SYBASE iAnywhere

Mobile Link
サーバー起動同期

バージョン 12.0.1

2012 年 1 月

バージョン 12.0.1
2012 年 1 月

目次

はじめに

このマニュアルでは、Mobile Link のサーバー起動同期について説明します。サーバー起動同期とは、Mobile Link サーバーから同期の開始またはリモートデバイス上でのアクションの実行を可能にする機能です。

サーバー起動同期

注意
Sybase Central を使用してリモートデータベースを管理し、その後、サーバー起動同期への代替機能としてサーバー起動リモートタスク (SIRT) を使用できます。 詳細については、「リモートデータベースの集中管理」『Mobile Link サーバー管理』と「サーバー起動リモートタスク (SIRT)」『Mobile Link サーバー管理』を参照してください。

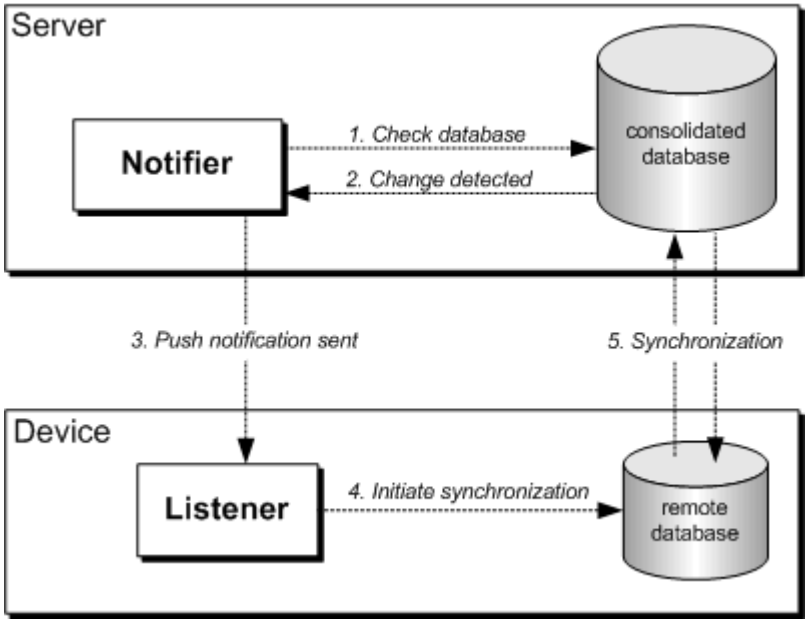
Mobile Link サーバー起動同期を使用すると、統合データベースから同期を開始できます。リモートデータベースに Push 通知を送信し、リモートデータベースから統合データベースを更新できます。この Mobile Link コンポーネントには、統合データベースで発生した変更の検出による同期の開始、Push 通知を送信するデバイスの選択、その Push 通知に対するデバイスの応答方法の決定を行うためのプログラム可能なオプションが用意されています。

例

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

サーバー起動同期のプロセス

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



サーバー起動同期プロセスでは、次の手順が実行されます。

1. Notifier はビジネスロジックに基づいたクエリを使用して統合データベースをチェックし、リモートデータベースとの同期が必要な変更があるかどうかを確認します。

2. 変更を検出すると、Notifier はデバイスに Push 通知を送信する準備を行います。

3. Notifier は Push 通知を送信します。Push 通知は、Device Tracker、UDP、SMTP、または SYNC ゲートウェイを使用して送信できます。

4. Listener は、件名、内容、または送信元をメッセージフィルターと照らし合わせて比較します。

5. フィルター条件が満たされると、アクションが開始されます。たとえば、標準的な実装では、アクションによって Mobile Link クライアントを実行したり、Ultra Light アプリケーションを起動したりできます。

サーバー起動同期のコンポーネント

Mobile Link サーバー起動同期では、次のコンポーネントが必要です。

- **Push 要求**　Push 要求は結果セット内の値のローで、デバイスに Push 通知を送信するよう Notifier に指示します。Push 要求によって、サーバー起動同期が実行されます。Notifier などの、あらゆるデータベースアプリケーションで Push 要求を作成できます。たとえば、価格が変更されたときにアクティブになるデータベーストリガーを使用して、Push 要求を作成できます。 「Push 要求」7 ページを参照してください。

- **Mobile Link Notifier**　Notifier は、Mobile Link サーバーに統合されたプログラムです。統合データベースを頻繁にチェックして、Push 要求があるかどうかを確認します。Notifier が Push 要求をチェックする頻度を制御するには、Notifier のプロパティを指定します。Push 要求をチェックするビジネスロジックと、通知対象のデバイスを決定するビジネスロジックを指定する必要があります。Notifier が Push 要求を検出すると、デバイスに Push 通知が送信されます。 「Notifier」12 ページを参照してください。

- **Mobile Link Listener**　Listener は、デバイス上で実行される 1 つのプログラムです。Notifier から Push 通知を受信すると、メッセージハンドラーを使用してメッセージをフィルターし、アクションを開始します。一般的なアプリケーションのアクションは同期の呼び出しですが、アプリケーションから他のアクションも実行できます。選択したサーバーソースからの Push 通知や特定の内容が含まれる Push 通知に対して、異なる動作をするように Mobile Link Listener を設定できます。

  Windows デバイスでは、Mobile Link Listener はコマンドラインオプションを使用して設定する実行プログラムです。Push 通知を受信するには、デバイスの電源がオンになっていて、Mobile Link Listener が動作中である必要があります。 「Windows デバイス用の Mobile Link Listener ユーティリティ (dblsn)」55 ページを参照してください。

- **ライトウェイトポーラー**　ライトウェイトポーラーは、指定された時間間隔で Push 通知をポーリングするデバイスアプリケーションです。ライトウェイトポーラーを使用すると、ゲートウェイを設定する代替手段となります。また、サーバーへの永続的な接続を必要とせず、バッテリの寿命を伸ばすことができるため、ライトウェイトポーラーを使用することをおすすめします。

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

  次の項を参照してください。

  ○「ライトウェイトポーリングオプションの設定」19 ページ
  ○「ライトウェイトポーリング API」81 ページ

- **ゲートウェイ (ライトウェイトポーラーの代替手段)**　ゲートウェイでは、デバイスに Push 通知を送信するための Notifier インターフェイスを提供します。ゲートウェイは、ライトウェイトポーラーの代替となる手段です。デバイストラッキングゲートウェイ、SYNC ゲートウェイ、UDP ゲートウェイ、または SMTP ゲートウェイを使用して、メッセージを送信できます。 「ゲートウェイと Carrier」22 ページを参照してください。

注意
Sybase Central を使用してリモートデータベースを管理し、その後、サーバー起動同期への代替機能としてサーバー起動リモートタスク (SIRT) を使用できます。 詳細については、「リモートデータベースの集中管理」『Mobile Link サーバー管理』と「サーバー起動リモートタスク (SIRT)」『Mobile Link サーバー管理』を参照してください。

サーバー起動同期の配備に関する考慮事項

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

UDP ゲートウェイを使用する場合のデバイスの制限事項

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

デバイストラッキングの制限事項

SQL Anywhere 9.0.0 または以前の Mobile Link Listener では、デバイストラッキングをサポートしていません。これらの Mobile Link Listener でデバイストラッキングを使用するには、デバイストラッキングを手動で設定する必要があります。

サポートされるデバイスプラットフォーム

Mobile Link Listener は、Windows と Windows Mobile でサポートされています。

参照

- 「デバイストラッキングのサポート」24 ページ

サーバー起動同期のクイックスタート

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

1. Mobile Link サーバーで、Push 要求を格納するための統合データベースを準備します。「Push 要求の要件」7 ページを参照してください。
2. Mobile Link サーバーで、Push 要求を作成および管理する Notifier イベントを設定します。「Notifier イベントとプロパティの設定」13 ページを参照してください。
3. デバイスで、ライトウェイトポーラーを設定します。「ライトウェイトポーラー」21 ページを参照してください。

   ライトウェイトポーラーを使用しない場合は、Mobile Link サーバーでサポートされているゲートウェイを設定します。SMTP ゲートウェイを使用する場合は、Carrier も設定する必要があります。 「ゲートウェイと Carrier」22 ページを参照してください。
4. デバイスで、メッセージをフィルターしてアクションを実行する Mobile Link Listener を設定します。 「メッセージハンドラー」15 ページを参照してください。

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

その他のリソース

- サンプルアプリケーションが *%SQLANYSAMP12%¥MobiLink¥* ディレクトリにインストールされています。サーバー起動同期に関連するすべてのアプリケーションは、**SIS_** プレフィクスの付いたディレクトリに配置されています。

サーバー起動同期の設定

次の項では、サーバー起動同期の設定方法について説明します。

- 「Push 要求」
- 「Notifier」
- 「Listener」
- 「ライトウェイトポーラー」
- 「ゲートウェイと Carrier」

Push 要求

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

Push 要求の要件

Push 要求の要件は、Mobile Link サーバーがデバイスとの通信に使用している方法によって異なります。すべての Push 要求に、subject カラムと content カラムが必要となります。

ライトウェイトポーラーを使用して Push 通知をポーリングする場合は、poll key カラムを作成して Push 通知を識別できるようにします。

ゲートウェイを使用して Push 通知を送信する場合は、gateway カラムと address カラムを作成する必要があります。

システムに Push 要求カラムがある場合は、カラムを作成する必要はありません。Push 要求の要件が満たされると、Push 要求を使用できます。 「Push 要求の使用」9 ページを参照してください。

ライトウェイトポーラーを使用する場合の Push 要求の要件 (推奨)

ライトウェイトポーラーを使用して Push 通知をポーリングする場合は、次のカラムを作成します。

カラム	型	説明
Poll key	VARCHAR	ライトウェイトポーラーを識別するために使用するキー。各ライトウェイトポーラーはユニークなキーを送信して、Mobile Link サーバー上で自身を識別します。

カラム	型	説明
Subject	VARCHAR	メッセージの件名の行です。
Content	VARCHAR	メッセージの内容。

ゲートウェイを使用する場合の Push 要求の要件 (推奨)

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

カラム	型	説明
Request ID	INTEGER	オプション。Push 要求のユニークな ID。 一部の Notifier イベントでは、このカラム名が必須です。「Notifier イベント」35 ページを参照してください。
Gateway	VARCHAR	メッセージの送信先ゲートウェイの名前。
Subject	VARCHAR	メッセージの件名の行です。
Content	VARCHAR	メッセージの内容。
Address	VARCHAR	デバイスの送信先アドレス。
Resend interval	VARCHAR	オプション。メッセージが再送される時間間隔。 resend interval は、信頼性の低いネットワークで UDP ゲートウェイを使用する場合に便利です。Notifier は、Push 要求に関連付けられたすべての属性が変更されないことを前提としています。要求を最初にポーリングした後、後続の更新は無視されます。次のポーリング時刻の前に Push 通知を送信する必要がある場合、Notifier は次のポーリング間隔を自動的に調整します。Push 要求の送信を停止するには、request_cursor イベントで同期論理を使用します。対象 Mobile Link Listener から受信確認が届くと、次の再送は停止されます。 「request_cursor イベント」39 ページを参照してください。
Time to live	VARCHAR	オプション。再送の有効期限が切れるまでの時間です。

参照

- 「Notifier イベントとプロパティの設定」13 ページ
- 「request_cursor イベント」39 ページ

例

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

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

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

Push 要求の使用

Push 要求の生成

Push 要求を生成するには、サーバー起動同期に必要な Push 要求のカラムが対象となるデータベースに含まれている必要があります。また、単一のデータベースクエリを使用して値を取得できる必要もあります。Push 要求は、Push 要求のカラムを選択する request_cursor イベントでデータベースクエリを指定すると、自動的に生成されます。 Push 要求の要件の詳細については、「Push 要求の要件」7 ページを参照してください。

ライトウェイトポーラーを使用した Push 要求の生成例

Mobile Link サーバーが unique_device_ID として検出したリモートデバイスがあり、統合データベースに次の SQL 文を使用して作成された PushRequest という名前のテーブルが含まれるとします。

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

この例では、Push 要求の準備に、統合データベースで次の SQL 文を実行します。

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

上記のスクリプトを使用して PushRequest テーブルに値を挿入しても、Push 要求自体は生成されません。Mobile Link サーバーの request_cursor イベントでデータベースクエリを設定して、挿入する値を選択し、Push 要求を生成できるようにします。

この例では、Mobile Link サーバーの request_cursor イベントスクリプトで次の SQL 文を定義します。

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

これで unique_device_ID デバイスがサーバーに対して Push 通知をポーリングし、request_cursor イベントが PushRequest テーブルでデータを検出すると、Push 要求が生成されます。デバイスに送信されると、Push 通知の件名は **synchronize** と定義され、内容は **ASAP** と定義されます。

Push 要求の制限事項

次の表は、Push 要求の制限事項をカラムごとに示します。

カラム	型	制限
Request ID	INTEGER	この値は、ユニークなプライマリキーにしてください。
Poll key	VARCHAR	ライトウェイトポーラーの使用時にのみ必要です。 ポーリングキーには制限がありません。
Gateway	VARCHAR	ゲートウェイを使用する場合にのみ必要です。 この値には、有効なゲートウェイの名前を設定してください。独自のカスタムゲートウェイ名を指定するか、または次の事前に設定されたゲートウェイ名のいずれかを選択します。 ● **Default-DeviceTracker** ● **Default-SMTP** ● **Default-SYNC** ● **Default-UDP** 「ライトウェイトポーラーの代替としてのゲートウェイ」 22 ページを参照してください。
Subject	VARCHAR	この値の設定では、英数字以外の文字を使用しないでください。中カッコ、カレット、二重引用符、一重引用符、角カッコは内部用に予約されているため、subject カラムで使用しないでください。
Content	VARCHAR	メッセージの内容には制限がありません。

カラム	型	制限
Address	VARCHAR	ゲートウェイを使用する場合にのみ必要です。 UDP ゲートウェイの場合、この値は IP アドレスまたはホスト名にしてください。次のフォーマットのポート番号のサフィックスがサポートされています。 ● *IP-address*:*port-number* ● *hostname*:*port-number* SMTP ゲートウェイの場合、この値は電子メールアドレスにしてください。 SYNC ゲートウェイとデバイストラッキングゲートウェイの場合、この値は Mobile Link Listener -t+ オプションで定義されている受信者名にしてください。 「-t dblsn オプション」67 ページを参照してください。
Resend interval	VARCHAR	デフォルトでは、この値は分単位で測定されます。秒、分、時間それぞれの単位として **S**、**M**、**H** を指定できます。また、単位を組み合わせることもできます。たとえば、1H 30M 10S は、メッセージを 1 時間ごと、30 分ごと、10 秒ごとに再送するよう Notifier に通知します。 この値が NULL または未指定の場合、デフォルトでは一度だけ送信され、再送は行われません。
Time to live	VARCHAR	デフォルトでは、この値は分単位で測定されます。秒、分、時間それぞれの単位として **S**、**M**、**H** を指定できます。また、単位を組み合わせることもできます。たとえば、3H 30M 10S は、メッセージの最初の送信の 3 時間後、30 分後、10 秒後に再送を停止するよう Notifier に通知します。 この値が NULL または未指定の場合、デフォルトでは一度だけ送信され、再送は行われません。

Push 要求の検出と Push 通知の送信

Notifier では、request_cursor イベントを頻繁に起動することによって、Push 要求を検出します。デフォルトでは、このイベントにはスクリプトが指定されていません。Notifier が Push 要求を検出できるよう、request_cursor イベントを指定してください。典型的なアプリケーションでは、request_cursor イベントスクリプトは SELECT 文です。 「request_cursor イベント」39 ページを参照してください。

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

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

注意
カラムは、Push 要求で指定されている順序と同じ順序で選択してください。 「Push 要求の要件」7 ページを参照してください。

Notifier イベントの設定については、「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

Push 要求の削除

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

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

次の項を参照してください。

- 「Push 要求の要件」7 ページ
- 「request_delete イベント」40 ページ
- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ

参照

- 「チュートリアル：ライトウェイトポーリングを使用したサーバー起動同期の設定」99 ページ
- 「チュートリアル：ゲートウェイを使用したサーバー起動同期の設定」110 ページ

Notifier

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

Mobile Link サーバーを最初にロードすると、Notifier が起動します。Mobile Link サーバーの単一インスタンス内で複数の Notifier を実行できます。複数の Notifier の使用方法の例については、*%SQLANYSAMP12%¥MobiLink¥SIS_MultipleNotifier* にあるサンプルアプリケーションを参照してください。

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

Mobile Link サーバーファームでの Notifier

11.0 以前の Mobile Link では、Mobile Link サーバーファームでサーバー起動同期を使用すると、冗長な Push 通知が発生し、追加の同期が行われ、Mobile Link サーバーファームの統合データベースに負荷がかかることがありました。現行バージョンでは、ファーム内のすべての Mobile Link サーバーで Notifier を実行できるようになりました。これらの Notifier によって、同じ Mobile Link Listener への冗長な Push 通知がなくなります。ローカルの Mobile Link サーバーに接続する必要があるときは、mlsrv12 -lsc サーバーオプションを使用して、他のサーバーに情報を渡すことができます。 「-lsc mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。

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

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

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

この機能を使用するには、ファーム内のすべての Mobile Link サーバーに次の mlsrv12 コマンドラインオプションが必要です。

- 「-lsc mlsrv12 オプション」『Mobile Link サーバー管理』
- 「-notifier mlsrv12 オプション」『Mobile Link サーバー管理』
- 「-zs mlsrv12 オプション」『Mobile Link サーバー管理』

例

host001 の場合：

```
mlsrv12 -notifier -zs ml001 -lsc tcpip(host=host001;port=2439) ...
```

host007 の場合：

```
mlsrv12 -notifier -zs ml007 -lsc tcpip(host=host007;port=2439) ...
```

Notifier イベントとプロパテの設定

Notifier イベントを使用することにより、サーバー起動同期処理全体を管理するスクリプトを埋め込むことができます。 Notifier イベントの流れとその起動方法の詳細については、「Notifier イベント」35 ページを参照してください。

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

●request_cursor イベントを使用して、Push 要求で送信する情報、送信方法、送信先を決定する。

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

Notifier プロパティはイベントに似ています。イベントは通知プロセスを管理しますが、プロパティは Notifier の動作を管理します。たとえば、Notifier プロパティでは、Notifier が統合データベースをポーリングする頻度や起動時に Notifier を有効にするかどうかを決定します。Notifier プロパティおよびイベントは、サーバー側の設定として設定します。

参照

- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ

Notifier の起動

Mobile Link サーバーをロードすると、有効な Notifier がすべて起動します。Notifier を無効にするには、有効な Notifier のプロパティ値を false に設定してください。 「Notifier プロパティ」46 ページを参照してください。

Notifier を起動するには、次のいずれかの方法を使用します。

- Notifier を設定し、指定した -notifier オプションで mlsrv12 を実行する。
- 設定が Notifier 設定ファイルに格納されている場合は、コマンドラインで mlsrv12 を実行してデータベースをロードし、-notifier オプションを使用してファイルを指定する。たとえば、ファイル *myfirst.Notifier* を使用する場合は、次のコマンドによって、このファイルに指定されているプロパティおよびイベントを使用するよう Mobile Link サーバーを設定します。

  ```
  mlsrv12 ... -notifier c:¥myfirst.Notifier
  ```

  QAnywhere を使用している場合は、コマンドラインで、指定した -m オプションとともに mlsrv12 を実行してデータベースをロードします。

参照

- 「-notifier mlsrv12 オプション」『Mobile Link サーバー管理』
- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ
- 「Notifier イベントとプロパテの設定」13 ページ
- 「-m mlsrv12 オプション」『Mobile Link サーバー管理』

Listener

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

参照

- 「デバイストラッキングゲートウェイ」23 ページ
- 「メッセージハンドラー」15 ページ
- 「Windows デバイス用の Mobile Link Listener ユーティリティ (dblsn)」55 ページ

例

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

```
dblsn -v2 -m -ot dblsn.log
   -l "poll_connect='host=localhost';
   poll_notifier=notifier_name1;
   poll_key=sis_user1;
   poll_every=10;
   subject=sync;
   action='start dbmlsync.exe
      -c SERVER=rem1;UID=DBA;PWD=sql
      -ot dbmlsyncOut.txt -qc';"
```

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

Windows 用の Mobile Link Listener のコマンドラインオプションの詳細については、「Windows デバイス用の Mobile Link Listener オプション」57 ページを参照してください。

メッセージハンドラー

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

メッセージハンドラーは、次のコンポーネントから構成されています。

- **フィルターのキーワード**　Push 通知が前処理されると、フィルターのキーワードを使用してメッセージの内容をスキャンできます。フィルター条件が満たされると、アクションが開始されます。たとえば、**subject** キーワードを指定して特定の件名を含むメッセージをフィルターしたり、**sender** キーワードを指定して特定の Mobile Link サーバーから受信したメッセージをフィルターしたりできます。

- **アクション**　メッセージでフィルター条件が満たされると、アクションが開始されます。典型的なアプリケーションでは、アクションを指定して同期を開始しますが、別の操作も実行

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

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

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

参照

- 「-l dblsn オプション」62 ページ
- 「Windows デバイス用の Mobile Link Listener キーワード」70 ページ
- 「Windows デバイス用の Mobile Link Listener アクションコマンド」72 ページ

メッセージハンドラーの使用

Mobile Link Listener が Push 通知を受信すると、Push 通知はメッセージを抽出します。メッセージは複数のキーワードに分割されています。**message** キーワードには、メッセージ全体が未加工形式で記述されています。このメッセージは、**subject** キーワード、**content** キーワード、**sender** キーワードに分割されます。これらのキーワードは、メッセージフィルターを介して実行され、開始するアクションを決定します。 これらのキーワードを使用したメッセージのフィルタリングの詳細については、「メッセージのフィルタリング」16 ページを参照してください。

メッセージのフィルタリング

フィルターキーワードを使用して、Push 通知の一部とユーザー定義のフレーズを比較します。2 つのフレーズのテキストが同等の場合、アクションが開始されます。 メッセージフィルタリング用の Push 通知の前処理については、「メッセージ構文」95 ページを参照してください。

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

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

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

次の各キーワードは、メッセージハンドラーに 1 回のみ表示されます。

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

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


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

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

- **message**　このキーワードは、未加工データに基づいてメッセージをフィルタリングするために使用します。フィルター値がメッセージの正確な長さと一致するようにしてください。このキーワードには変数構造があるため、使用しないことをおすすめします。 メッセージフィルタリング用の Push 通知の前処理については、「メッセージ構文」95 ページを参照してください。

- **message_start**　このキーワードは、未加工データの先頭からの一部に基づいてメッセージをフィルタリングするために使用します。 メッセージフィルタリング用の Push 通知の前処理については、「メッセージ構文」95 ページを参照してください。

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

- **sender**　このキーワードは、送信者に基づいてメッセージをフィルタリングするために使用します。このキーワードは、特定の Notifier が送信した Push 通知を追跡するのに役立ちます。この値は、使用されているゲートウェイによって異なります。UDP ゲートウェイの場合、この値はゲートウェイのホストの IP アドレスです。SYNC ゲートウェイの場合は、**MobiLink** です。また、SMTP ゲートウェイの場合は、ご使用の無線通信事業者によって異なります。 「ゲートウェイと Carrier」22 ページを参照してください。

参照

- 「Windows デバイス用の Mobile Link Listener キーワード」70 ページ
- 「action 変数」18 ページ

アクションの開始

メッセージがフィルターの条件を満たしている場合に、アクションが開始されます。 Push 通知のフィルタリングの詳細については、「メッセージのフィルタリング」16 ページを参照してください。

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

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

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

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

- **RUN**　アプリケーションを実行し、追加の Push 通知を受信する前にアプリケーションの完了を待機します。

- **POST**　すでに実行中のプロセスにウィンドウメッセージを送信する。このコマンドは、Windows デバイスでしか使用できません。
- **SOCKET**　TCP/IP 接続を使用して、アプリケーションにメッセージを送信します。
- **DBLSN FULL SHUTDOWN**　Mobile Link Listener を停止します。

参照

- 「Windows デバイス用の Mobile Link Listener アクションコマンド」72 ページ
- 「action 変数」18 ページ

action 変数

action 変数を使用すると、メッセージフィルターまたはアクションからの Push 通知の一部を参照できます。「アクションの開始」17 ページと「メッセージのフィルタリング」16 ページを参照してください。

action 変数の設定方法

ほとんどの action 変数は、Push 通知が受信されるたびに自動的に設定されます。変数名は、メッセージ構文で指定されている名前と似ています。たとえば、*message* は $message action 変数を、*subject* は $subject action 変数を、*sender* は $sender action 変数を、*content* は $content action 変数をそれぞれ設定します。「メッセージ構文」95 ページを参照してください。

action 変数の使用

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

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

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

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

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

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

参照

- 「Windows デバイス用の Mobile Link Listener アクションコマンド」72 ページ
- 「Windows デバイス用の Mobile Link Listener action 変数」76 ページ
- 「リモート ID によるメッセージのフィルタリング」19 ページ

ライトウェイトポーリングオプションの設定

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

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

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

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

- **poll_connect**　このオプションは、サーバーへの接続に必要なプロトコルオプションの指定に使用します。または、dblsn -x オプションを使用してデフォルトのプロトコルオプションを指定することもできます。poll_connect オプションを使用すると、メッセージハンドラーのデフォルトのプロトコルオプションが上書きされます。

- **poll_notifier**　このオプションは、Push 要求を処理するために Mobile Link サーバーで使用される Notifier の指定に使用します。Mobile Link サーバーでは複数の Notifier を受け入れることができるため、このオプションが必要になります。

- **poll_key**　このオプションは、Notifier に対する Mobile Link Listener の識別に使用します。Mobile Link サーバーはこの値を使用して、デバイスを対象とした Push 通知を送信します。典型的なアプリケーションでは、この値をデバイスのリモート ID にしてください。

- **poll_every**　このオプションは、Mobile Link Listener が Notifier をポーリングする頻度の指定に使用します。デフォルトでは、Mobile Link Listener は Mobile Link サーバーから自動的にこの値を取得します。この値は、秒単位です。

参照

- 「Push 要求の要件」7 ページ
- 「Notifier イベントとプロパテの設定」13 ページ
- 「ライトウェイトポーラー」21 ページ
- 「Windows デバイス用の Mobile Link Listener キーワード」70 ページ
- 「ライトウェイトポーリング API」81 ページ

メッセージハンドラーの高度な機能

リモート ID によるメッセージのフィルタリング

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

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

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

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

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

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

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

参照

- 「action 変数」 18 ページ
- 「リモート ID」『Mobile Link クライアント管理』
- 「-r dblsn オプション」 66 ページ
- 「Windows デバイス用の Mobile Link Listener action 変数」 76 ページ

接続起動同期

Windows デバイスでは、接続の変更時に同期を開始できます。

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

接続におけるすべての変更の識別

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


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

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

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

Mobile Link サーバーへの最適パスの変更の識別

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

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

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

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

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

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

参照

- 「Windows デバイス用の Mobile Link Listener ユーティリティ (dblsn)」55 ページ
- 「Windows デバイス用の Mobile Link Listener アクションコマンド」72 ページ
- 「Windows デバイス用の Mobile Link Listener action 変数」76 ページ

ライトウェイトポーラー

ライトウェイトポーラーは、指定された時間間隔で Push 通知をポーリングするデバイスアプリケーションです。ゲートウェイを設定する代わりにライトウェイトポーラーを使用できます。SYNC ゲートウェイとは異なりサーバーへの永続的接続を必要とせず、また、UDP ゲートウェイとは異なり連続的な接続を必要としないため、ライトウェイトポーラーの使用をおすすめします。

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

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

注意
Sybase Central を使用してリモートデータベースを管理し、その後、サーバー起動リモートタスク (SIRT) を使用して Push 通知を実装できます。 詳細については、「リモートデータベースの集中管理」『Mobile Link サーバー管理』と「サーバー起動リモートタスク (SIRT)」『Mobile Link サーバー管理』を参照してください。

参照

- 「ライトウェイトポーリング API」 81 ページ
- 「ライトウェイトポーリングオプションの設定」 19 ページ
- 「Windows デバイス用の Mobile Link Listener キーワード」 70 ページ

ゲートウェイと Carrier

ライトウェイトポーラーの代替としてのゲートウェイ

ゲートウェイは、Mobile Link システムテーブルまたは Notifier プロパティファイルに保存される Mobile Link オブジェクトで、システム起動同期用のメッセージの送信方法に関する情報が含まれます。ライトウェイトポーラーの代わりに使用でき、継続したネットワーク接続を必要とします。

ゲートウェイのプロパティは、Mobile Link サーバーで設定します。1 つの Mobile Link サーバーに複数のゲートウェイを設定できます。

サポートされているゲートウェイ

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

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

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

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

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

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

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

デバイストラッキングゲートウェイ

サポートされているゲートウェイ以外に、Push 通知を送信する最適なゲートウェイを自動的に選択するデバイストラッキングゲートウェイを設定できます。デフォルトのデバイストラッキングゲートウェイは、Default-DeviceTracker です。ライトウェイトポーラーを使用しない場合は、このゲートウェイを使用することをおすすめします。

参照

- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ
- 「SYNC ゲートウェイプロパティ」51 ページ
- 「UDP ゲートウェイプロパティ」52 ページ
- 「SMTP ゲートウェイプロパティ」50 ページ
- 「デバイストラッキングゲートウェイ」23 ページ

デバイストラッキングゲートウェイ

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

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

9.0.1 以降のほとんどの Mobile Link 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ない Mobile Link Listener を使用している場合、トラッキング情報を提供することで、デバイストラッキングゲートウェイを使用することもできます。

参照

- 「デバイストラッキングのサポート」24 ページ
- 「ライトウェイトポーラーの代替としてのゲートウェイ」22 ページ
- 「Carrier と Carrier 設定」27 ページ

デバイストラッキングのサポート

注意
SQL Anywhere 9.0.0 以前で実行している Mobile Link Listener を使用している場合にのみ、デバイストラッキングがサポートされている必要があります。その他すべての Windows デバイス用 Mobile Link Listener では、デバイストラッキングがサポートされています。

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

前提条件

この作業を実行するための前提条件は、ありません。

内容と備考

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

UDP ゲートウェイでは、再接続のたびに IP アドレスが変更される場合、静的エントリに依存することはできません。この問題を解決するには、IP アドレスではなくホスト名をアドレス指定します。ただし、このソリューションでは、DNS サーバーテーブルの更新速度が低下するため、Push 通知が誤配信される可能性があります。システムプロシージャーを設定して、システムテーブルをプログラムによって更新することもできます。

♦ 9.0.0 Mobile Link Listener のデバイストラッキングの手動での設定

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

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

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

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

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

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

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

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

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

結果

指定したデバイスがデバイストラッキングを行うよう設定されます。

次の手順

なし。

参照

- 「ml_set_device システムプロシージャー」 89 ページ
- 「ml_set_listening システムプロシージャー」 91 ページ
- 「ml_set_device_address システムプロシージャー」 90 ページ
- 「デバイストラッキングゲートウェイ」 23 ページ
- 「Carrier プロパティ」 53 ページ

デバイストラッキングゲートウェイ設定のクイックスタート

♦ デバイストラッキングの設定

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

   Mobile Link サーバーを起動すると、これらのゲートウェイがデフォルト設定を使用して設定されています。 プロパティの設定や独自のゲートウェイの作成の詳細については、「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

   注意
   SMTP ゲートウェイの場合、Carrier の設定が必要です。 「Carrier と Carrier 設定」27 ページを参照してください。

2. 次の条件に従って新しい Notifier を作成し、request_cursor イベントを設定します。

   - ゲートウェイ名は、使用するデバイストラッキングゲートウェイの名前にしてください。デフォルトのゲートウェイ名は、Default-DeviceTracker です。この名前は、結果セットの最初のカラムで指定されています。
   - アドレス名には、デバイスのリモート ID を設定してください。dblsn -t+ オプションを使用して、Mobile Link サーバーにリモート ID を登録します。この名前は、結果セットの 4 番目のカラムで指定されています。

   request_cursor イベントの設定の詳細については、「request_cursor イベント」39 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 Mobile Link Listener 名を追加します。

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

   Mobile Link Listener を実行して、Mobile Link Listener メッセージウィンドウ内のデバイス名を確認します。または、dblsn -e オプションを使用してデバイス名を設定するか、dblsn -u オプションを使用して別の Mobile Link Listener 名を設定できます。 「-e dblsn オプション」61 ページと「-u dblsn オプション」67 ページを参照してください。

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

4. 必要なオプションを指定して Mobile Link Listener を起動します。 Mobile Link Listener の起動の詳細については、「Windows デバイス用の Mobile Link Listener ユーティリティ (dblsn)」55 ページを参照してください。

Carrier と Carrier 設定

Carrier は、Mobile Link システムテーブルまたは Notifier プロパティファイルに保存される Mobile Link オブジェクトで、サーバー起動同期で使用される通信業者に関する情報が含まれます。

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

ネットワークプロバイダー ID や SMS 電子メールのプレフィクスなど、Carrier のプロパティは、Mobile Link サーバーで設定します。複数の Carrier サービスに対応するには、Mobile Link サーバーで複数の Carrier を設定します。

送信者の構文

Push 通知は、Mobile Link Listener で受信され、メッセージフィルタリング用に前処理されると、複数のキーワードに分割されます。**message** の **sender** キーワードは電子メールアドレスです。この電子メールアドレスは、デバイスで生成され、無線通信事業者によって異なります。 メッセージの前処理の詳細については、「メッセージ構文」95 ページを参照してください。

sender 構文は、次のフォーマットになります。

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

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

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

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

参照

- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ
- 「Carrier プロパティ」53 ページ

サーバー起動同期の Mobile Link サーバー設定

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

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

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

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

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

ml_add_property システムプロシージャーを使用したサーバー側の設定

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

注意
Notifier、ゲートウェイ、Carrier に名前を付ける場合は、ANSI 標準を使用してください。

共通プロパティ構文

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

Notifier プロパティとイベントの構文

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

ゲートウェイプロパティ構文

```
CALL ml_add_property('SIS', 'DeviceTracker(DeviceTrackerName)', 'Property', Value);
CALL ml_add_property('SIS', 'SMTP(SMTPName)', 'Property', Value);
CALL ml_add_property('SIS', 'UDP(UDPName)', 'Property', Value);
CALL ml_add_property('SIS', 'SYNC(SYNCName)', 'Property', Value);
```

Carrier プロパティ構文

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

参照

- 「ml_add_property システムプロシージャー」『Mobile Link サーバー管理』

Sybase Central を使用した Notifier、ゲートウェイ、Carrier の設定

Sybase Central には、プロパティとイベントを変更するためのグラフィカルユーザーインターフェイスが用意されています。Sybase Central を使用すると、複数の Notifier、ゲートウェイ、Carrier を設定できます。

前提条件

この作業を実行するための前提条件は、ありません。

内容と備考

Notifier、ゲートウェイ、Carrier に名前を付ける場合は、ANSI 標準を使用してください。

Sybase Central を使用してサーバー側設定を実行すると、mlsrv12 -notifier オプションを使用するときに、コマンドラインで Notifier 設定ファイルを指定する必要がありません。

♦ Sybase Central を使用した Notifier、ゲートウェイ、Carrier の設定

1. 統合データベース用に Mobile Link プロジェクトをまだ作成していない場合は、Mobile Link プラグインを使用して作成します。

   「Mobile Link プロジェクトの作成」『Mobile Link クイックスタート』を参照してください。

2. **[表示]** » **[フォルダー]** をクリックします。

3. Sybase Central の左ウィンドウ枠で、**[Mobile Link 12]**、Mobile Link プロジェクト名、**[統合データベース]**、統合データベース名の順に展開し、**[通知]** を選択します。

   右ウィンドウ枠に、使用可能な Notifier、ゲートウェイ、Carrier がすべて表示されます。

4. 新しい Notifier、ゲートウェイ、Carrier を作成します。

   - 新しい Notifier を作成するには、右ウィンドウ枠の **[Notifier]** タブをクリックし、**[ファイル]** - **[新規]** - **[Notifier]** を選択します。
   - 新しいゲートウェイを作成するには、右ウィンドウ枠の **[ゲートウェイ]** タブをクリックし、**[ファイル]** - **[新規]** - **[ゲートウェイ]** をクリックします。
   - 新しい Carrier を作成するには、右ウィンドウ枠の **[Carrier]** タブをクリックし、**[ファイル]** - **[新規]** - **[Carrier]** をクリックします。

5. 設定する Notifier、ゲートウェイ、または Carrier を選択します。

   ●Notifier プロパティまたはイベントを設定するには、右ウィンドウ枠の **[Notifier]** タブをクリックし、設定する Notifier を選択します。

   ●ゲートウェイプロパティを設定するには、右ウィンドウ枠の **[ゲートウェイ]** タブをクリックし、設定するゲートウェイを選択します。

   ●Carrier プロパティを設定するには、右ウィンドウ枠の **[Carrier]** タブをクリックし、設定する Carrier を選択します。

   [ファイル] - **[プロパティ]** をクリックします。

   選択した Notifier、ゲートウェイ、または Carrier に適用可能なすべての設定を調整できるウィンドウが表示されます。

6. **[OK]** をクリックします。

結果

Notifier、ゲートウェイまたは Carrier が設定され、使用できるようになります。

次の手順

なし。

Notifier 設定ファイルからのサーバー側設定のインポート

サーバー側設定を ml_property_table にインポートするには、Notifier 設定ファイルを使用します。

前提条件

この作業を実行するための前提条件は、ありません。

内容と備考

次のとおりです。

♦ Notifier 設定ファイルからのサーバー側設定のインポート

1. 統合データベース用に Mobile Link プロジェクトをまだ作成していない場合は、Mobile Link プラグインを使用して作成します。

   「Mobile Link プロジェクトの作成」『Mobile Link クイックスタート』を参照してください。

2. **[表示]** » **[フォルダー]** をクリックします。

3. Sybase Central の左ウィンドウ枠で、**[Mobile Link 12]**、Mobile Link プロジェクト名、**[統合データベース]**、統合データベース名の順に展開し、**[通知]** を選択します。

4. **[ファイル]** » **[インポートの設定]** をクリックし、ウィザードの指示に従います。

結果

設定が、Notifier 設定ファイルから ml_property_table にインポートされます。

次の手順

なし。

Notifier 設定ファイルへのサーバー側設定のエクスポート

サーバー側設定は、ml_property テーブルから Notifier 設定ファイルにエクスポートできます。設定をエクスポートすると、複数のバージョンのサーバー側設定を作成し、mlsrv12 -notifier オプションを使用して異なるバージョンをロードできます。

前提条件

この作業を実行するための前提条件は、ありません。

内容と備考

次のとおりです。

♦ Notifier 設定ファイルへのサーバー側設定のエクスポート

1. 統合データベース用に Mobile Link プロジェクトをまだ作成していない場合は、Mobile Link プラグインを使用して作成します。

   「Mobile Link プロジェクトの作成」『Mobile Link クイックスタート』を参照してください。

2. **[表示]** » **[フォルダー]** をクリックします。
3. Sybase Central の左ウィンドウ枠で、**[Mobile Link 12]**、Mobile Link プロジェクト名、**[統合データベース]**、統合データベース名の順に展開し、**[通知]** を選択します。
4. **[ファイル]** » **[設定のエクスポート]** をクリックし、ウィザードの指示に従います。

結果

指定した設定が Notifier 設定ファイルへエクスポートされます。

次の手順

なし。

参照

- 「同期モデルでのサーバー起動同期の設定」『Mobile Link クイックスタート』

Notifier 設定ファイルを使用したサーバー側の設定

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

注意
Notifier、ゲートウェイ、Carrier に名前を付ける場合は、ANSI 標準を使用してください。

Notifier 設定ファイルの作成と設定

Notifier 設定ファイルは、テキストエディターを使用して作成したり、Sybase Central からエクスポートしたプロパティとイベントの設定から生成したりできます。 「Sybase Central を使用した Notifier、ゲートウェイ、Carrier の設定」30 ページを参照してください。

一般的な Notifier 設定ファイルのレイアウトを参照するには、*%SQLANYSAMP12%¥MobiLink ¥template.Notifier* テンプレートファイルを開きます。このテンプレートファイルでは、サーバー側プロパティとイベントを設定するための例を提供します。

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

共通プロパティ構文

```
Property = Value
```

Notifier イベント構文

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

Notifier プロパティ構文

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

ゲートウェイプロパティ構文

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

Carrier プロパティ構文

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

Notifier 設定ファイルのロード

Notifier 設定ファイルを Mobile Link サーバーにロードするには、コマンドラインから -notifier オプションを指定して mlsrv12 を実行します。たとえば、CarDealer.Notifier 設定ファイルに定義されたサーバー側設定を使用するには、次のコマンドを実行します。

```
mlsrv12 ... -notifier "c:¥CarDealer.Notifier"
```

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

mlsrv12 の -notifier オプションの詳細については、「-notifier mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。

注意
デフォルトの SYNC ゲートウェイを使用する場合、サーバー側設定を Notifier 設定ファイルには保存できません。別の方法を使用して、この設定を ml_property システムテーブルに格納する必要があります。 「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

エスケープシーケンスの使用

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

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

Unicode のエスケープシーケンス形式は ¥u*XXXX*、ASCII のエスケープシーケンス形式は ¥x*XX* です。ここで、各 *X* は 16 進数字を表します。

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

Notifier イベント

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

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

特に指定しないかぎり、Notifier イベントはおすすめする方法のいずれかを使用して設定できます。 Notifier イベントの設定の詳細については、「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

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

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

ポーリング中のイベント

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

begin_poll イベント

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

参照

- 「Push 要求」7 ページ
- 「Notifier イベント」35 ページ
- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ

例

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

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

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

end_poll イベント

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

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

参照

- 「Notifier イベント」35 ページ
- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ

error_handler イベント

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

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

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

注意
このイベントにはシステムプロシージャーの使用が必要です。Sybase Central を使用して、このイベントを直接設定することはできません。 「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

参照

- 「Notifier イベント」35 ページ
- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ

例

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

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

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

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

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

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

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

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

mlsrv12 -notifier オプションを使用してファイルを実行します。 Notifier 設定ファイルを設定する方法の詳細については、「Notifier 設定ファイルを使用したサーバー側の設定」33 ページを参照してください。

request_cursor イベント

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

ライトウェイトポーラーを使用する場合の Push 要求のフェッチ (推奨)

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

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

- ポーリングキー
- 件名 (オプション)
- 内容 (オプション)

ゲートウェイを使用する場合の Push 通知のフェッチ

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

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

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

参照

- 「Push 要求の要件」7 ページ
- 「Notifier イベント」35 ページ
- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ

例

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

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

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

request_delete イベント

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

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

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

request_delete イベントを使用すると、有効期限が切れた要求または暗黙的に除外された要求を削除できなくなります。たとえば、*%SQLANYSAMP12%¥MobiLink¥SIS_CarDealer* ディレクトリの CarDealer サンプルでは、request_delete イベントを使用して、PushRequest テーブルのステータスフィールドを 'processed' に設定しています。

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

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

参照

- 「Notifier イベント」35 ページ
- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ

shutdown_query イベント

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

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

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

参照

- 「end_connection イベント」42 ページ
- 「Notifier イベント」35 ページ
- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ

例

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

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

接続イベント

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

begin_connection イベント

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

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

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

参照

- 「Notifier プロパティ」46 ページ
- 「Notifier イベント」35 ページ
- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ

end_connection イベント

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

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

参照

- 「Notifier イベント」35 ページ
- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ

非同期イベント

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

confirmation_handler イベント

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

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

注意
Mobile Link Listener が配信確認情報をアップロードできるようにするには、dblsn -x オプションを使用します。配信確認は必要だが IP 追跡は不要な場合は、dblsn -ni オプションを使用します。 「Windows デバイス用の Mobile Link Listener オプション」57 ページを参照してください。

注意
このイベントにはシステムプロシージャーの使用が必要です。Sybase Central を使用する方法では、このイベントを直接設定できません。 「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

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

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

スクリプトパラメーター	データ型	説明
remote_code (in)	Integer	Mobile Link Listener からレポートされた概要です。次の値が返されます。 ● **1**：メッセージは受け入れられました。 ● **2**：メッセージは拒否されました。 ● **3**：メッセージは受け入れられ、アクションは正常に終了しました。 ● **4**：メッセージは受け入れられ、アクションは失敗しました。
remote_device (in)	varchar	応答 Mobile Link Listener のデバイス名です。
remote_mluser (in)	varchar	応答 Mobile Link Listener の Mobile Link ユーザー名です。
remote_action_return (in)	varchar	リモートアクションのリターンコードです。
remote_action (in)	varchar	アクションコマンド用に予約済みです。
gateway (in)	varchar	要求に関連付けられているゲートウェイです。
address (in)	varchar	要求に関連付けられているアドレスです。
subject (in)	varchar	要求に関連付けられている件名です。
content (in)	varchar	要求に関連付けられている内容です。

参照

- 「Notifier イベント」35 ページ
- 「サーバー起動同期の Mobile Link サーバー設定」29 ページ
- 「ゲートウェイプロパティ」48 ページ

例

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

```
CREATE TABLE CustomConfirmation(
   error_code   integer,
   request_id   integer,
```

```
    remote_code   integer,
    remote_device  varchar(128),
    remote_mluser  varchar(128),
    remote_action_return varchar(128),
    remote_action  varchar(128),
    gateway   varchar(255),
    address   varchar(255),
    subject   varchar(255),
    content   varchar(255),
    occurAt   timestamp not null default timestamp
);

CREATE PROCEDURE CustomConfirmationHandler(
    out @request_option integer,
    in @error_code  integer,
    in @request_id  integer,
    in @remote_code  integer,
    in @remote_device  varchar(128),
    in @remote_mluser  varchar(128),
    in @remote_action_return varchar(128),
    in @remote_action  varchar(128),
    in @gateway   varchar(255),
    in @address   varchar(255),
    in @subject   varchar(255),
    in @content   varchar(255)
)

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

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

```
call ml_add_property(
    'SIS',
    'Notifier(myNotifier)',
```

```
'confirmation_handler',
'call CustomConfirmation(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)');
```

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

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

mlsrv12 -notifier オプションを使用してファイルを実行します。 Notifier 設定ファイルを設定する方法の詳細については、「Notifier 設定ファイルを使用したサーバー側の設定」33 ページを参照してください。

共通プロパティ

共通プロパティは、Notifier、ゲートウェイ、Carrier で共有されます。共通プロパティは、すべてオプションです。 これらのプロパティの設定の詳細については、「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

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

Notifier プロパティ

Notifier プロパティを使用すると、Notifier の動作を変更できます。Notifier のプロパティは、すべてオプションです。 これらのプロパティの設定の詳細については、「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

プロパティ	値	説明
connect_string	*connection_string*	データベースへの接続に使用されるデフォルトの接続動作を上書きします。デフォルト値は **ianywhere.ml.script.ServerContext** です。この値では、mlsrv12 コマンドラインで指定された接続文字列を使用します。 別のデータベースに接続するときに通知ロジックとデータを同期データから分離するのに便利です。ほとんどの展開ではこのプロパティを設定しません。
enable	{ **yes** | **no** }	Notifier を有効にするかどうかを指定します。-notifier mlsrv12 オプションを実行すると、有効な Notifier がすべて起動します。
gui	{ **yes** | **no** }	Notifier の動作中に Notifier ウィンドウを表示するかどうかを指定します。デフォルト値は **yes** です。 この Notifier ウィンドウを使用すると、ポーリング間隔を一時的に変更したり、すぐにポーリングを実行したりできます。また、Mobile Link サーバーを停止せずに Notifier を停止するために使用することも可能です一度停止すると、Mobile Link サーバーを停止して再度起動しないと、Notifier を再度起動できません。
isolation	{ **0** | **1** | **2** | **3** }	Notifier のデータベース接続の独立性レベルを指定します。次の値を使用できます。 ● **0**：コミットされない読み出し ● **1**：コミットされた読み出し ● **2**：繰り返し可能読み出し ● **3**：直列化可能 デフォルト値は **1** です。レベルが高くなると競合が増加しますが、パフォーマンスが逆に低下することがあります。独立性レベルを **0** に設定すると、コミットされていないデータ (ロールバックする可能性のあるデータ) を読み出すことができます。

プロパティ	値	説明
poll_every	*number*{ **s** | **m** | **h** }	確認がタイムアウトになるまでに待機する時間を指定します。次に、使用可能な時間単位のリストを示します。 ● **s**：秒単位。 ● **m**：分単位。 ● **h**：時間単位。 デフォルト値は **1m** です。時間単位は、*HH***h** *MM***m** *SS***s** のフォーマットで組み合わせることができます。時間単位が指定されていない場合、時間は秒単位で測定されます。
shared_database_connection	{ **yes** | **no** }	Notifier がデータベース接続を共有するかどうかを指定します。デフォルト値は **no** です。Notifier が接続を共有できるのは、その独立性レベルが同じ場合のみです。 パフォーマンスに悪影響を出さずにリソースを節約するには、**yes** を指定します。状況によっては、接続を共有できないことがあります。たとえば、アプリケーションが Notifier 間でユニークでない SQL 変数名を使用している場合などが該当します。

ゲートウェイプロパティ

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

●Default-DeviceTracker ゲートウェイ

●Default-SYNC ゲートウェイ

●Default-UDP ゲートウェイ

●Default-SMTP ゲートウェイ

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

DefaultSYNC と DefaultUDP で定義されているプロパティを変更する必要はありませんが、DefaultSYNC ゲートウェイには、SMTP サーバー情報を指定する必要があります。デフォルトの

ゲートウェイを使用する必要がありますが、必要に応じて代替設定を使用できます。この項では、ゲートウェイプロパティをカスタマイズする手順について説明します。

デバイストラッキングゲートウェイプロパティ

デバイストラッキングゲートウェイプロパティを使用すると、デバイストラッキングゲートウェイの動作を変更できます。デバイストラッキングゲートウェイプロパティは、すべてオプションです。 これらのプロパティの設定の詳細については、「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

プロパティ	値	説明
confirm_action	{ **yes** | **no** }	確認が配信時にこのゲートウェイを通じて送信されるかどうかを指定します。デフォルト値は **no** です。
confirm_delivery	{ **yes** | **no** }	Mobile Link Listener がメッセージを受信する統合データベースを確認するかどうかを指定します。デフォルト値は **yes** です。Mobile Link Listener は、-x Mobile Link Listener オプションを指定して起動する必要があります。
description	*description_text*	ゲートウェイに関する説明です。
enable	{ **yes** | **no** }	デバイストラッキングゲートウェイを使用するかどうかを指定します。
smtp_gateway	*smtp_gateway_name*	SMTP 従属ゲートウェイの名前を指定します。デフォルト値は **DefaultSMTP** です。デバイストラッキングゲートウェイが使用できる SMTP ゲートウェイは、1 つのみです。このゲートウェイは有効にしておく必要があります。
sync_gateway	*sync_gateway_name*	SYNC 従属ゲートウェイの名前を指定します。デフォルト値は **DefaultSYNC** です。デバイストラッキングゲートウェイが使用できる SYNC ゲートウェイは、1 つのみです。このゲートウェイは有効にしておく必要があります。
udp_gateway	*udp_gateway_name*	UDP 従属ゲートウェイの名前を指定します。デフォルト値は **DefaultUDP** です。デバイストラッキングゲートウェイが使用できる UDP ゲートウェイは、1 つのみです。このゲートウェイは有効にしておく必要があります。

SMTP ゲートウェイプロパティ

SMTP ゲートウェイプロパティを使用すると、SMTP ゲートウェイの動作を変更できます。サーバーのプロパティは必須ですが、他の SMTP ゲートウェイプロパティは、すべてオプションです。 これらのプロパティの設定の詳細については、「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

プロパティ	値	説明
confirm_action	{ **yes** | **no** }	確認が配信時にこのゲートウェイを通じて送信されるかどうかを指定します。デフォルト値は **no** です。
confirm_delivery	{ **yes** | **no** }	このゲートウェイが配信を確認するかどうかを指定します。デフォルト値は **no** です。
confirm_timeout	*number*{ **s** | **m** | **h** }	確認がタイムアウトになるまでに待機する時間を指定します。次に、使用可能な時間単位のリストを示します。 ● **s**：秒単位。 ● **m**：分単位。 ● **h**：時間単位。 デフォルト値は **1m** です。時間単位は、*HH***h** *MM***m** *SS***s** のフォーマットで組み合わせることができます。時間単位が指定されていない場合、時間は秒単位で測定されます。
description	*description_text*	ゲートウェイに関する説明です。
enable	{ **yes** | **no** }	SYNC ゲートウェイを使用するかどうかを指定します。
Listeners_are_900	{ **yes** | **no** }	すべての Mobile Link Listener が SQL Anywhere 9.0.0 クライアントであるかどうかを指定します。デフォルト値は **no** です。SQL Anywhere 9.0.1 以降のクライアントについては、この値を **no** のままにします。
password	*password*	SMTP サービスのパスワードを指定します。一部のサービスでは必須です。
sender	*SMTP_address*	SMTP Push 通知の送信側アドレスを指定します。デフォルト値は **anonymous** です。

プロパティ	値	説明
server	*IP_address_or_hostname*	メッセージを Mobile Link Listener に送信するために使用する SMTP サーバーの IP アドレスまたはホスト名を指定します。デフォルト値は **mail** です。
user	*username*	SMTP サービスのユーザー名を指定します。一部のサービスでは必須です。

SYNC ゲートウェイプロパティ

SYNC ゲートウェイプロパティを使用すると、SYNC ゲートウェイの動作を変更できます。SYNC ゲートウェイプロパティは、すべてオプションです。 これらのプロパティの設定の詳細については、「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

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

プロパティ	値	説明
Listeners_are_900	{ **yes** | **no** }	すべての Mobile Link Listener が SQL Anywhere 9.0.0 クライアントであるかどうかを指定します。デフォルト値は **no** です。SQL Anywhere 9.0.1 以降のクライアントについては、この値を **no** のままにします。

UDP ゲートウェイプロパティ

UDP ゲートウェイプロパティを使用すると、IP アドレスやポート番号など、UDP ゲートウェイの動作を変更できます。UDP ゲートウェイプロパティは、すべてオプションです。 これらのプロパティの設定の詳細については、「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

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

プロパティ	値	説明
Listeners_are_900	{ **yes** | **no** }	すべての Mobile Link Listener が SQL Anywhere 9.0.0 クライアントであるかどうかを指定します。デフォルト値は **no** です。SQL Anywhere 9.0.1 以降のクライアントについては、この値を **no** のままにします。
Listener_port	*port_number*	リモートデバイスが UDP パケットの送信に使用するポートを指定します。デフォルト値は **5001** です。
sender	*IP_address_or_hostname*	マルチホームのホストの場合にのみ使用します。送信者の IP アドレスまたはホスト名を指定します。デフォルト値は **localhost** です。
sender_port	*port_number*	UDP パケットの送信に使用するポート番号を指定します。デフォルトでは、オペレーティングシステムによって空きポート番号がランダムに割り当てられます。

Carrier プロパティ

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

これらのプロパティの設定の詳細については、「サーバー起動同期の Mobile Link サーバー設定」29 ページを参照してください。

プロパティ	値	説明
enable	{ **yes** | **no** }	Carrier を使用するかどうかを指定します。
description	*description_text*	Carrier に関する説明です。
network_provider_id	*id_text*	ネットワークプロバイダー ID を指定します。Windows Mobile Phone Edition で SMS を使用するには、このプロパティを **_generic_** に設定します。
sms_email_domain	*domain_name*	Carrier のドメイン名を指定します。

プロパティ	値	説明
sms_email_user_prefix	*prefix_name*	電子メールアドレスで使用されるプレフィクスを指定します。

Windows デバイス用の Mobile Link Listener ユーティリティ (dblsn)

この項では、受信ライブラリ、Listener のオプションとキーワード、Listener のアクションコマンドと action 変数、Listener 設定など、Windows デバイス用の Mobile Link Listener ユーティリティについて説明します。

構文

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

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

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

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

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

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

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

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

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

参照

- 「Windows デバイス用の Mobile Link Listener オプション」57 ページ
- 「Windows デバイス用の Mobile Link Listener キーワード」70 ページ
- 「Windows デバイス用の Mobile Link Listener アクションコマンド」72 ページ
- 「Windows デバイス用の Mobile Link Listener action 変数」76 ページ

Windows デバイス用の受信ライブラリ

Mobile Link Listener には、UDP 受信ライブラリ *lsn_udp12.dll* が付属しており、デフォルトではこのライブラリがロードされます。

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

UDP (*lsn_udp12.dll*)

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

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

参照

- 「-d dblsn オプション」60 ページ
- 「-a dblsn オプション」60 ページ

Windows デバイス用の Mobile Link Listener オプション

次のオプションは、Mobile Link Listener の設定に使用できます。

オプション	説明
@{ *variable* | *filename* }	指定された環境変数またはテキストファイルからの Mobile Link Listener オプションを適用します。「@data dblsn オプション」 59 ページを参照してください。
-a *value*	受信ライブラリの 1 つのライブラリオプションを指定します。 「-a dblsn オプション」 60 ページを参照してください。
-d *filename*	受信ライブラリを指定します。「-d dblsn オプション」 60 ページを参照してください。
-e *device-name*	デバイス名を指定します。 「-e dblsn オプション」 61 ページを参照してください。
-f *string*	デバイスに関する追加の情報を指定します。 「-f dblsn オプション」 61 ページを参照してください。
-gi *seconds*	IP トラッカーのポーリング間隔を指定します。「-gi dblsn オプション」 61 ページを参照してください。
-i *seconds*	SMTP 接続のポーリング間隔を指定します。 「-i dblsn オプション」 62 ページを参照してください。
-l " *keyword=value;...* **"**	メッセージハンドラーの定義と作成を行います。「-l dblsn オプション」 62 ページを参照してください。
-m	メッセージのロギングを有効にします。「-m dblsn オプション」 63 ページを参照してください。
-ni	IP 追跡を無効にします。 「-ni dblsn オプション」 63 ページを参照してください。
-ns	SMS 受信を無効にします。 「-ns dblsn オプション」 63 ページを参照してください。

オプション	説明
-nu	UDP 受信を無効にします。 「-nu dblsn オプション」 63 ページを参照してください。
-o *filename*	ファイルに出力のログを取ります。 「-o dblsn オプション」 64 ページを参照してください。
-os *bytes*	ログファイルの最大サイズを指定します。 「-os dblsn オプション」 64 ページを参照してください。
-ot *filename*	ファイルをトランケートし、そのファイルに出力を記録します。 「-ot dblsn オプション」 64 ページを参照してください。
-p	アイドル状態になったときにデバイスを自動的に停止できます。 「-p dblsn オプション」 65 ページを参照してください。
-pc { + | - }	永続的な接続を有効または無効にします。 「-pc dblsn オプション」 65 ページを参照してください。
-q	Mobile Link Listener をクワイエットモードで実行します。 「-q dblsn オプション」 65 ページを参照してください。
-qi	dblsn アイコンとメッセージウィンドウを非表示にします。 「-qi dblsn オプション」 66 ページを参照してください。
-r *filename*	メッセージフィルターの応答アクションに関わるリモートデータベースを指定します。 「-r dblsn オプション」 66 ページを参照してください。
-sv *script-version*	認証に使用されるスクリプトバージョンを指定します。 「-sv dblsn オプション」 66 ページを参照してください。
-t {+ | - } *name*	リモートデータベースのリモート ID の登録または登録解除を行います。 「-t dblsn オプション」 67 ページを参照してください。
-u *username*	Mobile Link ユーザー名を指定します。 「-u dblsn オプション」 67 ページを参照してください。

オプション	説明
-v { **0** | **1** | **2** | **3** }	メッセージログの冗長性レベルを指定します。「-v dblsn オプション」68 ページを参照してください。
-w *password*	Mobile Link パスワードを指定します。「-w dblsn オプション」68 ページを参照してください。
-x { **http** | **https** | **tcpip** } [**(** *protocol-option=value*;... **)**]	ネットワークプロトコルと、Mobile Link サーバーのプロトコルオプションを指定します。「-x dblsn オプション」69 ページを参照してください。
-y *newpassword*	新しい Mobile Link パスワードを指定します。「-y dblsn オプション」69 ページを参照してください。

@data dblsn オプション

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

構文

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

備考

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

同じ名前を持つファイルと環境変数が存在する場合は、環境変数が使用されます。

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

参照

- 「設定ファイル」『SQL Anywhere サーバー データベース管理』
- 「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバー データベース管理』

例

サンプルのテキストファイルは、*%SQLANYSAMP12%¥MobiLink¥SIS_SimpleListener¥dblsn.txt* にあります。

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

```
dblsn @dblsnoptions
```

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

```
dblsn @mydblsn.txt
```

-a dblsn オプション

受信ライブラリの 1 つのライブラリオプションを指定します。

構文

dblsn -a *value* ...

備考

デフォルトでは、ライブラリを指定しない場合、Mobile Link Listener は *lsn_udp12.dll* を使用します。他のライブラリまたは追加のライブラリを指定するには、-d オプションを使用します。

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

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

参照

- 「Windows デバイス用の受信ライブラリ」 56 ページ
- 「-d dblsn オプション」 60 ページ

例

次の例では、ポートオプションを指定し、*lsn_udp12.dll* 受信ライブラリの ShowSenderPort オプションを宣言しています。

```
dblsn -d lsn_udp12.dll -a port=1234 -a ShowSenderPort
```

次の例では、2 つの異なるライブラリのポートオプションを指定しています。

```
dblsn -d lsn_udp12.dll -a port=1234 -d maac750.dll -a port=2345
```

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

```
dblsn -a ?
```

-d dblsn オプション

受信ライブラリを指定します。

構文

dblsn -d *filename* ...

備考

デフォルトでは、Mobile Link Listener は *lsn_udp12.dll* 受信ライブラリを使用します。

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

参照

- 「Windows デバイス用の受信ライブラリ」56 ページ

例

次の例では、*maac750.dll* 受信ライブラリを指定しています。

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

-e dblsn オプション

デバイス名を指定します。

構文

dblsn -e *device-name* ...

備考

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

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

デバイス名は Mobile Link Listener ウィンドウで参照できます。

-f dblsn オプション

デバイスに関する追加の情報を指定します。

構文

dblsn -f *string* ...

備考

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

-gi dblsn オプション

IP トラッカーのポーリング間隔を指定します。

構文

dblsn -gi *number* ...

備考

デフォルトでは、IP トラッカーは 60 秒ごとにポーリングします。

-i dblsn オプション

SMTP 接続のポーリング間隔を指定します。

構文

dblsn -i *number* ...

備考

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

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

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

参照

- 「-d dblsn オプション」 60 ページ

例

次の例では、2 つの異なるライブラリのポーリング間隔を指定しています。

```
dblsn -d lsn_udp12.dll -i 60 -d maac750.dll -i 45
```

-l dblsn オプション

メッセージハンドラーの定義と作成を行います。

構文

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

備考

指定可能なキーワードのリストについては、「Windows デバイス用の Mobile Link Listener キーワード」 70 ページを参照してください。

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

参照

- 「メッセージハンドラー」 15 ページ

-m dblsn オプション

メッセージのロギングを有効にします。

構文

dblsn -m ...

備考

デフォルトでは、メッセージのロギングはオフです。

-ni dblsn オプション

IP 追跡を無効にします。

構文

dblsn -ni ...

備考

デフォルトでは、IP 追跡は有効です。

このオプションでは、配信確認は停止しません。

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

参照

- 「-x dblsn オプション」 69 ページ

-ns dblsn オプション

SMS 受信を無効にします。

構文

dblsn -ns ...

備考

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

-nu dblsn オプション

UDP 受信を無効にします。

構文

dblsn -nu ...

備考

デフォルトでは、UDP 受信は有効です。

-o dblsn オプション

ファイルに出力のログを取ります。

構文

dblsn -o *filename* ...

備考

デフォルトでは、出力は Mobile Link Listener ウィンドウに表示されます。

参照

- 「-ot dblsn オプション」 64 ページ

-os dblsn オプション

ログファイルの最大サイズを指定します。

構文

dblsn -os *bytes* ...

備考

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

-ot dblsn オプション

ファイルをトランケートし、そのファイルに出力を記録します。

構文

dblsn -ot *filename* ...

備考

ファイルの内容が削除されてから、出力が記録されます。

参照

- 「-o dblsn オプション」 64 ページ

-p dblsn オプション

アイドル状態になったときにデバイスを自動的に停止できます。

構文

dblsn -p ...

備考

デフォルトでは、Mobile Link Listener がデバイスの停止を防止します。

このオプションが適用されるのは、Windows Mobile のみです。

-pc dblsn オプション

構文

永続的な接続を有効または無効にします。

dblsn -pc { + | - } ...

備考

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

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

永続的接続が切断された場合、Mobile Link Listener は継続的に再接続を試みます。

例

次の例では、永続的接続を無効にします。

```
dblsn -pc-
```

-q dblsn オプション

Mobile Link Listener をクワイエットモードで実行します。

構文

dblsn -q ...

備考

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

-qi dblsn オプション

dblsn アイコンとメッセージウィンドウを非表示にします。

構文

dblsn -qi ...

備考

このオプションでは、dblsn の実行が視覚的に確認できないようにします。起動時のエラーのウィンドウは開く場合はあります。-o で指定したログファイルを使用してエラーを診断できます。

参照

- 「-o dblsn オプション」64 ページ

-r dblsn オプション

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

構文

dblsn -r *filename* ...

備考

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

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

複数のデータベースを識別するには、-r オプションを複数回使用します。

参照

- 「メッセージハンドラーの使用」16 ページ
- 「Windows デバイス用の Mobile Link Listener アクションコマンド」72 ページ

-sv dblsn オプション

認証に使用されるスクリプトバージョンを指定します。

構文

dblsn -sv *script-version* ...

備考

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

-t dblsn オプション

リモートデータベースのリモート ID の登録または登録解除を行います。

構文

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

備考

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

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

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

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

参照

- 「Windows デバイス用の Mobile Link Listener アクションコマンド」72 ページ

-u dblsn オプション

Mobile Link Listener の Mobile Link ユーザー名を指定します。

構文

dblsn -u *username* ...

備考

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

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

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

参照

- 「-e dblsn オプション」61 ページ
- 「-w dblsn オプション」68 ページ
- 「Mobile Link ユーザーの作成と登録」『Mobile Link クライアント管理』

-v dblsn オプション

メッセージログの冗長性レベルを指定します。

構文

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

備考

デフォルトでは、冗長性レベルは 0 に設定されています。

次の表に、使用可能な冗長性レベル値の概要を示します。

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

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

参照

- 「-m dblsn オプション」63 ページ
- 「ファイルへのデータベースサーバーメッセージのロギング」『SQL Anywhere サーバー データベース管理』

-w dblsn オプション

Mobile Link パスワードを指定します。

構文

dblsn -w *password* ...

備考

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

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

参照

- 「-u dblsn オプション」 67 ページ

-x dblsn オプション

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

構文

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

備考

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

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

参照

- 「Mobile Link クライアントネットワークプロトコルオプション」『Mobile Link クライアント管理』

-y dblsn オプション

新しい Mobile Link パスワードを指定します。

構文

dblsn -y *newpassword* ...

備考

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

Windows デバイス用の Mobile Link Listener キーワード

次のキーワードを使用すると、dblsn -l オプションを使用して作成されたメッセージハンドラーを設定できます。 Mobile Link Listener キーワードの使用方法の詳細については、「-l dblsn オプション」62 ページを参照してください。

フィルターのキーワード

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

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

アクションのキーワード

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

キーワードの構文	説明
action= *command*	アクションコマンドを指定します。 「Windows デバイス用の Mobile Link Listener アクションコマンド」72 ページを参照してください。
altaction= *command*	アクションコマンドが失敗した場合に開始される代替アクションコマンドを指定します。 「Windows デバイス用の Mobile Link Listener アクションコマンド」72 ページを参照してください。

ポーリングのオプション

ライトウェイトポーラーを設定するには、次のオプションを使用します。

キーワードの構文	説明
poll_connect={ **http** | **https** | **tcpip** } [(*protocol-option*=*value*;...)]	サーバーへの接続に必要なライトウェイトネットワークプロトコルオプションを指定します。デフォルト値は、dblsn -x オプションから継承されます。「-x dblsn オプション」69 ページを参照してください。
poll_notifier= *Notifier-string*	Push 要求を処理する Notifier の名前を指定します。必須。
poll_key= *key-string*	Notifier に Mobile Link Listener 自体を示すための Listener の名前を指定します。この値は、ユニークにする必要があります。必須。
poll_every= *seconds-number*	Mobile Link Listener がサーバーをポーリングする頻度を指定します。間隔は秒単位で測定されます。デフォルト値は、Mobile Link サーバーから自動的に取得されます。

オプション

次のオプションを使用すると、メッセージハンドラーの動作を設定できます。

キーワードの構文	説明
continue=[**yes** | **no**]	最初の一致を検出した後に Mobile Link Listener が受信を継続するかどうかを指定します。デフォルト値は **no** です。**yes** 値を使用すると、複数のフィルターを指定するときに、1 つのメッセージによって複数のアクションが開始される場合に役立ちます。
confirm_action=[**yes** | **no**]	フィルターがアクションを確認するかどうかを指定します。デフォルト値は **yes** です。

キーワードの構文	説明
confirm_delivery=[yes | no]	フィルターが条件付きのメッセージ配信を確認するかどうかを指定します。デフォルト値は **yes** です。したがって、最初のフィルターがメッセージを受け入れたときに配信確認が送信されます。 メッセージの確認が必要で、フィルターがメッセージを受け入れた場合にのみ、配信を確認できます。指定したゲートウェイに、**yes** に設定された confirm_delivery キーワード値が定義されている場合は、メッセージに確認が必要です。**no** 値は、複数のフィルターが同一メッセージを受け入れる場合に、どのフィルターが配信確認をするかを細かく制御するために使用できます。 サーバーでの配信確認の処理の詳細については、「confirmation_handler イベント」42 ページを参照してください。
maydial=[yes | no]	アクションがモデムにダイヤル接続するパーミッションがあるかどうかを指定します。デフォルト値は **yes** です。**no** 値を指定すると、Mobile Link Listener はアクションの前にモデムを解放します。

参照

- 「メッセージハンドラー」15 ページ
- 「Windows デバイス用の Mobile Link Listener アクションコマンド」72 ページ

Windows デバイス用の Mobile Link Listener アクションコマンド

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

アクションコマンドのリストを次に示します。

コマンド	説明
START *program arglist*	バックグラウンドで Mobile Link Listener が実行されているときにプログラムを開始します。「START アクションコマンド」73 ページを参照してください。
RUN *program arglist*	Mobile Link Listener を一時停止してプログラムを実行します。「RUN アクションコマンド」74 ページを参照してください。
POST *windowmessage* | *id* to *windowclass* | *windowtitle*	ウィンドウクラスにウィンドウメッセージを送信します。「POST アクションコマンド」74 ページを参照してください。
SOCKET port=*windowname*[;**host=***hostname*] [;**sendText=***text*] [;**recvText=***text*[;**timeout=***seconds*]]	TCP/IP 接続を使用して、アプリケーションにメッセージを送信します。「SOCKET アクションコマンド」75 ページを参照してください。
DBLSN FULL SHUTDOWN	強制的に Mobile Link Listener をシャットダウンします。「DBLSN FULL SHUTDOWN アクションコマンド」76 ページを参照してください。

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

参照

- 「アクションの開始」17 ページ
- 「Windows デバイス用の Mobile Link Listener キーワード」70 ページ

START アクションコマンド

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

構文

action='START *program arglist*'

備考

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

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

例

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

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

RUN アクションコマンド

Mobile Link Listener を一時停止してプログラムを実行します。

構文

action='RUN *program arglist*'

備考

Mobile Link Listener は、プログラムの終了を待機してから受信を再開します。

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

例

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

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

POST アクションコマンド

ウィンドウクラスにウィンドウメッセージを送信します。

構文

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

備考

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

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

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

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

POST には次が有効です。

- **10 進 id による送信**　たとえば post 999 to <wc|wt> のように指定します。
- **16 進 id による送信**　たとえば post 0x3E7 to <wc|wt> のように指定します。
- **登録されたメッセージ名による送信**　たとえば post myRegisteredMsgName to <wc|wt> のように指定します。

例

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

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

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

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

参照

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

SOCKET アクションコマンド

TCP/IP 接続を使用して、アプリケーションにメッセージを送信します。

構文

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

備考

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

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

アプリケーションが sendText の受信に成功したことを確認するときにメッセージを表示するには、recvText を使用します。recvText を使用すると、タイムアウト制限を指定できます。Mobile

Link Listener が接続できない場合、受信確認を送信できない場合、またはタイムアウト制限内に確認を受信できない場合は、アクションの実行に失敗します。

例

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

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

DBLSN FULL SHUTDOWN アクションコマンド

強制的に Mobile Link Listener をシャットダウンします。

構文

action='DBLSN FULL SHUTDOWN'

備考

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

Windows デバイス用の Mobile Link Listener action 変数

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

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

変数	説明
$subject	メッセージの件名。
$content	メッセージの内容。
$message	件名、内容、送信者を含むメッセージ全体。

変数	説明
$message_start	message_start フィルターキーワードで指定された、メッセージの先頭の一部。この変数を使用できるのは、message_start フィルターキーワードを指定した場合のみです。 「Windows デバイス用の Mobile Link Listener キーワード」70 ページを参照してください。
$message_end	message_start フィルターキーワードで除外された、メッセージの一部。この変数を使用できるのは、message_start フィルターキーワードを指定した場合のみです。 「Windows デバイス用の Mobile Link Listener キーワード」70 ページを参照してください。
$ml_connect	dblsn -x オプションによって指定された Mobile Link ネットワークプロトコルオプション。デフォルトは、空の文字列です。 「-x dblsn オプション」69 ページを参照してください。
$ml_user	dblsn -u オプションによって指定された Mobile Link ユーザー名。デフォルトの名前は *device-name*-**dblsn** です。
$ml_password	dblsn -w オプションによって指定される Mobile Link パスワード、または -y を使用した場合は新しい Mobile Link パスワード。
$priority	この変数の意味は、carrier ライブラリに依存します。
$request_id	Push 要求で指定された要求 ID。 「Push 要求」7 ページを参照してください。
$remote_id	リモート ID です。この変数は、dblsn -r オプションが指定された場合にだけ使用されます。 「リモート ID によるメッセージのフィルタリング」19 ページを参照してください。
$sender	メッセージの送信者。
$type	この変数の意味は、carrier ライブラリに依存します。
$year	この変数の意味は、carrier ライブラリに依存します。
$month	この変数の意味は、carrier ライブラリに依存します。値は 1 ～ 12 までです。
$day	この変数の意味は、carrier ライブラリに依存します。値は 1 ～ 31 までです。
$hour	この変数の意味は、carrier ライブラリに依存します。値は 0 ～ 23 までです。
$minute	この変数の意味は、carrier ライブラリに依存します。値は 0 ～ 59 までです。

変数	説明
$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 -x オプションによって指定された Mobile Link サーバーに到達するための最善の NIC の MAC アドレス。最善のルートが NIC を経由しない場合、この変数の値は空文字列になります。
$best_adapter_name	dblsn -x オプションによって指定された Mobile Link サーバーに到達するための最善の NIC のアダプター名。最善のルートが NIC を経由しない場合、この変数の値は空文字列になります。
$best_ip	dblsn -x オプションによって指定された Mobile Link サーバーに到達するための最善の IP インターフェイスの IP アドレス。サーバーが到達不能な場合、この変数の値は 0.0.0.0 になります。
$best_network_name	dblsn -x オプションによって指定された Mobile Link サーバーに到達するための最善のプロファイルの RAS またはダイヤルアッププロファイル名。最善のルートが RAS またはダイヤルアップ接続を経由しない場合、この変数の値は空文字列になります。
$adapters	アクティブなネットワークアダプター名のリストで、それぞれパイプ記号 (|) で分割します。
$network_names	接続 RAS エントリ名のリストで、それぞれパイプ記号 (|) で分割します。RAS エントリ名は、ダイヤルアップネットワーク (DUN) のダイヤルアップエントリ名と呼ばれる場合もあります。
$poll_connect	poll_connect ポーリングキーワードによって指定された Mobile Link ネットワークプロトコルオプション。デフォルトは、空の文字列です。 「Windows デバイス用の Mobile Link Listener キーワード」 70 ページを参照してください。
$poll_notifier	poll_notifier ポーリングキーワードによって指定された Notifier の名前。 「Windows デバイス用の Mobile Link Listener キーワード」 70 ページを参照してください。
$poll_key	poll_key ポーリングキーワードによって指定されたポーリングキー。 「Windows デバイス用の Mobile Link Listener キーワード」70 ページを参照してください。
$poll_every	poll_every ポーリングキーワードによって指定されたポーリング頻度。 「Windows デバイス用の Mobile Link Listener キーワード」 70 ページを参照してください。

参照

- 「-l dblsn オプション」 62 ページ
- 「Windows デバイス用の Mobile Link Listener キーワード」 70 ページ
- 「Windows デバイス用の Mobile Link Listener アクションコマンド」 72 ページ

例

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

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

ライトウェイトポーリング API

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

必要なファイル

すべてのディレクトリは、*%SQLANY12%* を基準とした相対ディレクトリです。次に、ライトウェイトポーリング API のコンパイルに必要なファイルのリストを示します。

ファイル名またはロケーション	説明
Bin32¥mllplib12.dll	ライトウェイトポーリング API ランタイムダイナミックライブラリ
SDK¥Lib¥x86¥mllplib12.lib と *SDK¥Lib¥x64¥mllplib12.lib*	ライトウェイトポーリング API ランタイムインポートライブラリ
SDK¥Include¥mllplib.h	ライトウェイトポーリング API ヘッダーファイル

C の SIS_CarDealer_LP_API サンプルアプリケーションは、*%SQLANYSAMP12%¥MobiLink¥SIS_CarDealer_LP_API* にあります。

API メンバー

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

MLLightPoller クラス

ライトウェイトポーラーオブジェクトを表します。

構文

```
public class MLLightPoller
```

メンバー

名前	説明
「Poll メソッド」	Notifier にキャッシュの Push 要求をチェックさせ、サーバーをポーリングします。
「SetConnectInfo メソッド」	Mobile Link クライアントのストリームタイプとネットワークプロトコルオプションを設定します。
「auth_status 列挙体」	可能な認証ステータスコードを指定します。
「return_code 列挙体」	可能なリターンコードを指定します。

例

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

Poll メソッド

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

構文

```
public virtual return_code MLLightPoller::Poll(
    const char * notifier,
    const char * key,
    char * subject = 0,
    size_t * subjectSize = 0,
    char * content = 0,
    size_t * contentSize = 0
)
```

パラメーター

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

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

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

戻り値

return_code 列挙体にリストされているコードの 1 つ。 「return_code 列挙体」84 ページを参照してください。

備考

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

SetConnectInfo メソッド

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

構文

```
public virtual return_code MLLightPoller::SetConnectInfo(
    const char * streamName,
    const char * streamParams
);
```

パラメーター

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

戻り値

return_code 列挙体にリストされているコードの 1 つ。 「return_code 列挙体」84 ページを参照してください。

例

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

auth_status 列挙体

可能な認証ステータスコードを指定します。

構文

```
public typedef enum MLLightPoller::auth_status;
```

メンバー

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

return_code 列挙体

可能なリターンコードを指定します。

構文

```
public typedef enum MLLightPoller::return_code;
```

メンバー

名前	説明
AUTH_FAILED	Mobile Link サーバーへの接続が認証されませんでした。
BAD_STREAM_NAME	認識されないストリーム名。
BAD_STREAM_PARAM	ストリームパラメーターを解析できませんでした。
COMMUNICATION_ERROR	通信エラーにより失敗しました。
CONNECT_FAILED	Mobile Link サーバーに接続できませんでした。
CONTENT_OVERFLOW	内容バッファーが小さすぎます。
KEY_NOT_FOUND	指定した Notifier から指定したキーの Push 通知がありません。
NYI	内部でのみ使用されます。

名前	説明
OK	メソッドが正常に実行されました。
SUBJECT_OVERFLOW	件名バッファーが小さすぎます。

MLLPCreatePoller メソッド

MLLightPoller のインスタンスを作成します。

構文

```
extern MLLightPoller * MLLPCreatePoller()
```

戻り値

新しい MLLightPoller オブジェクト

参照

- 「MLLPDestroyPoller メソッド」85 ページ

例

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

MLLPDestroyPoller メソッド

MLLightPoller のインスタンスを破棄します。

構文

```
extern void MLLPDestroyPoller(
    MLLightPoller * poller
)
```

パラメーター

- **poller**　破棄する MLLightPoller。

備考

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

参照

- 「MLLPCreatePoller メソッド」85 ページ

例

```
MLLPDestroyPoller(poller);
```


サーバー起動同期のシステムプロシージャー

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

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

参照

- 「デバイストラッキングゲートウェイ」23 ページ
- 「デバイストラッキングのサポート」24 ページ
- 「Mobile Link サーバーのシステムテーブル」『Mobile Link サーバー管理』
- 「Mobile Link サーバーシステムプロシージャー」『Mobile Link サーバー管理』

ml_delete_device システムプロシージャー

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

パラメーター

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

備考

この機能は、デバイストラッキングを手動で設定する場合にだけ役立ちます。

参照

- 「デバイストラッキングのサポート」24 ページ

例

デバイスレコードとこれを参照するすべての関連レコードを削除します。

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

ml_delete_device_address システムプロシージャー

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

パラメーター

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

備考

このシステムプロシージャーは、デバイストラッキングを手動で設定する場合にだけ役立ちます。

参照

- 「デバイストラッキングのサポート」24 ページ

例

アドレスレコードを削除します。

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

ml_delete_listening システムプロシージャー

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

パラメーター

項目	パラメーター	説明
1	ml_user	VARCHAR(128)

備考

このシステムプロシージャーは、デバイストラッキングを手動で設定する場合にだけ役立ちます。

参照

- 「デバイストラッキングのサポート」24 ページ

例

受信者レコードを削除します。

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

ml_set_device システムプロシージャー

手動でデバイストラッキングを設定している場合、このシステムプロシージャーを使用して、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。

パラメーター

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

備考

システムプロシージャー 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。

このシステムプロシージャーは、デバイストラッキングを手動で設定する場合にだけ役立ちます。

参照

- 「デバイストラッキングのサポート」24 ページ
- 「ml_set_device_address システムプロシージャー」90 ページ
- 「ml_set_listening システムプロシージャー」91 ページ

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(
   'myWindowsMobile',
   'MobiLink Listeners for myWindowsMobile - 9.0.1',
   '1',
   'not used',
   'y',
   'manually entered by administrator'
);
```

ml_set_device_address システムプロシージャー

手動でデバイストラッキングを設定している場合、このシステムプロシージャーを使用して、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。

パラメーター

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

備考

システムプロシージャー 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。

このシステムプロシージャーは、デバイストラッキングを手動で設定する場合にだけ役立ちます。

参照

- 「デバイストラッキングのサポート」24 ページ
- 「ml_set_device システムプロシージャー」89 ページ
- 「ml_set_listening システムプロシージャー」91 ページ

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

ml_set_listening システムプロシージャー

手動でデバイストラッキングを設定している場合、このシステムプロシージャーを使用して、Mobile Link ユーザー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。

パラメーター

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

備考

システムプロシージャー 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無効にするのに使用します。

このシステムプロシージャーは、デバイストラッキングを手動で設定する場合にだけ役立ちます。

参照

- 「デバイストラッキングのサポート」24 ページ
- 「ml_set_device システムプロシージャー」89 ページ
- 「ml_set_device_address システムプロシージャー」90 ページ

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 ユーザー名にマッピングします。

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

ml_set_sis_sync_state システムプロシージャー

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

パラメーター

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

備考

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

参照

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

例

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

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

サーバー起動同期の高度なトピック

次の項では、サーバー起動同期に関連する高度なトピックについて説明します。

メッセージ構文

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

message = [subject]content

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

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

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

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

備考

大カッコ、シェブロン、二重引用符、カッコ、一重引用符、角カッコは、内部使用のために予約されています。**subject** 内では使用しないでください。 メッセージの制限事項の詳細については、「Push 要求の使用」9 ページを参照してください。

参照

- 「Windows デバイス用の Mobile Link Listener キーワード」70 ページ
- 「ゲートウェイと Carrier」22 ページ
- 「-m dblsn オプション」63 ページ
- 「-v dblsn オプション」68 ページ

sa_send_udp システムプロシージャーを使用した Push 通知の送信

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

前提条件

●デバイスに Mobile Link Listener が設定され、Push 通知を受信するようになります

●デバイスに Internet Explorer がインストールされます

●デバイス上で次のコマンドが実行されます

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

●SQL Anywhere 統合データベースが Mobile Link サーバー上で稼働されます

内容と備考

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

♦ sa_send_udp システムプロシージャーを使用した Push 通知の送信

1. Interactive SQL を実行し、次などのコマンドを使用し統合データベースに接続します。その際には、*consdb_source_name* を統合データベースの ODBC 名と置き換えます。

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

2. 次のコマンドを実行して、Push 通知を送信します。

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

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

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

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

結果

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

次の手順

なし。

参照

- 「sa_send_udp システムプロシージャー」『SQL Anywhere サーバー SQL リファレンス』

サーバー起動同期チュートリアル

サーバー起動同期の使用方法についての理解を深めるには、次のチュートリアルを使用してください。

チュートリアル：ライトウェイトポーリングを使用したサーバー起動同期の設定

このチュートリアルでは、サーバー起動同期を使用できるように SQL Anywhere 統合データベースとリモートデータベースを設定する方法について説明します。このチュートリアルは、*%SQLANYSAMP12%¥MobiLink¥SIS_CarDealer_LP_DBLSN* に配置されているサンプルコードに基づいています。

サーバー起動同期の実装サンプルは、*%SQLANYSAMP12%¥MobiLink* にあります。サーバー起動同期のすべてのサンプルディレクトリ名には、プレフィクスの SIS_ が付いています。

注意
Sybase Central を使用してリモートデータベースを管理し、その後、ライトウェイトポーリングを使用するサーバー起動同期への代替機能としてサーバー起動リモートタスク (SIRT) を使用できます。 詳細については、「サーバー起動リモートタスク (SIRT)」『Mobile Link サーバー管理』と「チュートリアル：リモートデータベースの集中管理の使用」『Mobile Link クイックスタート』を参照してください。

必要なソフトウェア

- SQL Anywhere 12

前提知識と経験

- Mobile Link イベントスクリプトの基本的な知識。

目的

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

関連項目

- 「サーバー起動同期」 1 ページ

レッスン 1：統合データベースの設定

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

SQL Anywhere 12 ドライバーを使用して、**SIS_CarDealer_LP_DBLSN_CONDB** データベース用の ODBC データソースを定義します。

♦ SQL Anywhere 統合データベースの設定

1. 統合データベースを格納する新しい作業ディレクトリを作成します。

   このチュートリアルでは、*c:¥MLsis* を作業フォルダーとします。

2. dbinit ユーティリティを使用して、新しい SQL Anywhere 統合データベースを作成します。

   次のコマンドを実行します。

   ```
   dbinit SIS_CarDealer_LP_DBLSN_CONDB
   ```

3. 統合データベースを起動します。

   次のコマンドを実行します。

   ```
   dbeng12 SIS_CarDealer_LP_DBLSN_CONDB
   ```

4. **[スタート]** - **[プログラム]** - **[SQL Anywhere 12]** - **[管理ツール]** - **[ODBC データソースアドミニストレーター]** をクリックします。

5. **[ユーザー DSN]** タブをクリックしてから、**[追加]** をクリックします。

6. **[データソースの新規作成]** ウィンドウで、**[SQL Anywhere 12]** をクリックし、**[完了]** をクリックします。

7. **[SQL Anywhere の ODBC 設定]** ウィンドウで、次の操作を行います。
   a. **[ODBC]** タブをクリックします。
   b. **[データソース名]** フィールドに **SIS_CarDealer_LP_DBLSN_CONDB** と入力します。
   c. **[ログイン]** タブをクリックします。
   d. **[ユーザー ID]** フィールドに **DBA** と入力します。
   e. **[パスワード]** フィールドに **sql** と入力します。
   f. **[アクション]** ドロップダウンリストから、**[このコンピューターで稼働しているデータベースに接続]** を選択します。
   g. **[サーバー名]** フィールドに、**SIS_CarDealer_LP_DBLSN_CONDB** と入力します。
   h. **[OK]** をクリックします。

8. ODBC データソースアドミニストレーターを閉じます。

   [ODBC データソースアドミニストレーター] ウィンドウで **[OK]** をクリックします。

9. 「レッスン 2：データベーススキーマの生成」101 ページに進みます。

参照

- 「ODBC データソース」『SQL Anywhere サーバー データベース管理』

レッスン 2：データベーススキーマの生成

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。「レッスン 1：統合データベースの設定」99 ページを参照してください。

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

♦ データベーススキーマの設定

1. **[スタート]** » **[プログラム]** » **[SQL Anywhere 12]** » **[管理ツール]** » **[Sybase Central]** をクリックします。
2. 次のタスクを実行して、統合データベースに接続します。
   a. **[接続]** - **[SQL Anywhere 12 に接続]** をクリックします。
   b. **[アクション]** ドロップダウンリストから、**[ODBC データソースを使用した接続]** を選択します。
   c. **[ODBC データソース名]** をクリックし、**[参照]** をクリックします。
   d. **SIS_CarDealer_LP_DBLSN_CONDB** を選択し、**[OK]** をクリックします。
   e. **[接続]** をクリックします。
3. Interactive SQL を使用してデータベースに接続します。

   Interactive SQL は、Sybase Central またはコマンドプロンプトから起動できます。

   ●Sybase Central から Interactive SQL を起動するには、**SIS_CarDealer_LP_DBLSN_CONDB - DBA** データベースを右クリックし、**[Interactive SQL を開く]** をクリックします。

   ●コマンドプロンプトで Interactive SQL を起動するには、次のコマンドを実行します。

   ```
   dbisql -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB"
   ```

4. 次の SQL 文を実行し、Dealer テーブルと non_sync_request テーブルを作成して設定します。

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

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

5. 次の文を使用して、Dealer テーブルにデータを挿入します。

   ```
   INSERT INTO Dealer(name, rating) VALUES ('Audi', 'a');
   INSERT INTO Dealer(name, rating) VALUES ('Buick', 'b');
   INSERT INTO Dealer(name, rating) VALUES ('Chrysler', 'c');
   INSERT INTO Dealer(name, rating) VALUES ('Dodge', 'd');
   INSERT INTO Dealer(name, rating) VALUES ('Eagle', 'e');
   INSERT INTO Dealer(name, rating) VALUES ('Ford', 'f');
   ```

```
INSERT INTO Dealer(name, rating) VALUES ('Geo', 'g');
INSERT INTO Dealer(name, rating) VALUES ('Honda', 'h');
INSERT INTO Dealer(name, rating) VALUES ('Isuzu', 'l');
COMMIT;
```

6. 次の SQL 文を実行して Mobile Link のシステムテーブルとストアドプロシージャーを作成します。*C:¥Program Files¥SQL Anywhere 12¥* は、SQL Anywhere 12 インストール環境のロケーションに置き換えてください。

```
READ "C:¥Program Files¥SQL Anywhere 12¥MobiLink¥setup¥syncsa.sql"
```

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

```
CALL ml_add_table_script(
   'CarDealer',
   'Dealer',
   'download_cursor',
   'SELECT * FROM Dealer WHERE last_modified >= ?'
);

CALL ml_add_connection_script(
   'CarDealer',
   'publication_nonblocking_download_ack',
   'CALL ml_set_sis_sync_state(
      {ml s.remote_id},
      NULL,
      {ml s.publication_name},
      {ml s.username},
      NULL,
      {ml s.last_publication_download}
   )'
);

CALL ml_add_table_script(
   'CarDealer', 'Dealer', 'download_delete_cursor', '--{ml_ignore}'
);

COMMIT;
```

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

8. Interactive SQL を閉じます。

9. 「レッスン 3：Mobile Link プロジェクトの作成」103 ページに進みます。

参照

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

レッスン 3：Mobile Link プロジェクトの作成

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。「レッスン 1：統合データベースの設定」99 ページを参照してください。

このレッスンでは、新しい Mobile Link プロジェクトを作成して、統合データベースに接続します。

♦ 新しい Mobile Link プロジェクトの作成

1. **[スタート]** » **[プログラム]** » **[SQL Anywhere 12]** » **[管理ツール]** » **[Sybase Central]** をクリックします。
2. **[ツール]** » **[Mobile Link 12]** » **[新しいプロジェクト]** をクリックします。
3. **[新しいプロジェクトの名前を指定してください。]** フィールドに **SIS_CarDealer_LP_DBLSN_CONDB_project** と入力します。
4. **[新しいプロジェクトの保存場所を指定してください。]** フィールドに *C:¥MLsis* と入力し、**[次へ]** をクリックします。
5. **[統合データベースをプロジェクトに追加]** オプションをオンにします。
6. **[データベースの表示名]** フィールドに **SIS_CarDealer_LP_DBLSN_CONDB** と入力します。
7. **[編集]** をクリックします。**[汎用 ODBC データベースに接続]** ウィンドウが表示されます。
8. **[ユーザー ID]** フィールドに **DBA** と入力します。
9. **[パスワード]** フィールドに **sql** と入力します。
10. **[ODBC データソース名]** フィールドで、**[参照]** をクリックして **SIS_CarDealer_LP_DBLSN_CONDB** を選択します。
11. **[OK]** をクリックし、**[保存]** をクリックします。
12. **[パスワードを記憶]** オプションをオンにし、**[完了]** をクリックします。
13. **[OK]** をクリックします。
14. 「レッスン 4：Notifier の設定」103 ページに進みます。

レッスン 4：Notifier の設定

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。「レッスン 1：統合データベースの設定」99 ページを参照してください。

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

request_cursor イベントスクリプトによって、Push 要求が検出されます。各 Push 要求によって、送信される情報と情報を受信するデバイスが決まります。

♦ 新しい Notifier の作成と設定

1. Sybase Central の左ウィンドウ枠の **[Mobile Link 12]** で、**SIS_CarDealer_LP_DBLSN_CONDB_project**、**[統合データベース]**、**SIS_CarDealer_LP_DBLSN_CONDB - DBA** の順に展開します。
2. **[通知]** を右クリックし、**[新規]** » **[Notifier]** をクリックします。
3. **[新しい Notifier の名前を指定してください。]** フィールドに、**CarDealerNotifier** と入力します。
4. **[完了]** をクリックします。
5. 右ウィンドウ枠で **CarDealerNotifier** を選択し、**[ファイル]** - **[プロパティ]** をクリックします。
6. **[イベント]** タブをクリックし、**[イベント]** リストから **[request_cursor]** をクリックします。
7. 表示されたテキストフィールドで次の SQL 文を入力します。

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

8. **[OK]** をクリックして Notifier イベントを保存します。
9. 「レッスン 5：Mobile Link サーバーの起動」104 ページに進みます。

参照

- 「request_cursor イベント」39 ページ
- 「ml_set_sis_sync_state システムプロシージャー」92 ページ

レッスン 5：Mobile Link サーバーの起動

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。「レッスン 1：統合データベースの設定」99 ページを参照してください。

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

♦ Mobile Link サーバー (mlsrv12) を実行します。

1. 統合データベースに接続します。

   次のコマンドを実行します。


```
mlsrv12 -notifier -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB" -o serverOut.txt -v+ -dl -zu+ -x tcpip
```

Mobile Link サーバーメッセージウィンドウが表示されます。Notifier は、デバイスから Push 要求を受信する準備が整ったことを示します。

次の表は、このレッスンで使用する mlsrv12 オプションを示します。オプション -o、-v、は、デバッグとトラブルシューティングの情報を提供します。これらのロギングオプションは、開発環境での使用に適しています。パフォーマンス上の理由から、一般的に -v は運用環境では使用しません。

オプション	説明
-notifier	サーバー起動同期用に有効なすべての Notifier を起動します。 「-notifier mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-c	接続文字列を指定します。 「-c mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-o	メッセージログファイル *serverOut.txt* を指定します。 「-o mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-v+	ログを取る対象となる情報を指定します。-v+ を使用して、最大冗長ロギングをオンに設定します。 「-v mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-zu+	自動的に新しいユーザーを追加します。 「-zu mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-x	Mobile Link クライアントの通信プロトコルとプロトコルオプションを設定します。 「-x mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。

2. 「レッスン 6：リモートデータベースの設定」106 ページに進みます。

参照

- 「Mobile Link サーバー」『Mobile Link サーバー管理』
- 「Mobile Link サーバーオプション」『Mobile Link サーバー管理』

レッスン 6：リモートデータベースの設定

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。 「レッスン 1：統合データベースの設定」99 ページを参照してください。

このレッスンでは、SQL Anywhere リモートデータベースを作成し、同期パブリケーション、ユーザー、サブスクリプションを作成します。

♦ Mobile Link クライアントデータベースの設定

1. dbinit コマンドラインユーティリティを使用して、Mobile Link クライアントデータベースを作成します。

   次のコマンドを実行します。

   ```
   dbinit SIS_CarDealer_LP_DBLSN_REM
   ```

2. dbeng12 コマンドラインユーティリティを使用して、Mobile Link クライアントデータベースを起動します。

   次のコマンドを実行します。

   ```
   dbeng12 SIS_CarDealer_LP_DBLSN_REM
   ```

3. Interactive SQL を使用して Mobile Link クライアントデータベースに接続します。

   次のコマンドを実行します。

   ```
   dbisql -c "SERVER=SIS_CarDealer_LP_DBLSN_REM;UID=DBA;PWD=sql"
   ```

4. Dealer テーブルを作成します。

   Interactive SQL で次の SQL 文を実行します。

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

5. Mobile Link 同期ユーザー、パブリケーション、サブスクリプションを作成します。

   Interactive SQL で次の SQL 文を実行します。

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

6. 「レッスン 7：Mobile Link Listener の設定」107 ページに進みます。

参照

- 「Mobile Link クライアント」『Mobile Link クライアント管理』
- 「CREATE TABLE 文」『SQL Anywhere サーバー SQL リファレンス』
- 「パブリケーション」『Mobile Link クライアント管理』
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバー SQL リファレンス』
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバー SQL リファレンス』
- 「スクリプトバージョン」『Mobile Link サーバー管理』

レッスン 7：Mobile Link Listener の設定

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。 「レッスン 1：統合データベースの設定」99 ページを参照してください。

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

♦ Mobile Link Listner の設定

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

   ```
   dbmlsync -c "SERVER=SIS_CarDealer_LP_DBLSN_REM;UID=DBA;PWD=sql" -e sa=on -o
   rem1.txt -v+
   ```

   Mobile Link Listener は、$remote_id action 変数を使用してポーリングキーを定義できます。このキーは、Mobile Link サーバーでデバイスの識別に使用されます。この変数は、リモート ID ファイル *SIS_CarDealer_LP_DBLSN_REM.rid* から取得します。このファイルは、Mobile Link サーバーと初期同期するときに作成されます。リモート ID ファイルを使用する場合は、Mobile Link サーバーと同期する必要があります。

2. SQL Anywhere Mobile Link クライアントウィンドウで **[シャットダウン]** をクリックします。

3. 次の内容の新しいテキストファイルを作成します。

   ```
   # Verbosity level
   -v2

   # Show notification messages in console and log
   -m

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

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

   # Message handlers

   # Watch for a notification without action
   ```

```
-l "poll_connect='tcpip(host=localhost)';
    poll_notifier=CarDealerNotifier;
    poll_key=$remote_id.no_action;"

# Signal dbmlsync to launch, sync and then shutdown
-l "poll_connect='tcpip(host=localhost)';
    poll_notifier=CarDealerNotifier;
    poll_key=$remote_id.sync;
    action='run dbmlsync.exe -c SERVER=SIS_CarDealer_LP_DBLSN_REM;UID=DBA;PWD=sql -e
sa=on -o rem1.txt -v+';"

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

4. このチュートリアルでは、*c:¥MLsis* をサーバー側コンポーネントの作業フォルダーとします。テキストファイルを *mydblsn.txt* という名前でこのディレクトリに保存します。

5. Mobile Link Listener を起動します。

   コマンドプロンプトで、Mobile Link 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 Mobile Link Listener を起動します。

   ```
   dblsn @mydblsn.txt
   ```

   Mobile Link Listener がスリープ中であることを示す **[MobiLink Listener for Windows]** ウィンドウが表示されます。

6. 「レッスン 8：Push 要求の発行」108 ページに進みます。

参照

- 「Listener」15 ページ
- 「Windows デバイス用の Mobile Link Listener ユーティリティ (dblsn)」55 ページ
- 「@data dblsn オプション」59 ページ
- 「action 変数」18 ページ

レッスン 8：Push 要求の発行

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。「レッスン 1：統合データベースの設定」99 ページを参照してください。

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

♦ サーバー起動同期を実行するための統合データベースの変更

1. 統合データベースに接続していない場合は、Interactive SQL で接続します。

   次のコマンドを実行します。

   ```
   dbisql -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB"
   ```

2. 次の SQL 文を実行します。

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

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

   次のスクリプトを入力します。

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

4. 同期が発生するまで数秒待ちます。

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

5. non_sync_request テーブルからポーリングキー値を削除し、デバイスとのサーバー起動同期を停止します。

   次のスクリプトを入力します。

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

6. 「クリーンアップ」 109 ページに進みます。

参照

- 「Push 要求の生成」 9 ページ
- 「INSERT 文」『SQL Anywhere サーバー SQL リファレンス』
- 「UPDATE 文」『SQL Anywhere サーバー SQL リファレンス』
- 「DELETE 文」『SQL Anywhere サーバー SQL リファレンス』

クリーンアップ

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。 「レッスン 1：統合データベースの設定」 99 ページを参照してください。

チュートリアルをコンピューターから削除します。

♦ コンピューターからのチュートリアルの削除

1. Interactive SQL を閉じます。

2. SQL Anywhere、Mobile Link、同期クライアントの各ウィンドウを閉じます。
3. 次の手順で、チュートリアルに関連するすべての ODBC データソースを削除します。
   a. ODBC データソースアドミニストレーターを起動します。
      コマンドプロンプトで次のコマンドを入力します。

      ```
      odbcad32
      ```

   b. **SIS_CarDealer_LP_DBLSN_CONDB** データソースを削除します。
4. 統合データベースとリモートデータベースが保存されているディレクトリ *c:¥MLsis¥* に移動し、すべてのファイルを削除します。

チュートリアル：ゲートウェイを使用したサーバー起動同期の設定

このチュートリアルでは、サーバー起動同期を使用できるように SQL Anywhere 統合データベースとリモートデータベースを設定する方法について説明します。このチュートリアルは、*%SQLANYSAMP12%¥MobiLink¥SIS_CarDealer* に配置されているサンプルコードに基づいています。

サーバー起動同期のいくつかの実装サンプルは、*%SQLANYSAMP12%¥MobiLink* にあります。サーバー起動同期のすべてのサンプルディレクトリ名には、プレフィクスの SIS_ が付いています。

必要なソフトウェア

- SQL Anywhere 12

前提知識と経験

- Mobile Link イベントスクリプトの基本的な知識。

目的

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

関連項目

- 「サーバー起動同期」1 ページ

レッスン 1：統合データベースの設定

このレッスンでは、dbinit ユーティリティを使用して、同期に必要なスクリプトで **MLconsolidated** という名前の統合データベースを作成します。その後、SQL Anywhere 12 ドライバーを使用して、データベース用の ODBC データソースを定義します。

♦ SQL Anywhere 統合データベースの設定

1. 統合データベースを格納する新しい作業ディレクトリを作成します。

   このチュートリアルでは、*c:¥MLsis* を作業フォルダーとします。

2. dbinit ユーティリティを使用して、新しい SQL Anywhere 統合データベースを作成します。

3. 次のコマンドを実行します。

   ```
   dbinit MLconsolidated
   ```

4. dbeng12 ユーティリティを使用して統合データベースを起動します。

   次のコマンドを実行します。

   ```
   dbeng12 MLconsolidated
   ```

5. **[スタート]** - **[プログラム]** - **[SQL Anywhere 12]** - **[管理ツール]** - **[ODBC データソースアドミニストレーター]** をクリックします。

6. **[ユーザー DSN]** タブをクリックしてから、**[追加]** をクリックします。

7. **[データソースの新規作成]** ウィンドウで、**[SQL Anywhere 12]** をクリックし、**[完了]** をクリックします。

8. **[SQL Anywhere の ODBC 設定]** ウィンドウで、次の操作を行います。
   a. **[ODBC]** タブをクリックします。
   b. **[データソース名]** フィールドに **sis_cons** と入力します。
   c. **[ログイン]** タブをクリックします。
   d. **[ユーザー ID]** フィールドに **DBA** と入力します。
   e. **[パスワード]** フィールドに **sql** と入力します。
   f. **[アクション]** ドロップダウンリストから、**[このコンピューターで稼働しているデータベースに接続]** を選択します。
   g. **[サーバー名]** フィールドに、**MLconsolidated** と入力します。
   h. **[OK]** をクリックします。

9. ODBC データソースアドミニストレーターを閉じます。

   [ODBC データソースアドミニストレーター] ウィンドウで **[OK]** をクリックします。

10. 「レッスン 2：データベーススキーマの生成」 112 ページに進みます。

参照

- 「ODBC データソース」『SQL Anywhere サーバー データベース管理』

レッスン 2：データベーススキーマの生成

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。 「レッスン 1：統合データベースの設定」111 ページを参照してください。

このレッスンでは、データベーススキーマを生成します。このスキーマには、Dealer テーブルと download_cursor 同期スクリプトが含まれます。テーブルとストアドプロシージャーは、サーバー起動同期の Push 要求を生成するために使用されます。

♦ データベーススキーマの設定

1. **[スタート] » [プログラム] » [SQL Anywhere 12] » [管理ツール] » [Sybase Central]** をクリックします。
2. 次のタスクを実行して、統合データベースに接続します。
   a. **[接続]** - **[SQL Anywhere 12 に接続]** をクリックします。
   b. **[アクション]** ドロップダウンリストから、**[ODBC データソースを使用した接続]** をクリックします。
   c. **[ODBC データソース名]** をクリックし、**[参照]** をクリックします。
   d. **sis_cons** を選択し、**[OK]** をクリックします。
   e. **[接続]** をクリックします。
3. Interactive SQL を使用してデータベースに接続します。

   Interactive SQL は、Sybase Central またはコマンドプロンプトから起動できます。

   - Sybase Central から Interactive SQL を起動するには、**MLconsolidated - DBA** データベースを右クリックし、**[Interactive SQL を開く]** をクリックします。
   - コマンドプロンプトで Interactive SQL を起動するには、次のコマンドを実行します。

     ```
     dbisql -c "dsn=sis_cons"
     ```
4. 次の SQL 文を実行し、Dealer テーブルを作成して設定します。

   ```
   CREATE TABLE Dealer (
      name VARCHAR(10) NOT NULL PRIMARY KEY,
      rating VARCHAR(5),
      last_modified TIMESTAMP DEFAULT TIMESTAMP
   )
   ```
5. 次の文を使用して、Dealer テーブルにデータを挿入します。

   ```
   INSERT INTO Dealer(name, rating) VALUES ('Audi', 'a');
   INSERT INTO Dealer(name, rating) VALUES ('Buick', 'b');
   INSERT INTO Dealer(name, rating) VALUES ('Chrysler', 'c');
   INSERT INTO Dealer(name, rating) VALUES ('Dodge', 'd');
   ```



```
INSERT INTO Dealer(name, rating) VALUES ('Eagle', 'e');
INSERT INTO Dealer(name, rating) VALUES ('Ford', 'f');
INSERT INTO Dealer(name, rating) VALUES ('Geo', 'g');
INSERT INTO Dealer(name, rating) VALUES ('Honda', 'h');
INSERT INTO Dealer(name, rating) VALUES ('Isuzu', 'I');
COMMIT;
```

6. 次の SQL スクリプトを実行して Mobile Link のシステムテーブルとストアドプロシージャーを作成します。*C:¥Program Files¥SQL Anywhere 12¥* は、SQL Anywhere 12 インストール環境のロケーションに置き換えてください。

```
READ "C:¥Program Files¥SQL Anywhere 12¥MobiLink¥setup¥syncsa.sql"
```

7. 次の SQL スクリプトを実行し、download_cursor 同期スクリプトを指定して同期を記録します。

```
CALL ml_add_table_script(
   'sis_ver1',
   'Dealer',
   'download_cursor',
   'SELECT * FROM Dealer WHERE last_modified >= ?'
);

CALL ml_add_table_script(
   'sis_ver1', 'Dealer', 'download_delete_cursor', '--{ml_ignore}'
);

COMMIT
```

8. 「レッスン 3：Push 要求を格納するテーブルの作成」113 ページに進みます。

参照

- 「SQL Anywhere データベースサーバーの構文」『SQL Anywhere サーバー データベース管理』
- 「CREATE TABLE 文」『SQL Anywhere サーバー SQL リファレンス』
- 「同期スクリプトの作成」『Mobile Link サーバー管理』
- 「download_cursor テーブルイベント」『Mobile Link サーバー管理』

レッスン 3：Push 要求を格納するテーブルの作成

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。「レッスン 1：統合データベースの設定」111 ページを参照してください。

このレッスンでは、Push 要求を格納する Push 要求テーブルを作成します。Notifier は、Push 要求を検出すると、デバイスにメッセージを送信します。

♦ Push 要求を格納する簡単なテーブルの作成

1. Interactive SQL を使用してデータベースに接続します。

   Interactive SQL は、Sybase Central またはコマンドプロンプトから起動できます。

   ●Sybase Central から Interactive SQL を起動するには、**MLconsolidated - DBA** データベースを右クリックし、**[Interactive SQL を開く]** をクリックします。

● コマンドプロンプトで Interactive SQL を起動するには、次のコマンドを実行します。

```
dbisql -c "dsn=sis_cons"
```

2. Interactive SQL で次の SQL 文を実行します。

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

3. Interactive SQL を閉じます。

4. 「レッスン 4：Mobile Link プロジェクトの作成」114 ページに進みます。

参照

- 「Push 要求」7 ページ
- 「サーバー起動同期」1 ページ
- 「サーバー起動同期のコンポーネント」2 ページ

レッスン 4：Mobile Link プロジェクトの作成

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。 「レッスン 1：統合データベースの設定」111 ページを参照してください。

このレッスンでは、新しい Mobile Link プロジェクトを作成して、統合データベースに接続します。

♦ 新しい Mobile Link プロジェクトの作成

1. **[スタート]** » **[プログラム]** » **[SQL Anywhere 12]** » **[管理ツール]** » **[Sybase Central]** をクリックします。

2. **[ツール]** » **[Mobile Link 12]** » **[新しいプロジェクト]** をクリックします。

3. **[新しいプロジェクトの名前を指定してください。]** フィールドに **sis_cons_project** と入力します。

4. **[新しいプロジェクトの保存場所を指定してください。]** フィールドに *C:¥MLsis* と入力し、**[次へ]** をクリックします。

5. **[統合データベースをプロジェクトに追加]** オプションをオンにします。

6. **[データベースの表示名]** フィールドに **sis_cons** と入力します。

7. **[編集]** をクリックします。**[汎用 ODBC データベースに接続]** ウィンドウが表示されます。

8. **[ユーザー ID]** フィールドに **DBA** と入力します。
9. **[パスワード]** フィールドに **sql** と入力します。
10. **[ODBC データソース名]** フィールドで、**[参照]** をクリックして **sis_cons** を選択します。
11. **[OK]** をクリックし、**[保存]** をクリックします。
12. **[パスワードを記憶]** オプションをオンにし、**[完了]** をクリックします。
13. **[OK]** をクリックします。
14. 「レッスン 5：Notifier の設定」115 ページに進みます。

レッスン 5：Notifier の設定

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。 「レッスン 1：統合データベースの設定」111 ページを参照してください。

このレッスンでは、Notifier で Push 要求を作成し、要求を Mobile Link Listener に送信し、有効期限が切れた要求を削除する方法を定義する、3 つの Notifier イベントを設定します。

Notifier は、統合データベース内の変更を検出し、begin_poll イベントを使用して Push 要求を作成します。この場合、変更が Dealer テーブルで発生し、リモートデータベースが最新でない場合、begin_poll スクリプトは、PushRequest テーブルを設定します。

request_cursor スクリプトは、Push 要求をフェッチします。各 Push 要求により、メッセージで送信される情報、情報を受信するリモートデータベースが決まります。

request_delete Notifier イベントはクリーンアップ処理を指定します。このスクリプトを使用すると、暗黙に除外された要求、期限が切れた要求が自動的に削除されます。

♦ 新しい Notifier の作成と設定

1. Sybase Central の左ウィンドウ枠の **[Mobile Link 12]** で、**sis_cons_project**、**[統合データベース]**、**sis_cons** の順に展開します。
2. **[通知]** を右クリックし、**[新規]** » **[Notifier]** をクリックします。
3. **[新しい Notifier の名前を指定してください。]** フィールドに、**CarDealerNotifier** と入力します。
4. **[完了]** をクリックします。
5. begin_poll イベントスクリプトを入力します。
   a. 右ウィンドウ枠で **CarDealerNotifier** を選択し、**[ファイル]** - **[プロパティ]** をクリックします。
   b. **[イベント]** タブをクリックします。
   c. **[イベント]** リストから **[begin_poll]** を選択します。

d. 表示されたテキストフィールドで次の SQL 文を入力します。

```
--
-- Insert the last consolidated database
-- modification date into @last_modified
--
DECLARE @last_modified timestamp;
SELECT MAX(last_modified) INTO @last_modified FROM Dealer;

--
-- Delete processed requests if the mluser is up-to-date
--
DELETE FROM PushRequest
   FROM PushRequest AS p, ml_user AS u, ml_subscription AS s
   WHERE p.status = 'processed'
      AND u.name = p.mluser
      AND u.user_id = s.user_id
      AND @last_modified <= GREATER(s.last_upload_time, s.last_download_time);

--
-- Insert new requests when a device is not up-to-date
--
INSERT INTO PushRequest(mluser, subject, content)
SELECT u.name, 'sync', 'ignored'
   FROM ml_user as u, ml_subscription as s
   WHERE u.name IN (SELECT name FROM ml_listening WHERE listening = 'y')
      AND u.user_id = s.user_id
      AND @last_modified > greater(s.last_upload_time, s.last_download_time)
      AND u.name NOT LIKE '%-dblsn'
      AND NOT EXISTS(SELECT * FROM PushRequest
         WHERE PushRequest.mluser = u.name
            AND PushRequest.subject = 'sync')
```

begin_poll スクリプトの最初の主要セクションでは、デバイスが最新の状態でない場合は、PushRequest テーブルからの処理済み要求は削除されます。

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

@last_modified は、統合データベース Dealer テーブルで最大の変更が行われた日です。式 greater(s.last_upload_time, s.last_download_time) は、リモートデータベースの最後の同期時間を表します。

request_delete イベントを使用して、直接 Push 要求を削除することもできます。ただし、この場合に begin_poll イベントを使用すると、リモートデータベースが同期する前に、同期期限が切れた要求や暗黙的に除外された要求が削除されないようにすることができます。

コードの次のセクションは、Dealer テーブルの last_modified カラムに加えられた変更をチェックし、ml_listening テーブルにリストされた最新の状態にないすべてのアクティブ Mobile Link Listener に対して Push 要求を発行します。

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

PushRequest テーブルが移植されると、begin_poll スクリプトが件名を 'sync' に設定します。

6. request_cursor スクリプトを入力します。

   a. **[イベント]** リストから **[request_cursor]** をクリックします。

b. 表示されたテキストフィールドで次の SQL 文を入力します。

```
SELECT
   p.req_id,
   'Default-DeviceTracker',
   p.subject,
   p.content,
   p.mluser,
   p.resend_interval,
   p.time_to_live
   FROM PushRequest AS p
```

PushRequest テーブルによって、request_cursor スクリプトにローが入力されます。

順序と request_cursor 結果セットの値は重要です。たとえば、2 番目のパラメーターは、デフォルトのゲートウェイ Default-DeviceTracker を定義します。デバイストラッキングゲートウェイは、ユーザーへのアクセス方法を追跡し、UDP または SMTP を自動的に選択してリモートデバイスに接続します。

7. request_delete スクリプトを入力します。
   a. **[イベント]** リストから **[request_delete]** をクリックします。
   b. 表示されたテキストフィールドで次の SQL 文を入力します。

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

request_delete スクリプトは、ローを削除するのではなく、PushRequest テーブルのローのステータスを 'processed' に更新します。

8. **[OK]** をクリックして Notifier イベントを保存します。
9. 「レッスン 6：ゲートウェイと Carrier の設定」117 ページに進みます。

参照

- 「request_delete イベント」40 ページ
- 「begin_poll イベント」35 ページ
- 「デバイストラッキングゲートウェイ」23 ページ
- 「request_cursor イベント」39 ページ
- 「request_delete イベント」40 ページ

レッスン 6：ゲートウェイと Carrier の設定

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。 「レッスン 1：統合データベースの設定」111 ページを参照してください。

ゲートウェイは、メッセージを送信するためのメカニズムです。サポートされるゲートウェイまたはデバイストラッキングゲートウェイを定義できます。デバイストラッキングゲートウェイを指定すると、Mobile Link サーバーではクライアントへのアクセス方法を追跡して、最適なゲートウェイを自動的に選択します。

このチュートリアルでは、デフォルトのデバイストラッキングゲートウェイを使用するため、設定は必要ありません。

「レッスン 7：Mobile Link サーバーの起動」118 ページに進みます。

参照

- 「ライトウェイトポーラーの代替としてのゲートウェイ」22 ページ
- 「ゲートウェイと Carrier」22 ページ
- 「デバイストラッキングゲートウェイプロパティ」49 ページ

レッスン 7：Mobile Link サーバーの起動

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。 「レッスン 1：統合データベースの設定」111 ページを参照してください。

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

♦ Mobile Link サーバー (mlsrv12) を実行します。

1. 統合データベースに接続します。

   次のコマンドを実行します。

   ```
   mlsrv12 -notifier -c "dsn=sis_cons" -o serverOut.txt -v+ -dl -zu+ -x tcpip
   ```

   Mobile Link サーバーメッセージウィンドウが表示されます。Notifier は、デバイスから Push 要求を受信する準備が整ったことを示します。

   次の表は、このレッスンで使用する mlsrv12 オプションを示します。オプション -o、-v、は、デバッグとトラブルシューティングの情報を提供します。これらのロギングオプションは、開発環境での使用に適しています。パフォーマンス上の理由から、一般的に -v は運用環境では使用しません。

オプション	説明
-notifier	サーバー起動同期用に有効なすべての Notifier を起動します。 「-notifier mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-c	接続文字列を指定します。 「-c mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-o	メッセージログファイル *serverOut.txt* を指定します。 「-o mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。

オプション	説明
-v+	ログを取る対象となる情報を指定します。-v+ を使用して、最大冗長ロギングをオンに設定します。 「-v mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-zu+	自動的に新しいユーザーを追加します。 「-zu mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-x	Mobile Link クライアントの通信プロトコルとプロトコルオプションを設定します。 「-x mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。

2. 「レッスン 8：リモートデータベースの設定」119 ページに進みます。

参照

このレッスンのトピックの詳細については、次の各項を参照してください。

- 「Mobile Link サーバー」『Mobile Link サーバー管理』
- 「Mobile Link サーバーオプション」『Mobile Link サーバー管理』

レッスン 8：リモートデータベースの設定

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。 「レッスン 1：統合データベースの設定」111 ページを参照してください。

このレッスンでは、SQL Anywhere リモートデータベースを作成し、同期パブリケーション、ユーザー、サブスクリプションを作成します。

♦ Mobile Link クライアントデータベースの設定

1. dbinit コマンドラインユーティリティを使用して、Mobile Link クライアントデータベースを作成します。

   次のコマンドを実行します。

   ```
   dbinit remote1
   ```

2. dbeng12 コマンドラインユーティリティを使用して、Mobile Link クライアントデータベースを起動します。

   次のコマンドを実行します。

   ```
   dbeng12 remote1
   ```

3. Interactive SQL を使用して Mobile Link クライアントデータベースに接続します。

   次のコマンドを実行します。

   ```
   dbisql -c "SERVER=remote1;UID=DBA;PWD=sql"
   ```

4. Dealer テーブルを作成します。

   Interactive SQL で次の SQL 文を実行します。

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

5. Mobile Link 同期ユーザー、パブリケーション、サブスクリプションを作成します。

   Interactive SQL で次の SQL 文を実行します。

   ```
   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';
   COMMIT;
   ```

6. 「レッスン 9：Mobile Link Listener の設定」 120 ページに進みます。

参照

- 「Mobile Link クライアント」『Mobile Link クライアント管理』
- 「CREATE TABLE 文」『SQL Anywhere サーバー SQL リファレンス』
- 「パブリケーション」『Mobile Link クライアント管理』
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバー SQL リファレンス』
- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバー SQL リファレンス』
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバー SQL リファレンス』
- 「スクリプトバージョン」『Mobile Link サーバー管理』

レッスン 9：Mobile Link Listener の設定

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。 「レッスン 1：統合データベースの設定」 111 ページを参照してください。

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

♦ Mobile Link Listner の設定

1. 次の内容の新しい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.txt
-ot dblsn.txt

# 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 SERVER=remote1;UID=DBA;PWD=sql
 -o dbmlsyncOut.txt';"
```

2. このチュートリアルでは、*c:¥MLsis* をサーバー側コンポーネントの作業フォルダーとします。テキストファイルを *mydblsn.txt* という名前でこのディレクトリに保存します。

3. Mobile Link Listener を起動します。

   コマンドプロンプトで、Mobile Link 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 Mobile Link Listener を起動します。

```
dblsn @mydblsn.txt
```

   Mobile Link Listener がスリープ中であることを示す **[MobiLink Listener for Windows]** ウィンドウが表示されます。

   トラッキング情報が統合データベースにアップロードされると、Mobile Link サーバーメッセージウィンドウに新しいエントリが表示されます。この情報は、Mobile Link Listener と Mobile Link サーバー間の正常な初期通信をリレーします。

4. 「レッスン 10：Push 要求の発行」122 ページに進みます。

参照

- 「Listener」15 ページ
- 「Windows デバイス用の Mobile Link Listener ユーティリティ (dblsn)」55 ページ
- 「@data dblsn オプション」59 ページ

レッスン 10：Push 要求の発行

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。「レッスン 1：統合データベースの設定」111 ページを参照してください。

サーバー起動同期では、直接 PushRequest テーブルを移植するか、Dealer テーブルで変更を加えることで、Push 要求を発行することができます。後者の場合、Notifier の begin_poll スクリプトは、Dealer テーブルの変更を検出して、PushRequest テーブルを移植します。

どちらの場合も、PushRequest テーブルが Notifier の request_cursor スクリプトにローを入力します。これによって、リモートデバイスでメッセージを受信する方法が決まります。

♦ サーバー起動同期を要求する Push 要求の PushRequest テーブルへの直接の挿入

1. 統合データベースに接続していない場合は、Interactive SQL で接続します。

   次のコマンドを実行します。

   ```
   dbisql -c "dsn=sis_cons"
   ```

2. 次の SQL 文を実行します。

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

3. 同期が発生するまで数秒待ちます。

   移植すると、PushRequest テーブルは Notifier の request_cursor スクリプトにローを入力します。request_cursor スクリプトは、メッセージで送信される情報と、情報を受信するリモートデバイスを決定します。

4. 次の SQL 文を実行し、サーバー起動同期を要求するように、統合データベース Dealer に変更を加えます。

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

5. 同期が発生するまで数秒待ちます。

   この場合、Notifier の begin_poll スクリプトは Dealer テーブルの変更を検出し、PushRequest テーブルを適切に移植します。この場合も、PushRequest テーブルが移植されると、Notifier の request_cursor スクリプトは、メッセージで送信される情報と、情報を受信するリモートデバイスを決定します。

6. 「クリーンアップ」123 ページに進みます。

参照

- 「Push 要求の生成」9 ページ
- 「INSERT 文」『SQL Anywhere サーバー SQL リファレンス』
- 「UPDATE 文」『SQL Anywhere サーバー SQL リファレンス』

クリーンアップ

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。「レッスン 1：統合データベースの設定」111 ページを参照してください。

チュートリアルをコンピューターから削除します。

♦ コンピューターからのチュートリアルの削除

1. Interactive SQL を閉じます。
2. SQL Anywhere、Mobile Link、同期クライアントの各ウィンドウを閉じます。
3. 次の手順で、チュートリアルに関連するすべての ODBC データソースを削除します。
   a. ODBC データソースアドミニストレーターを起動します。
      コマンドプロンプトで次のコマンドを入力します。

      ```
      odbcad32
      ```

   b. **sis_cons** データソースを削除します。
4. 統合データベースとリモートデータベースが保存されているディレクトリ *c:¥MLsis¥* に移動し、すべてのファイルを削除します。

索引

記号

A

B

C

D

E

L

M

N

P

R

S

U

あ

う

え

お

か

と

は

ふ

ほ

ま

め

ゆ

ら

り