関数

機能　a_msg インスタンスの sender および message バッファのサイズを増やします。

プロトタイプ　unsigned char **reallocBuffers (**
long *senderTChars*,
long *messageTChars*
)

パラメータ

- **senderTChars** sender フィールドの文字数。
- **messageTChars** message フィールドの文字数。

戻り値　sender および message フィールドが有効か NULL 以外の場合は TRUE、それ以外の場合は FALSE。

説明　入力引数に指定された値が、sender または message フィールドの現在のサイズよりも小さい場合、フィールド・サイズは変更されません。

参照

- 「メッセージ・クラス」113 ページ

例

次の例では、reallocBuffers 関数を使用して a_msg インスタンスの sender および message フィールドのサイズを増やしています。フィールドの再割り当てに失敗すると、リターン・コード LSN_RET_OUT_OF_MEMORY が返されます。

LSN_RET_OUT_OF_MEMORY の詳細については、「LSN_RET 列挙」118 ページを参照してください。

```
#define MAX_SENDER_TCHARS        16
 #define MAX_MESSAGE_TCHARS        200
 //...

 a_msg* msgA;
 //...

 if( msgA.reallocBuffers(MAX_SENDER_TCHARS,
MAX_MESSAGE_TCHARS))
 {
     ret = LSN_RET_OUT_OF_MEMORY;
     goto failed;
 }
```

LSN_RET 列挙

機能

Windows Listener SDK の各関数の戻り値のセットを定義します。

プロトタイプ

```
typedef enum {
    LSN_RET_OK = 0,
    LSN_RET_NOT_SUP = 1,
    LSN_RET_NO_RESP = 2,
    LSN_RET_FAILED = 3,
    LSN_RET_MSG_NOT_READ = 4,
    LSN_RET_NO_MORE_MSG = 5,
    LSN_RET_OUT_OF_MEMORY = 6,
    LSN_RET_BAD_ARG = 7,
    LSN_RET_NOT_ENOUGH_ARG = 8,
    LSN_RET_TOO_MANY_ARGS = 9,
    LSN_RET_SERVICE_NOT_ACTIVATED = 10
} LSN_RET;
```

パラメータ

値	説明
LSN_RET_OK	関数呼び出しが正常終了した。
LSN_RET_NOT_SUP	関数呼び出しが無効か適用できない。たとえば、UDP Listener の実装に、LsnSuspendListening 関数を使用することはできません。詳細については、「LsnSuspendListening 関数」131 ページを参照してください。
LSN_RET_NO_RESP	要求のタイムアウトが発生した。
LSN_RET_FAILED	一般的な失敗。
LSN_RET_MSG_NOT_READ	Listener 記憶領域内に未読メッセージがある。新しいメッセージの受信要求は無視されました。
LSN_RET_NO_MORE_MSG	未読メッセージがない。メッセージの読み取りに失敗しました。
LSN_RET_OUT_OF_MEMORY	メモリ領域不足のため関数の実行を継続できない。
LSN_RET_BAD_ARG	1 つ以上の入力パラメータで問題が発生した。
LSN_RET_NOT_ENOUGH_ARG	入力引数が不足している。この戻り値は、入力配列に含まれる文字列引数の数が不足していることを示します。
LSN_RET_TOO_MANY_ARGS	入力引数が多過ぎる。この戻り値は、入力配列に含まれる文字列引数の数が多過ぎることを示します。

LSN_VERSION 列挙

機能　Listener SDK のバージョン番号を定義します。

プロトタイプ

```
typedef enum {
        LSN_VERSION_1 = 1,
        LSN_VERSION_2 = 2,
        LSN_CURRENT_VERSION = LSN_VERSION_2
    } LSN_VERSION;
```

パラメータ

値	説明
LSN_VERSION_1	Listener SDK バージョン 1 を指定する。
LSN_VERSION_2	Listener SDK バージョン 2 を指定する。
LSN_CURRENT_VERSION	Listener の現在のバージョンを指定する。

説明

この列挙は、アプリケーションの開発に使用した Listener SDK のバージョンを指定するために使用します。現在のバージョンを指定するには、LSN_CURRENT_VERSION を使用します。

lsn_info 構造体

機能

Listener の静的な情報を格納します。

プロトタイプ

```
typedef struct lsn_info {
        lsn_ulong version;
        lsn_ulong maxSenderTChars;
        lsn_ulong maxMessageTChars;
        lsn_bool isExtDialNeedSuspendListening;
        lsn_bool isMsgTimeApplicable;
        lsn_bool isMsgTypeApplicable;
        lsn_bool isMsgPriorityApplicable;
        lsn_bool isMsgTimeInUTC;

        // LSN_VERSION_1 fields ends here

    } lsn_info;
```

パラメータ

値	説明
version	Listener インタフェースまたは SDK のバージョン。
maxSenderTChars	sender フィールドの最大文字数。
maxMessageTChars	message フィールドの最大文字数。
isExtDialNeedSuspendListening	別のアプリケーションがモデムを使用してダイヤルする前に Listener ライブラリがメッセージの受信を保留する必要がある場合は、このフィールドを TRUE に設定する。
isMsgTimeApplicable	メッセージ・クラスの時刻フィールド (year、month、day、hour、minute、second) が有効な場合は、このフィールドを TRUE に設定する。詳細については、「a_msg public フィールド」113 ページを参照してください。
isMsgTypeApplicable	メッセージ・クラスの type フィールドが有効な場合は、このフィールドを TRUE に設定する。詳細については、「a_msg public フィールド」113 ページを参照してください。
isMsgPriorityApplicable	メッセージ・クラスの priority フィールドが有効な場合は、このフィールドを TRUE に設定する。詳細については、「a_msg public フィールド」113 ページを参照してください。
isMsgTimeInUTC	メッセージ・クラスの各時刻フィールドに UTC (協定世界時) が使用されている場合は、このフィールドを TRUE に設定する。詳細については、「a_msg public フィールド」113 ページを参照してください。

参照

- 「メッセージ・クラス」113 ページ

例

次の例では、UDP Listener 用に lsn_info 構造体の各フィールドを初期化しています。

```
lsn_info * info;

 info->version = LSN_VERSION_2;
 info->maxSenderTChars = MAX_SENDER_TCHARS;
 info->maxMessageTChars = MAX_MESSAGE_TCHARS;
 info->isExtDialNeedSuspendListening = false;
 info->isMsgTimeApplicable = false;
 info->isMsgTypeApplicable = false;
 info->isMsgPriorityApplicable = false;
 info->isMsgTimeInUTC = false;
```

lsn_state 構造体

機能

動的情報として使用される Listener 状態構造体。

プロトタイプ

```
struct lsn_state;
 // Add your Listener state fields here
```

説明

この構造体にフィールドを追加することで、メッセージ・フィールド、受信バッファ、未読メッセージを指定するフラグといった動的な Listener 情報を格納します。

例

次の例は、UDP Listener の実装に使用する lsn_state 構造体です。

```
typedef struct lsn_state {
     WSADATA wsaData;
     SOCKET socket;
     bool hideWSAErrorBox;
     bool unread;
     bool showSenderPort;
     a_msg * msg;
     struct timeval timeout;
     char * recvBuff;
     UINT codePage;
     TCHAR port[6];
 } lsn_state;
```

次の例は、Sierra Wireless AirCard 510 Listener の実装に使用する lsn_state 構造体です。

```
typedef struct lsn_state {
     bool apiOpened;
     int nextMsg;
     int numReceivedMsg;
     int messageStoreSize;
     a_msg copyOfLastReceivedMsg;
     a_msg ** messages;
     TCHAR * phone;
     TCHAR * carrier;
 } lsn_state;
```

LsnInit 関数

機能

Listener 状態構造体の割り当てと初期化を実行します。

プロトタイプ

LSN_RET **LsnInit (**
lsn_info* *info*,
lsn_state** *state*,
lsn_long *argc*,
char* *argv[]*,
lsn_long* *badArg*
)

パラメータ

- **info** 静的な Listener 情報を保持する lsn_info 構造体が格納された出力パラメータ。

  詳細については、「lsn_info 構造体」120 ページを参照してください。

- **state** 割り当てられた lsn_state 構造体を指す出力パラメータ。

  詳細については、「lsn_state 構造体」122 ページを参照してください。

- **argc** argv[] 配列内の文字列引数の数。

- **argv[]** Listener 状態構造体を初期化するために使用される文字列引数配列。

argv[] に渡される引数は、dblsn.exe -a コマンド・ライン・オプションを使用して指定します。次に例を示します。

```
dblsn -d my_lsn.dll -a prop1=v1 -a prop2=v2 ...
```

上記のコマンド・ライン・オプションを指定すると、argv[] の中身は次のようになります。

```
argv[] = { "prop1=v1", "prop2=v2", ... }
```

-a オプションの詳細については、「Listener ユーティリティ」50 ページを参照してください。

- **badArg** 引数エラー情報を提供する出力パラメータ。このパラメータの意味は、次のように LsnInit の戻り値によって異なります。
  - 戻り値が LSN_RET_BAD_ARG の場合、badArg には argv の中の問題のある引数のインデックスが返される。
  - 戻り値が LSN_RET_NOT_ENOUGH_ARG の場合、badArg には最低限必要な引数の数が返される。
  - 戻り値が LSN_TOO_MANY_ARGS の場合、badArg には argv 内に格納できる引数の最大個数が返される。

戻り値

この関数では、LSN_RET 列挙に定義された次の戻り値を使用します。

値	説明
LSN_RET_OK	関数呼び出しが正常終了した。
LSN_RET_FAILED	一般的な失敗。
LSN_RET_OUT_OF_MEMORY	メモリ領域不足のため関数の実行を継続できない。
LSN_RET_BAD_ARG	1 つ以上の入力パラメータで問題が発生した。

値	説明
LSN_RET_NOT_ENOUGH_ARG	入力引数が不足している。この戻り値は、入力配列に含まれる文字列引数の数が不足していることを示します。
LSN_RET_TOO_MANY_ARGS	入力引数が多過ぎる。この戻り値は、入力配列に含まれる文字列引数の数が多過ぎることを示します。

LSN_RET 列挙の詳細については、「LSN_RET 列挙」118 ページを参照してください。

説明

この関数は Listener を初期化し、Listener が動作するために必要なリソースを確保します。この関数が正常終了すると、lsn_info 構造体に必要な情報が設定され、lsn_state 構造体が割り当てられます。

argv[] 配列に格納される入力パラメータの数とタイプは実装によって異なります。

参照

- 「LSN_RET 列挙」118 ページ
- 「lsn_state 構造体」122 ページ
- 「lsn_info 構造体」120 ページ
- 「メッセージ・クラス」113 ページ

例

次の例は、UDP Listener の実装に使用する LsnInit 関数です。

```
LsnExport LSN_RET LsnExportFunc LsnInit(
     lsn_info * info, lsn_state** state, lsn_long argc,
     char* argv[], lsn_long* badArg )
 /
*************************************************/
 {
     // Argument names
     #define ARGN_RECEIVER_PORT "Port"
     #define ARGN_HIDE_WSA_ERROR_BOX "HideWSAErrorBox"
     #define ARGN_TIMEOUT_SEC "Timeout"
     #define ARGN_SHOW_SENDER_PORT "ShowSenderPort"
     #define ARGN_CODEPAGE "CodePage"

   // Helper macros
```

```
    #define IS_ARG( argn, x ) _strnicmp( argn "=", x,
sizeof(argn) ) == 0
    #define ARG_VAL( argn, x ) &(x)[sizeof(argn)]


  LSN_RET ret = LSN_RET_FAILED;
    lsn_state * s;
    struct sockaddr_in receiverAddr;
    unsigned short port;
    int i;
    bool showUsageBox = true;
    *state = NULL;

        // Fill static info structure
    info->version = LSN_VERSION_2;
    info->maxSenderTChars = MAX_SENDER_TCHARS;
    info->maxMessageTChars = MAX_MESSAGE_TCHARS;
    info->isExtDialNeedSuspendListening = false;
    info->isMsgTimeApplicable = false;
    info->isMsgTypeApplicable = false;
    info->isMsgPriorityApplicable = false;
    info->isMsgTimeInUTC = false;


// Check argc
    if( argc < 0 ) {
    *badArg = 0;
    ret = LSN_RET_NOT_ENOUGH_ARG;
    goto failed;
    }
    if( argc > 4 ) {
    *badArg = 4;
    ret = LSN_RET_TOO_MANY_ARGS;
    goto failed;
    }
```

```
 // Allocate state structure
     s = (lsn_state*)calloc( 1, sizeof(lsn_state) );
     if( s == NULL ) {
     showUsageBox = false;        ret =
LSN_RET_OUT_OF_MEMORY;
     goto failed;
     }


 // Initialize state structure
     memset( s, 0, sizeof(lsn_state) );
     s->socket = INVALID_SOCKET;
     s->hideWSAErrorBox = false;
     s->unread = false;
     s->showSenderPort = false;
     s->timeout.tv_sec = 0;
     s->timeout.tv_usec = 0;


// Allocate message in state structure
     s->msg = a_msg::allocateSize( MAX_SENDER_TCHARS,
MAX_MESSAGE_TCHARS );
     if( s->msg == NULL ) {
     showUsageBox = false;        ret =
LSN_RET_OUT_OF_MEMORY;
     goto failed;
     }
 #if defined( UNICODE )
     // Allocate recvBuff
    s->recvBuff = (char*)malloc( MAX_MESSAGE_BYTES + 1
);
     if( s->recvBuff == NULL ) {
     ret = LSN_RET_OUT_OF_MEMORY;
     goto failed;
     }
 #endif
 port = 5001;


 // Read args
     for( i = 0; i < argc; i++ ) {
    *badArg = i;       if( IS_ARG( ARGN_RECEIVER_PORT,
argv[i] ) ) {
         port = atoi( ARG_VAL( ARGN_RECEIVER_PORT,
```

```
argv[i] ) );
     } else if( _stricmp( ARGN_HIDE_WSA_ERROR_BOX,
argv[i] ) == 0 ) {
          s->hideWSAErrorBox = true;
    } else if( _stricmp( ARGN_SHOW_SENDER_PORT, argv[i]
) == 0 ) {
          s->showSenderPort = true;
     } else if( IS_ARG( ARGN_TIMEOUT_SEC, argv[i] ) ) {
          s->timeout.tv_sec = atol( ARG_VAL(
ARGN_TIMEOUT_SEC, argv[i] ) );
     } else if( IS_ARG( ARGN_CODEPAGE, argv[i] ) ) {
          s->codePage = atol( ARG_VAL( ARGN_CODEPAGE,
argv[i] ) );
     } else {
          ret = LSN_RET_BAD_ARG;
          goto failed;
     }
     }



 // Fill the port buffer
     _stprintf( s->port, TEXT("%d"), port );
     // At this point, all args are valid
     showUsageBox = false;
     *badArg = -1;
     // WSAStartup
     if( WSAStartup( 0x202, &s->wsaData ) != 0 ) {
     foundWSAError( s );
     goto failed;
     }



// Setup receiver address
     memset( &receiverAddr, 0, sizeof(receiverAddr) );
     receiverAddr.sin_family = AF_INET;
     receiverAddr.sin_addr.s_addr = INADDR_ANY;
     receiverAddr.sin_port = htons( port );
     // Open socket
     s->socket = socket( AF_INET, SOCK_DGRAM, 0 );
     if( s->socket == INVALID_SOCKET ) {
```

```
        foundWSAError( s );
        goto failed;
        }


    // Bind the socket to the receiver address
        if( bind( s->socket, (struct
sockaddr*)&receiverAddr, sizeof( receiverAddr ) )
            == SOCKET_ERROR ) {
        foundWSAError( s );
        goto failed;
        }
        *state = s;
        return( LSN_RET_OK );
      failed:
        if( showUsageBox ) {
        MessageBox( NULL, USAGE, USAGE_TITLE,
MB_ICONINFORMATION );
        }
        LsnFini( s );
        return( ret );
    }
```

LsnFini 関数

機能　Listener を停止し、リソースを解放します。

プロトタイプ　void **LsnFini (** lsn_state* *state* **)**

パラメータ

- **state** lsn_state 構造体インスタンスへのポインタ。

  詳細については、「lsn_state 構造体」122 ページを参照してください。

説明　この関数は Listener を停止し、Listener に割り当てられていた Listener 状態構造体などのリソースを解放します。

参照

- 「lsn_state 構造体」122 ページ
- 「メッセージ・クラス」113 ページ

例

次の例は、UDP Listener の実装に使用する LsnFini 関数です。

```
LsnExport void LsnExportFunc LsnFini( lsn_state* state
)
 /
**************************************************
/
 {

  if( state != NULL ) {
      if( state->msg != NULL ) {
          delete( state->msg );
      }
      if( state->recvBuff != NULL ) {
          free( state->recvBuff );
      }
      if( state->socket != INVALID_SOCKET ) {
              closesocket( state->socket );
      }
      WSACleanup();
      free( state );
  }
 }
```

LsnIsListening

機能

Listener がメッセージの受信準備を完了したかどうかを判定します。

プロトタイプ

bool **LsnIsListening (** const lsn_state* *state* **)**

パラメータ

- **state** lsn_state 構造体インスタンスへのポインタ。

  lsn_state の詳細については、「lsn_state 構造体」122 ページを参照してください。

戻り値

Listener がメッセージを受信できる状態であれば TRUE を、それ以外の場合は FALSE を返します。

説明

この関数は、Listener がメッセージの受信準備を完了したかどうかを定義します。

参照

- 「lsn_state 構造体」122 ページ

- 「メッセージ・クラス」113 ページ

例　次の例は、UDP Listener 用の LsnIsListening 関数の実装です。

```
LsnExport bool LsnExportFunc LsnIsListening(
     const lsn_state* state)
/*******************************************/
{
     return( state->socket != INVALID_SOCKET );
}
```

LsnSuspendListening 関数

機能　Listener を中断します。

プロトタイプ　LSN_RET **LsnSuspendListening (** lsn_state* *state* **)**

パラメータ

- **state** Listener 状態構造体インスタンスへのポインタ。

  lsn_state の詳細については、「lsn_state 構造体」122 ページを参照してください。

戻り値　この関数では、LSN_RET 列挙に定義された次の戻り値を使用します。

値	説明
LSN_RET_OK	関数呼び出しが正常終了した。
LSN_RET_NOT_SUP	関数呼び出しが無効か適用できない。たとえば、UDP Listener の実装に、LsnSuspendListening 関数を使用することはできません。詳細については、「LsnSuspendListening 関数」131 ページを参照してください。
LSN_RET_NO_RESP	要求のタイムアウトが発生した。
LSN_RET_FAILED	一般的な失敗。

LSN_RET 列挙の詳細については、「LSN_RET 列挙」118 ページを参照してください。

説明

この関数は、Listener によるメッセージの受信を中断する際に使用します。

参照

- 「LSN_RET 列挙」118 ページ
- 「lsn_state 構造体」122 ページ
- 「メッセージ・クラス」113 ページ

例

次の例は、Sierra Wireless AirCard 510 Listener の実装に使用する LsnSuspendListening 関数です。`SwiApiClose()` 関数呼び出しによって、モデムが SMS メッセージの受信を中断します。

```
LsnExport LSN_RET LsnExportFunc LsnSuspendListening(
     lsn_state* state )
 /
*****************************************************/
 {
     if( !state->apiOpened ) {
          return( LSN_RET_OK );
     }
     state->apiOpened = FALSE;
     return( transformRcode( SwiApiClose() ) );
 }
```

次の例は、UDP Listener 用の LsnSuspendListening 関数の実装です。UDP Listener はこの関数をサポートしていないため、戻り値として LSN_RET_NOT_SUP を返しています。

LSN_RET_NOT_SUP の詳細については、「LSN_RET 列挙」118 ページを参照してください。

```
LsnExport LSN_RET LsnExportFunc LsnSuspendListening(
     lsn_state* state )
 /*****************************************************/
 {
     _unused( state );
     return( LSN_RET_NOT_SUP );
 }
```

LsnResumeListening 関数

機能

中断されたリスナを再開します。

プロトタイプ

LSN_RET **LsnResumeListening (** lsn_state* *state* **)**

パラメータ

- **state** Listener 状態構造体インスタンスへのポインタ。

  lsn_state の詳細については、「lsn_state 構造体」122 ページを参照してください。

戻り値

この関数では、LSN_RET 列挙に定義された次の戻り値を使用します。

値	説明
LSN_RET_OK	関数呼び出しが正常終了した。
LSN_RET_NOT_SUP	関数呼び出しが無効か適用できない。たとえば、UDP Listener の実装に、LsnSuspendListening 関数を使用することはできません。詳細については、「LsnSuspendListening 関数」131 ページを参照してください。
LSN_RET_NO_RESP	要求のタイムアウトが発生した。
LSN_RET_FAILED	一般的な失敗。

LSN_RET 列挙の詳細については、「LSN_RET 列挙」118 ページを参照してください。

説明

この関数は、中断した Listener を再開するために使用します。

参照

- 「LSN_RET 列挙」118 ページ
- 「lsn_state 構造体」122 ページ
- 「メッセージ・クラス」113 ページ

例

次の例は、Sierra Wireless AirCard 510 Listener の実装に使用する LsnResumeListening 関数です。`SwiApiOpen( SESSION_NAME )` 関数呼び出しによって、モデムがトリガされ、SMS メッセージの受信が再開されます。

```
LsnExport LSN_RET LsnExportFunc LsnResumeListening(
     lsn_state* state )
 /*************************************************/
 {
     LSN_RET ret;
     if( state->apiOpened ) {
          return( LSN_RET_OK );
     }
     ret = transformRcode( SwiApiOpen( SESSION_NAME ) );
     state->apiOpened = ( ret == LSN_RET_OK );
     return( ret );
 }
```

次の例は、UDP Listener 用の LsnResumeListening 関数の実装です。UDP Listener はこの関数をサポートしていないため、戻り値として LSN_RET_NOT_SUP を返しています。

LSN_RET_NOT_SUP の詳細については、「LSN_RET 列挙」118 ページを参照してください。

```
LsnExport LSN_RET LsnExportFunc LsnResumeListening(
     lsn_state* state
 )
 /
*************************************************/
 {
     _unused( state );
     return( LSN_RET_NOT_SUP );
 }
```

LsnReceiveAll 関数

機能

保留中のメッセージを受信します。

プロトタイプ

LSN_RET **LsnReceiveAll (**
lsn_state* *state*,
lsn_long* *nMsg*
)

パラメータ

- **state** lsn_state 構造体インスタンスへのポインタ。

  lsn_state の詳細については、「lsn_state 構造体」122 ページを参照してください。

- **nMsg** 受信されるメッセージの数。

戻り値

この関数では、LSN_RET 列挙に定義された次の戻り値を使用します。

値	説明
LSN_RET_OK	関数呼び出しが正常終了した。
LSN_RET_NO_RESP	要求のタイムアウトが発生した。
LSN_RET_FAILED	一般的な失敗。
LSN_RET_MSG_NOT_READ	Listener 記憶領域内に未読メッセージがある。新しいメッセージの受信要求は無視されました。

LSN_RET 列挙の詳細については、「LSN_RET 列挙」118 ページを参照してください。

説明

この関数は、保留中のメッセージを Listener 記憶領域内に受信するために使用します。この関数は、保留中のすべてのメッセージを受信するか、記憶領域が一杯になった時点で終了しなければなりません。

nMsg パラメータは、LsnReadNext 関数を使用して読み込むことができるメッセージの数を指します。

LsnReadNext 関数の詳細については、「LsnReadNext 関数」138 ページを参照してください。

参照

- 「LsnReadNext 関数」138 ページ
- 「LSN_RET 列挙」118 ページ
- 「lsn_state 構造体」122 ページ
- 「メッセージ・クラス」113 ページ

例

次の例は、UDP Listener の実装に使用する LsnReceiveAll 関数です。

```
LsnExport LSN_RET LsnExportFunc LsnReceiveAll(
     lsn_state* state, lsn_long* nMsg )
 /*******************************************/
 {
     LSN_RET ret = LSN_RET_FAILED;
     int wsaRet;
     struct sockaddr_in from;
```

```
    int fromLen = sizeof(from);
    fd_set readableSockets;
    char* senderIpA;
#if defined( UNICODE )
    int mbwcRet;
    WCHAR senderIpW[ MAX_SENDER_TCHARS + 1 ];
    #define RECVBUFF    (state->recvBuff)
    #define SENDERIP    senderIpW
#else
    #define RECVBUFF (state->msg->message)
    #define SENDERIP    senderIpA
#endif



   if( state->unread ) {
    return( LSN_RET_MSG_NOT_READ );
    }
    if( nMsg != NULL ) {
    *nMsg = 0;
    }
    state->unread = false;



// Select readable socket
    FD_ZERO( &readableSockets );
    FD_SET( state->socket, &readableSockets );
    switch( select( 0, &readableSockets, NULL, NULL,
&state->timeout ) ) {
    case 0:
        // Nothing to read
        ret = LSN_RET_OK;
    break;
    case 1:
        // Receive the message
    wsaRet = recvfrom( state->socket, RECVBUFF,
            MAX_MESSAGE_BYTES, 0,
            (struct sockaddr *)&from, &fromLen );
    if( wsaRet == SOCKET_ERROR ) {
        foundWSAError( state );
        break;
```

```
     }
     // Null terminate the message
     RECVBUFF[ wsaRet ] = '¥0';



#if defined( UNICODE )
     mbwcRet = MultiByteToWideChar( state->codePage, 0,
state->recvBuff,
              -1, state->msg->message,
MAX_MESSAGE_TCHARS );
     if( mbwcRet == 0 ) {
         MessageBox( NULL,
             TEXT("Failed to translate an incoming
message"),
         TEXT( "Message from lsn_udp.dll" ),
MB_ICONEXCLAMATION );
         break;
     }
 #endif



    // Retrieve the sender name and port
     senderIpA = inet_ntoa( from.sin_addr );
     if( senderIpA == NULL ) {
         break;
     }
 #if defined( UNICODE )
     size_t i;
     for( i = 0; i <= strlen( senderIpA ); i++ ) {
         senderIpW[i] = senderIpA[i];
     }
 #endif

if( state->showSenderPort ) {
         _stprintf( state->msg->sender, TEXT("%s:%d"),
             SENDERIP, ntohs( from.sin_port ) );
     } else {
         _tcscpy( state->msg->sender, SENDERIP );
     }
         if( nMsg != NULL ) {
         *nMsg = 1;
     }
```

```
        state->unread = true;
         ret = LSN_RET_OK;
         break;
         case SOCKET_ERROR:
         foundWSAError( state );
         break;
         }
         return( ret );
    }
```

LsnReadNext 関数

機能

次のメッセージをメッセージ・バッファ内に読み込みます。

プロトタイプ

LSN_RET **LsnReadNext (**
lsn_state* *state*,
a_msg* *msg*
)

パラメータ

- **state** Listener 状態構造体インスタンスへのポインタ。

  lsn_state の詳細については、「lsn_state 構造体」122 ページを参照してください。

- **msg** 読み込み先メッセージ・バッファへのポインタ。

戻り値

この関数では、LSN_RET 列挙に定義された次の戻り値を使用します。

値	説明
LSN_RET_OK	関数呼び出しが正常終了した。
LSN_RET_NO_MORE_MSG	未読メッセージがない。メッセージの読み取りに失敗しました。

LSN_RET 列挙の詳細については、「LSN_RET 列挙」118 ページを参照してください。

説明

この関数は、次のメッセージを Listener 記憶領域から読み込むために使用します。

参照

- 「LSN_RET 列挙」118 ページ
- 「lsn_state 構造体」122 ページ
- 「メッセージ・クラス」113 ページ

例

次の例は、UDP Listener の実装に使用する LsnReadNext 関数です。

```
LsnExport LSN_RET LsnExportFunc LsnReadNext(
 lsn_state* state, a_msg* msg )
 /*****************************************/
 {
     if( !state->unread ) {
     return( LSN_RET_NO_MORE_MSG );
     }
     msg->copy( state->msg );
     state->unread = false;
     return( LSN_RET_OK );
 }
```

LsnGetAddress

機能

Listener のアドレスを返します。

プロトタイプ

const TCHAR * **LsnGetAddress (** lsn_state* *state* **)**

パラメータ

- **state** Listener 状態構造体インスタンスへのポインタ。

  lsn_state の詳細については、「lsn_state 構造体」122 ページを参照してください。

戻り値

Listener アドレスが格納された文字配列。

説明

この関数は、Listener が使用するアドレスを返します。

このアドレスには、電話番号などが指定されています。

参照

- 「lsn_state 構造体」122 ページ
- 「メッセージ・クラス」113 ページ

例

次の例は、Sierra Wireless AirCard 510 Listener の実装に使用する LsnGetAddress 関数です。

```
LsnExport const TCHAR * LsnExportFunc LsnGetAddress(
   lsn_state* state )
 /************************************************/
 {
     if( state->phone == NULL ) {
     return( TEXT("Unspecified") );
     }
     return( state->phone );
 }
```

LsnGetMedium 関数

機能

Listener 通信に使用される Carrier または通信媒体を返します。

プロトタイプ

TCHAR * **LsnGetMedium (** lsn_state* *state* **);**

パラメータ

- **state** Listener 状態構造体インスタンスへのポインタ。

  lsn_state の詳細については、「lsn_state 構造体」122 ページを参照してください。

戻り値

通信媒体または Carrier の名前が格納される文字配列。

説明

この関数は、Listener が使用する Carrier または通信媒体を返します。

参照

- 「lsn_state 構造体」122 ページ
- 「メッセージ・クラス」113 ページ

例

次の例は、Sierra Wireless AirCard 510 Listener の実装に使用する LsnGetMedium 関数です。

```
LsnExport const TCHAR * LsnExportFunc LsnGetMedium(
   lsn_state* state )
 /************************************************/
 {
    if( state->carrier == NULL ) {
  return( TEXT("Unspecified") );
     }
    return( state->carrier );
 }
```

Palm 用 Listener SDK

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。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。

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」111 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。

  「概要」141 ページ

  「PalmLsnFree 関数」142 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」143 ページ

  「PalmLsnDupSender 関数」144 ページ

「PalmLsnDupTime 関数」145 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。

  詳細については、「PalmLsnProcess 関数」146 ページを参照してください。

PalmLsnAllocate 関数

機能 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」142 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

機能 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」141 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

機能

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」144 ページ
- 「PalmLsnDupTime 関数」145 ページ
- 「a_palm_msg 構造体」141 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

機能

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」143 ページ
- 「PalmLsnDupTime 関数」145 ページ
- 「a_palm_msg 構造体」141 ページ

PalmLsnDupTime 関数

機能

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」143 ページ
- 「PalmLsnDupSender 関数」144 ページ
- 「a_palm_msg 構造体」141 ページ

PalmLsnProcess 関数

機能

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」153 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」149 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」66 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。

action 変数の詳細については、「action 変数」59 ページを参照してください。

参照

- 「Palm Listener 設定ユーティリティ」66 ページ
- 「メッセージ・ハンドラ」39 ページ
- 「action 変数」59 ページ
- 「PalmLsnCheckConfigDB 関数」148 ページ
- 「a_palm_msg 構造体」141 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

機能

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」153 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」149 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」146 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

機能

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。

値	説明
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」146 ページ

LsnMain 関数

機能

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」146 ページ
- 「Palm 用 Listener SDK ファイル」111 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 および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」152 ページ

  「PalmLsnTargetDeviceID」152 ページ

  「PalmLsnGetConfigFileName」153 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」154 ページ

「PalmLsnNormalStop」154 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」154 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」155 ページ

PalmLsnTargetCompanyID

機能 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 および 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」152 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

機能 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 および 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」152 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

機能　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」146 ページ
- 「Palm 用 Listener SDK ファイル」111 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

機能 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」154 ページ
- 「PalmLsnSpecialLaunch」155 ページ

PalmLsnNormalStop

機能 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」154 ページ

PalmLsnNormalHandleEvent

機能 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

機能　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 );
 }
```

第 8 章

チュートリアル：サーバ起動同期

この章の内容

この章では、サーバ起動同期用の統合データベースおよびリモート・データベースの設定方法について説明します。ここでは、Adaptive Server Anywhere 統合データベースを使用して簡単な同期を最初から設定する方法を説明します。

サーバ起動同期の実装サンプルが、SQL Anywhere Studio のインストール環境にいくつか含まれています。これらのサンプルは、readme ファイルやコードのコメントで詳細に説明されています。サンプル・アプリケーションを見つけるには、SQL Anywhere Studio のインストール・パスの *Samples¥MobiLink* ディレクトリに移動します。サーバ起動同期のサンプル・ディレクトリは、いずれも SIS_ というプレフィクスで始まっています。

Car Dealer サンプルを使用したサーバ起動同期

Car Dealer サンプル

このチュートリアルは、SQL Anywhere インストール環境の *Samples¥MobiLink¥SIS_CarDealer* サブディレクトリにあるサンプルに基づいています。ここでは、デスクトップ・コンピュータ上にある単一のクライアントを使用しています。ただし、サーバ起動同期 Listener は、Windows CE や Palm OS などの他のプラットフォームにもインストールできます。詳細については、SIS_CarDealer サンプルおよび Mobile Link サーバ起動同期のマニュアルを参照してください。

チュートリアルの概要

ここでは、Car Dealer サンプルを使用して Mobile Link サーバ起動同期を設定するのに必要な手順の概要を説明します。このレッスンの目的は、以下に挙げる項目の概要を示すことです。これらの手順の実行方法については、後の項で説明します。

❖ 統合データベースと Notifier ユーティリティの設定

1. 次のスキーマで Adaptive Server Anywhere データベースを作成します。
   - 同期用に使用するテーブル

     このチュートリアルでは、Dealer というテーブルを使用します。このテーブルには、自動車製造メーカの名前とその評価が格納されています。
   - ダウンロード専用同期の同期論理

     このチュートリアルでは、download_cursor 同期スクリプトを実装します。
   - Push 要求を格納するテーブル

     このテーブルを移植すると、リモート通知がトリガされます。
2. Notifier ユーティリティを設定します。

begin_poll、request_cursor、request_delete イベントを処理する論理を指定します。

3　ゲートウェイと Carrier を設定します。

このチュートリアルでは、UDP Listener 用のデフォルトのゲートウェイ設定を使用します。

4　-notifier オプションを使用して Mobile Link 同期サーバを起動します。

❖ リモート・データベースと Listner ユーティリティの設定

1　同期で使用するテーブルを含む Adaptive Server Anywhere データベースを作成します。

このテーブルは、統合データベースの Dealer テーブルと同期します。

2　リモート同期パブリケーション、同期ユーザ、および同期サブスクリプションを作成します。

3　Mobile Link Listener 用のコマンド・ライン・ファイルである *dblsn.txt* を作成します。

4　ローカルの UDP Listener を起動します。

❖ Push 要求の発行

- Push 要求を発行するには、データを直接 Push 要求テーブルに挿入するか、Notifier の begin_poll イベントが Push 要求テーブルを移植するような変更を加えます。各要求により、Notifier は request_cursor イベントを使用してメッセージを送信します。

  Push 要求のメッセージが 'sync' などの Listener コマンド・ファイルに定義されたメッセージと一致する場合、リモート・データベースは同期を行います (または対応するアクションを実行します)。

レッスン 1 ：統合データベースの設定

このレッスンでは、同期に必要なスクリプトを使用して統合データベースを作成します。

Adaptive Server Anywhere データベースを作成する方法の 1 つに、dbinit コマンド・ライン・ユーティリティを使用するやり方があります。このチュートリアルでは、**cons** という統合データベースを使用します。

❖ Adaptive Server Anywhere 統合データベースを作成して起動するには、次の手順に従います。

1 コマンド・プロンプトで、データベースを作成するディレクトリに移動します。

2 次のコマンドを入力して、データベースを作成します。

```
dbinit cons.db
```

3 ここで、次のように入力してデータベースを起動します。

```
dbeng9 cons.db
```

統合データベース・スキーマの生成

統合データベース・スキーマには、Dealer テーブル、dowonload_cursor 同期スクリプト、およびサーバ起動同期の Push 要求を生成するテーブルとストアド・プロシージャが含まれています。

❖ Dealer テーブルと download_cursor 同期スクリプトを追加するには、次の手順に従います。

1 統合データベースに接続します。

- Sybase Central で、[Adaptive Server Anywhere 9] プラグインを選択し、[ファイル] メニューから [接続] を選択します。

  [接続] ダイアログが表示されます。

- [ID] タブで、ユーザ ID として **DBA**、パスワードとして **SQL** を入力します。[データベース] タブで、[サーバ名] として **cons** と入力します。
- [OK] をクリックして接続します。

2 Interactive SQL を起動します。

- Sybase Central で、cons データベースを選択します。[ファイル] ー [Interactive SQL を開く] を選択します。

3 Dealer テーブルと download_cursor 同期スクリプトをインストールします。

- Interactive SQL で次のコマンドを実行します。

```
/* the dealer table */
 create table Dealer (
     name        varchar(10) not null primary
key,
     rating        varchar(5),
     last_modified    timestamp default
timestamp
 )
 go
 insert into Dealer(name, rating) values (
'Audi', 'a');
 insert into Dealer(name, rating) values (
'Buick', 'b');
 insert into Dealer(name, rating) values (
'Chrysler', 'c' );
 insert into Dealer(name, rating) values (
'Dodge', 'd');
 insert into Dealer(name, rating) values (
'Eagle', 'e');
 insert into Dealer(name, rating) values (
'Ford', 'f');
 insert into Dealer(name, rating) values (
'Geo', 'g');
 insert into Dealer(name, rating) values (
'Honda', 'h');
 insert into Dealer(name, rating) values (
'Isuzu', 'i');
 go
```

```
 /* the download_cursor synchronization script
*/
 call ml_add_table_script( 'sis_ver1',
'Dealer', 'download_cursor',
 'SELECT * FROM Dealer WHERE last_modified >= ?'
)
 go
```

詳細情報

このレッスンの各トピックの詳細については、次の各項を参照してください。

- 『ASA データベース管理ガイド』> 「初期化ユーティリティ」
- 『ASA データベース管理ガイド』> 「データベース・サーバ」
- 『SQL Anywhere Studio の紹介』> 「Interactive SQL の使用」
- 『ASA SQL リファレンス・マニュアル』> 「CREATE TABLE 文」
- 『Mobile Link 管理ガイド』> 「同期スクリプトの作成」
- 『Mobile Link 管理ガイド』> 「download_cursor テーブル・イベント」

レッスン 2 ：Push 要求テーブルの作成

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。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。

❖ Push 要求用のテーブルを作成するには、次の手順に従います。

1　Interactive SQL で次のコマンドを実行します。

```
CREATE TABLE PushRequest (
    req_id    INTEGER DEFAULT AUTOINCREMENT PRIMARY
KEY,
     mluser    VARCHAR(128),
     subject    VARCHAR(128),
     content VARCHAR(128),
     resend_interval VARCHAR(30) DEFAULT '20s',
     time_to_live VARCHAR(30) DEFAULT '1m',
     status    VARCHAR(128) DEFAULT 'created'
 )
 go
```

2　Interactive SQL を閉じます。

詳細情報

このレッスンの各トピックの詳細については、次の各項を参照してください。

- 「Push 要求」12 ページ
- 「サーバ起動同期の概要」1 ページ
- 「サーバ起動同期のコンポーネント」5 ページ

レッスン 3 ：Notifier の設定

このレッスンでは、Notifier による Push 要求の作成、リモート Listener への要求の送信、および有効期限が切れた要求のクリーンアップ方法に影響する 3 つの Notifier プロパティを設定します。

❖ Notifier ユーティリティを設定するには、次の手順に従います。

1 Mobile Link 同期 9 プラグインを使用して、統合データベースに接続します。

- Sybase Central を開きます。
- 左ウィンドウ枠で、[Mobile Link 同期 9] プラグインを選択し、[ファイル] メニューから [接続] を選択します。

  [接続] ダイアログが表示されます。
- [ID] タブで、ユーザ ID として **DBA**、パスワードとして **SQL** を入力します。[データベース] タブで、[サーバ名] として **cons** と入力します。
- [OK] をクリックして接続します。

2 新しい Notifier を追加します。

左ウィンドウ枠で [通知] フォルダを開き、[Notifier] フォルダを選択します。右ウィンドウ枠で、[Notifier の追加] をダブルクリックします。

[新しい Notifier の追加] ダイアログが表示されます。

3 Notifier に **CarDealerNotifier** という名前を指定します。[完了] をクリックします。

4 begin_poll イベント・スクリプトを入力します。

Notifier が統合データベース内の変更を検出し、begin_poll イベントを使用して Push 要求を作成します。この場合、begin_poll スクリプトは、Dealer テーブルで変更が発生し、リモート・データベースが最新のものではないときに PushRequest テーブルを移植します。

- 右ウィンドウ枠で、[CarDealerNotifier] を選択します。[ファイル] ー [プロパティ] を選択します。

  [CarDealerNotifier Notifier プロパティ] ダイアログが表示されます。

- [論理] タブをクリックします。ドロップダウン・メニューから、[begin_poll] を選択します。

- begin_poll スクリプトに対して次のように入力します。

```
 --
 -- Insert the last consolidated database
 -- modification date into @last_modified
 --
 declare @last_modified timestamp;
 select max( last_modified ) into
@last_modified from Dealer;


--
 -- Delete processed requests if the mluser is
up-to-date
 --
 delete from PushRequest
   from PushRequest as p, ml_user as u,
ml_subscription as s
 where
         p.status = 'processed'
 and
   u.name = p.mluser
 and
   u.user_id = s.user_id
 and
   @last_modified <= greater(
s.last_upload_time, s.last_download_time );
```

```
 --
 -- Insert new requests when a device is not up-
to-date
 --
 insert into PushRequest( mluser, subject,
content )
 select u.name, 'sync', 'ignored'
  from ml_user as u, ml_subscription as s
 where
  u.name in ( select ml_user from ml_listening
where listening = 'y' )
 and
  u.user_id = s.user_id
 and
  @last_modified > greater( s.last_upload_time,
s.last_download_time )
 and
  u.name not like '%-dblsn'
 and
  not exists( select * from PushRequest
   where PushRequest.mluser = u.name
      and PushRequest.subject = 'sync')
```

begin_poll スクリプトの最初の主要なセクションでは、デバイスが最新の場合、PushRequest テーブルからの処理済み要求が削除されます。

```
@last_modified <= greater( s.last_upload_time,
s.last_download_time)
```

@last_modified は、統合データベースの Dealer テーブルにおける最新修正日です。大きい方の式 (s.last_upload_times、last_download_time) が、リモート・データベースの最終同期時間を表します。

request_delete イベントを使用して Push 要求を直接削除することもできます。ただし、この場合は、begin_poll イベントによって、リモート・データベースが同期を実行する前に、有効期限が切れた要求や暗黙に削除された要求が削除されないことが保証されます。

次のコード・セクションは、Dealer テーブルの last_modified カラムの変更をチェックして、(ml_listening テーブルにリストされた) 最新ではないすべてのアクティブ・リスナに対して Push 要求を発行します。

```
@last_modified > greater( s.last_upload_time,
s.last_download_time)
```

PushRequest テーブルの移植時に、begin_poll スクリプトはサブジェクトを 'sync' に設定します。

5　request_cursor スクリプトを入力します。

request_cursor スクリプトが Push 要求をフェッチします。各 Push 要求は、メッセージで送信された情報と、その情報を受信したリモート・データベースを判別します。

- ドロップダウン・メニューから、[request_cursor] を選択します。
- request_cursor スクリプト用に次のコードを入力します。

```
select
    p.req_id,
    'Default-DeviceTracker',
    p.subject,
    p.content,
    p.mluser,
    p.resend_interval,
    p.time_to_live
 from PushRequest as p
```

PushRequest テーブルによって、request_cursor スクリプトにローが提供されます。

request_cursor 結果セット内の順序と値は重要です。たとえば、2 番目のパラメータはデフォルトのゲートウェイ Default-DeviceTracker を定義します。デバイス・トラッカ・ゲートウェイは、ユーザのアクセス方法を追跡し、UDP または SMTP を自動的に選択してリモート・デバイスに接続します。

6　request_delete スクリプトを入力します。

request_delete Notifier イベントは、クリーンアップ操作を指定します。このスクリプトを使用すると、Notifier は暗黙的に削除された要求や有効期限の切れた要求を自動的に削除できます。

- ドロップダウン・メニューから、[request_delete] を選択します。
- request_delete スクリプト用に次のコードを入力します。

```
update PushRequest set status='processed' where
req_id = ?
```

ローを削除する代わりに、この request_delete スクリプトが PushRequest テーブル内のローのステータスを 'processed' に更新します。

7 [OK] をクリックして、Notifier プロパティを保存します。

詳細情報

このレッスンの各トピックの詳細については、次の各項を参照してください。

- 『SQL Anywhere Studio の紹介』> 「Sybase Central を使用したデータベースの管理」
- 「request_delete プロパティ」83 ページ
- 「begin_poll プロパティ」74 ページ
- 『Mobile Link 管理ガイド』> 「ml_listening」
- 「デバイス・トラッキング」27 ページ
- 「デバイス・トラッキング用の Listener オプション」29 ページ
- 「request_cursor プロパティ」82 ページ
- 「request_delete プロパティ」83 ページ

レッスン 4 ：ゲートウェイと Carrier の設定

ゲートウェイは、メッセージを送信するメカニズムです。ユーザは、UDP ゲートウェイと SMTP ゲートウェイを定義できます。別の方法として、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・トラッキングを使用すると、Mobile Link はユーザのアクセス方法を追跡し、UDP ゲートウェイを使用するか SMTP ゲートウェイを使用するかを自動的に判断します。

このチュートリアルでは、デフォルトのデバイス・トラッカ・ゲートウェイを使用するため、設定は不要です。

詳細情報

このレッスンの各トピックの詳細については、次の各項を参照してください。

- 「デバイス・トラッカ・ゲートウェイ・プロパティ」87 ページ
- 「ゲートウェイと Carrier」24 ページ

レッスン 5 ：ODBC データ・ソースの定義

Adaptive Server Anywhere 9.0 ドライバを使用して、データベース用の ODBC データ・ソースを定義します。

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

1 ODBC アドミニストレータを起動します。

[スタート] メニューから、[プログラム] － [Sybase SQL Anywhere 9] － [Adaptive Server Anywhere] － [ODBC アドミニストレータ] の順に選択します。

[ODBC データ・ソース・アドミニストレータ] が表示されます。

2 [ユーザ DSN] タブで [追加] をクリックします。

[データ・ソースの新規作成] ダイアログが表示されます。

3 [Adaptive Server Anywhere 9.0] を選択して [完了] をクリックします。

[Adaptive Server Anywhere 9 の ODBC 設定] ダイアログが表示されます。

4 [ODBC] タブで、データ・ソース名 **sis_cons** を入力します。[ログイン] タブで、ユーザ ID として **DBA**、パスワードとして **SQL** を入力します。[データベース] タブで、[サーバ名] として **cons** と入力します。

5 [OK] をクリックします。

詳細情報

このレッスンの各トピックの詳細については、次の各項を参照してください。

- 『ASA データベース管理ガイド』> 「ODBC データ・ソースの使用」

Adaptive Server Anywhere 以外の統合データベースを使用する場合は、『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』 > 「iAnywhere Solutions ODBC ドライバの概要」を参照してください。

レッスン 6 ：Mobile Link サーバの起動

❖ **Mobile Link 同期サーバ (dbmlsrv9) を実行するには、次の手順に従います。**

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

```
dbmlsrv9
    -notifier
    -c "dsn=sis_cons"
    -o ml.log
    -fr
    -v+
    -zu+
    -x tcpip
```

次の表に、dbmlsrv9 ユーティリティで使用する各オプションについて説明します。オプション -o と -v は、デバッグとトラブルシューティングの情報を提供します。これらのロギング・オプションは、開発環境での使用に適しています。パフォーマンス上の理由から、一般的に -v は運用環境では使用しません。

オプション	説明
-notifier	サーバ起動同期用に Notifier を開始します。 『Mobile Link 管理ガイド』> 「-notifier オプション」を参照してください。
-c	接続文字列を指定します。 『Mobile Link 管理ガイド』> 「-c オプション」を参照してください。
-o	メッセージ・ログ・ファイル *ml.log* を指定します。 『Mobile Link 管理ガイド』> 「-o オプション」を参照してください。

オプション	説明
-fr	同期にデータをアップロードするスクリプトとダウンロードするスクリプトが 1 つも含まれていない場合、Mobile Link 同期サーバが中止されないようにします。ダウンロード専用同期を使用するこのチュートリアルでは、このオプションが必要です。 『Mobile Link 管理ガイド』>「-fr オプション」を参照してください。
-v+	-v オプションは、ログを取る情報を指定します。-v+ を使用して、最大冗長ロギングをオンに設定します。 『Mobile Link 管理ガイド』>「-v オプション」を参照してください。
-zu+	自動的に新しいユーザを追加します。 『Mobile Link 管理ガイド』>「-zu オプション」を参照してください。
-x	Mobile Link クライアントの通信プロトコルとプロトコル・オプションを設定します。 『Mobile Link 管理ガイド』>「-x オプション」を参照してください。

Mobile Link 同期サーバが要求を処理する準備ができたことを示すダイアログが表示されます。また、Notifier ユーティリティも表示されます。

詳細情報

このレッスンの各トピックの詳細については、次の各項を参照してください。

- 『Mobile Link 管理ガイド』>「Mobile Link 同期サーバの実行」
- 『Mobile Link 管理ガイド』>「Mobile Link 同期サーバのオプション」

レッスン 7 ：リモート・データベースの設定

Mobile Link サーバ起動同期は、統合データベース・サーバや多数のモバイル・データベースを使用する同期用に設計されています。このレッスンでは、Adaptive Server Anywhere リモート・データベースを作成し、同期パブリケーション、ユーザ、およびサブスクリプションを作成します。次に、Listener ユーティリティ用のコマンド・ファイルを作成して、Listener を起動します。

❖ 新しい Adaptive Server Anywhere リモート・データベースを作成して起動するには、次の手順に従います。

1 コマンド・プロンプトで、データベースを作成するディレクトリに移動します。

2 次のコマンドを入力して、データベースを作成します。

```
dbinit rem1.db
```

3 ここで、次のように入力してデータベースを起動します。

```
dbeng9 rem1.db
```

❖ リモート・データベース・スキーマを生成するには、次の手順に従います。

1 データベースに接続します。

- Sybase Central で、[Adaptive Server Anywhere 9] プラグインを選択し、[ファイル] メニューから [接続] を選択します。

  [接続] ダイアログが表示されます。

- [ID] タブで、ユーザ ID として **DBA**、パスワードとして **SQL** を入力します。[データベース] タブで、[サーバ名] として **rem1** と入力します。

- [OK] をクリックして接続します。

2　Interactive SQL を起動します。

- Sybase Central で、rem1 データベースを選択します。[ファイル] ー [Interactive SQL を開く] を選択します。

3　Dealer テーブルを作成します。

- Interactive SQL で次のコマンドを実行します。

```
create table Dealer (
     name        varchar(10) not null primary
key,
     rating        varchar(5),
     last_modified    timestamp default
timestamp
 )
 go
```

4　リモート同期パブリケーション、同期ユーザ、および同期サブスクリプションを作成します。

- Interactive SQL で次のコマンドを実行します。

```
CREATE PUBLICATION car_dealer_pub (table
Dealer);
 CREATE SYNCHRONIZATION USER sis_user1;
 CREATE SYNCHRONIZATION SUBSCRIPTION
    TO car_dealer_pub
      FOR sis_user1
      OPTION scriptversion='sis_ver1';
```

詳細情報

このレッスンの各トピックの詳細については、次の各項を参照してください。

- 『Mobile Link クライアント』>「Mobile Link クライアントの紹介」
- 『ASA データベース管理ガイド』>「初期化ユーティリティ」
- 『ASA SQL リファレンス・マニュアル』>「CREATE TABLE 文」
- 『Mobile Link クライアント』>「データのパブリッシュ」
- 『ASA SQL リファレンス・マニュアル』>「CREATE PUBLICATION 文」

- 『ASA SQL リファレンス・マニュアル』> 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」
- 『ASA SQL リファレンス・マニュアル』> 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」
- 『Mobile Link 管理ガイド』> 「スクリプト・バージョン」

レッスン 8 ：Listener の設定

Listener は、リモート・デバイスで実行されます。Listener は、Notifier からのメッセージを受信して処理し、アクションを実行します。たとえば、Listener は、次の dblsn オプションが指定されている場合にメッセージ "sync" を受信すると、dbmlsync を起動します。

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

Listener を設定する簡単な方法は、コマンド・ライン・オプションをテキスト・ファイルに格納することです。たとえば、設定を *mydblsn.txt* に格納し、次のように入力して Lisetner を起動します。

```
dblsn @mydblsn.txt
```

別の方法として、パラメータなしで 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。

❖ Mobile Link Listener を作成して起動するには、次の手順に従います。

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

```
#-----------------------------------
 # Verbosity level
 -v2

 # Show notification messages in console and log
 -m

 # Polling interval, in seconds
 -i 3

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

```
# Mobilink address and connect parameter for dblsn
-x "host=localhost"

# Enable device tracking and specify the MobiLink
user name.
-t+ sis_user1

# Message handlers
# Synchronize using dbmlsync
-l "subject=sync;
action='start dbmlsync.exe
 -c eng=rem1;uid=dba;pwd=sql
 -ot dbmlsyncOut.txt -k';"
```

- *mydblsn.txt* という名前でファイルを保存します。

2 Listener を起動します。

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

次のように入力して、Listener を起動します。

```
dblsn @mydblsn.txt
```

Listener が実行され、Mobile Link 同期サーバにアップロードされたデバイス・トラッキング情報が含まれていることを示すダイアログが表示されます。

トラッキング情報が統合データベースにアップロードされると、Mobile Link 同期サーバのウィンドウ内に新しいエントリが表示されます。この情報により、Listener と Mobile Link 同期サーバ間の初期通信の成功が伝えられます。

詳細情報

このレッスンの各トピックの詳細については、次の各項を参照してください。

- 「Listener」37 ページ
- 「Listener ユーティリティ」50 ページ
- 「Listener オプションの保存」44 ページ

レッスン 9 ：Push 要求の発行

サーバ起動同期では、PushRequest テーブルに直接移植するか、または Dealer テーブルを変更することで、Push 要求を発行できます。Dealer テーブルを変更する場合は、Notifier begin_poll スクリプトが Dealer テーブル内の変更を検出し、PushRequest テーブルに移植します。

いずれの場合も、PushRequest テーブルが、リモート・デバイスのメッセージの受信方法を決める Notifier request_cursor スクリプトに対してローを提供します。

❖ Push 要求を PushRequest テーブルに直接挿入して、サーバ起動同期を行うには、次の手順に従います。

1　Interactive SQL から cons.db データベースに接続して、次のように入力します。

```
INSERT INTO pushrequest(mluser,subject,content)
   VALUES ('sis_user1','sync','not used');
   COMMIT;
```

2　同期が実行されるまで数秒待機します。

移植が実行されると、PushRequest テーブルが Notifier の request_cursor スクリプトに対してローを提供します。request_cursor スクリプトは、メッセージに送信される情報とその情報を受信するリモート・デバイスを判別します。

❖ 統合データベースの Dealer を変更して、サーバ起動同期を行うには、次の手順に従います。

1　Interactive SQL から、次のコマンドを入力します。

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

2　同期が実行されるまで数秒待機します。

この場合、Notifier begin_poll スクリプトが Dealer テーブル内の変更を検出し、適切に PushRequest テーブルに移植します。上記で述べたとおり、いったん PushRequest テーブルが移植されると、Notifier request_cursor スクリプトは、メッセージに送信される情報とその情報を受信するリモート・デバイスを判別します。

詳細情報

このレッスンの各トピックの詳細については、次の各項を参照してください。

- 「Push 要求の作成」14 ページ
- 『ASA SQL リファレンス・マニュアル』>「INSERT 文」
- 『ASA SQL リファレンス・マニュアル』>「UPDATE 文」

索引

記号

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

あ

き

く

け

こ

さ

し

す

せ

そ

ち

つ

て

と

に

は

ひ

ふ

へ

ま

め

ゆ