SYBASE iAnywhere

Ultra Light - M-Business Anywhere プログラミング

改訂 2007 年 3 月

版権と商標

目次

はじめに

このマニュアルの内容

このマニュアルは、Ultra Light for M-Business Anywhere について説明します。Ultra Light for M-Business Anywhere を使用すると、Palm OS、Windows CE、または Windows を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対して Web ベースのデータベース・アプリケーションを開発、配備できます。

M-Business Anywhere は、Web ベースのモバイル・アプリケーションを開発、配備するための iAnywhere プラットフォームです。以前の名称は、AvantGo M-Business Server でした。

対象読者

このマニュアルは、Ultra Light リレーショナル・データベースのパフォーマンス、リソース効率、堅牢性、セキュリティを利用してデータを格納、同期することを目的とするアプリケーション開発者を対象にしています。

SQL Anywhere のマニュアル

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

SQL Anywhere のマニュアル

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

- **『SQL Anywhere 10 - 紹介』** このマニュアルでは、データの管理および交換機能を提供する包括的なパッケージである SQL Anywhere 10 について説明します。SQL Anywhere を使用すると、サーバ環境、デスクトップ環境、モバイル環境、リモート・オフィス環境に適したデータベース・ベースのアプリケーションを迅速に開発できるようになります。
- **『SQL Anywhere 10 - 変更点とアップグレード』** このマニュアルでは、SQL Anywhere 10 とそれ以前のバージョンに含まれる新機能について説明します。
- **『SQL Anywhere サーバ - データベース管理』** このマニュアルでは、SQL Anywhere データベースの実行、管理、設定について説明します。管理ユーティリティとオプションのほか、データベース接続、データベース・サーバ、データベース・ファイル、バックアップ・プロシージャ、セキュリティ、高可用性、Replication Server を使用したレプリケーションについて説明します。
- **『SQL Anywhere サーバ - SQL の使用法』** このマニュアルでは、データベースの設計と作成の方法、データのインポート・エクスポート・変更の方法、データの検索方法、ストアド・プロシージャとトリガの構築方法について説明します。
- **『SQL Anywhere サーバ - SQL リファレンス』** このマニュアルは、SQL Anywhere で使用する SQL 言語の完全なリファレンスです。また、SQL Anywhere のシステム・ビューとシステム・プロシージャについても説明しています。
- **『SQL Anywhere サーバ - プログラミング』** このマニュアルでは、C、C++、Java プログラミング言語、Visual Studio .NET を使用してデータベース・アプリケーションを構築、配備する方法について説明します。Visual Basic や PowerBuilder などのツールのユーザは、それらのツールのプログラミング・インタフェースを使用できます。
- **『SQL Anywhere 10 - エラー・メッセージ』** このマニュアルでは、SQL Anywhere エラー・メッセージの完全なリストを、その診断情報とともに説明します。
- **『Mobile Link - クイック・スタート』** このマニュアルでは、セッションベースのリレーショナル・データベース同期システムである Mobile Link について説明します。Mobile Link テクノロジは、双方向レプリケーションを可能にし、モバイル・コンピューティング環境に非常に適しています。
- **『Mobile Link - サーバ管理』** このマニュアルでは、Mobile Link アプリケーションを設定して管理する方法について説明します。

- **『Mobile Link - クライアント管理』** このマニュアルでは、Mobile Link クライアントを設定、構成、同期する方法について説明します。Mobile Link クライアントには、SQL Anywhere または Ultra Light のいずれかのデータベースを使用できます。
- **『Mobile Link - サーバ起動同期』** このマニュアルでは、Mobile Link のサーバによって開始される同期について説明します。サーバによって開始される同期とは、統合データベースから同期またはその他のリモート・アクションの開始を可能にする Mobile Link の機能です。
- **『QAnywhere』** このマニュアルでは QAnywhere について説明します。QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアント用のメッセージング・プラットフォームであるほか、モバイル・クライアントや無線クライアント用のメッセージング・プラットフォームでもあります。
- **『SQL Remote』** このマニュアルでは、モバイル・コンピューティング用の SQL Remote データ・レプリケーション・システムについて説明します。このシステムによって、SQL Anywhere の統合データベースと複数の SQL Anywhere リモート・データベースの間で、電子メールやファイル転送などの間接的リンクを使用したデータ共有が可能になります。
- **『SQL Anywhere 10 - コンテキスト別ヘルプ』** このマニュアルには、[接続] ダイアログ、クエリ・エディタ、Mobile Link モニタ、SQL Anywhere コンソール・ユーティリティ、インデックス・コンサルタント、Interactive SQL のコンテキスト別のヘルプが収録されています。
- **『Ultra Light - データベース管理とリファレンス』** このマニュアルでは、小型デバイス用 Ultra Light データベース・システムの概要を説明します。
- **『Ultra Light - AppForge プログラミング』** このマニュアルでは、Ultra Light for AppForge について説明します。Ultra Light for AppForge を使用すると、Palm OS、Symbian OS、または Windows CE を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。
- **『Ultra Light - .NET プログラミング』** このマニュアルでは、Ultra Light.NET について説明します。Ultra Light.NET を使用すると、PC、ハンドヘルド、モバイル、埋め込みデバイスのデータベース・アプリケーションを開発し、これらのデバイスに配備できます。
- **『Ultra Light - M-Business Anywhere プログラミング』** このマニュアルは、Ultra Light for M-Business Anywhere について説明します。Ultra Light for M-Business Anywhere を使用すると、Palm OS、Windows CE、または Windows XP を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対して Web ベースのデータベース・アプリケーションを開発、配備できます。
- **『Ultra Light - C/C++ プログラミング』** このマニュアルでは、Ultra Light C および Ultra Light C++ のプログラミング・インタフェースについて説明します。Ultra Light を使用すると、ハンドヘルド、モバイル、埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。

マニュアルの形式

SQL Anywhere のマニュアルは、次の形式で提供されています。

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

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

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

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

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

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

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

表記の規則

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

SQL 構文の表記規則

SQL 構文の表記には、次の規則が適用されます。

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

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

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

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

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

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

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

- **オプション部分**　文のオプション部分は角カッコで囲みます。

  RELEASE SAVEPOINT [*savepoint-name*]

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

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

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

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

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

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

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

オペレーティング・システムの表記規則

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

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

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

ファイルの命名規則

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

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

  ```
  MobiLink¥redirector
  ```

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

  ```
  MobiLink/redirector
  ```

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

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

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

オペレーティング・システム別の *install-dir* のデフォルト・ロケーションの詳細については、「SQLANY10 環境変数」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **samples-dir** インストール・プロセスでは、SQL Anywhere に含まれるサンプルをインストールするロケーションを選択できます。マニュアルでは、このロケーションは *samples-dir* という表記で示されます。

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

  オペレーティング・システム別の *samples-dir* のデフォルト・ロケーションの詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **環境変数** マニュアルでは、環境変数設定が引用されます。Windows では、環境変数を参照するのに、構文 *%envvar%* が使用されます。UNIX、Linux、Mac OS X では、環境変数を参照するのに、構文 *$envvar* または *${envvar}* が使用されます。

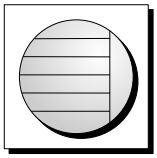
  UNIX、Linux、Mac OS X 環境変数は、*.cshrc* や *.tcshrc* などのシェルとログイン・スタートアップ・ファイルに格納されます。

グラフィック・アイコン

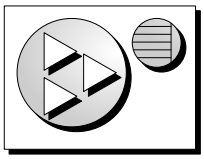
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

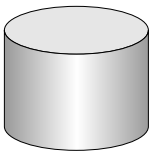
- SQL Anywhere などのデータベース・サーバ



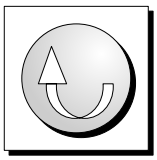
- Ultra Light アプリケーション



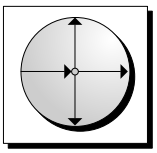
- データベース。高度な図では、データベースとデータベースを管理するデータ・サーバの両方をこのアイコンで表します。



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



- Sybase Replication Server



- プログラミング・インタフェース

詳細情報の検索／フィードバックの提供

詳細情報の検索

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

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

ニュースグループにメッセージをお送りいただく際には、ご使用の SQL Anywhere バージョンのビルド番号を明記し、現在発生している問題について詳しくお知らせくださいますようお願いいたします。バージョン情報は、コマンド・プロンプトで **dbeng10 -v** と入力して確認できます。

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

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

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

フィードバック

このマニュアルに関するご意見、ご提案、フィードバックをお寄せください。

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

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

第 1 章

Ultra Light for M-Business の概要

目次

Ultra Light for M-Business Anywhere の特徴

Ultra Light for M-Business Anywhere は、モバイル・デバイスのためのリレーショナル・データ管理システムです。ビジネス・アプリケーションに必要なパフォーマンス、リソース効率、堅牢性、セキュリティを備えています。Ultra Light では、エンタープライズ・データ・ストアとの同期も提供されます。

システムの稼働条件とサポートされるプラットフォーム

開発プラットフォーム

Ultra Light for M-Business Anywhere を使用してアプリケーションを開発するには、以下が必要です。

- M-Business Anywhere は、AvantGo M-Business Server の新しい名前です。このソフトウェアには、M-Business Server 5.3 以降と、該当する M-Business Anywhere クライアントが必要です。

ターゲット・プラットフォーム

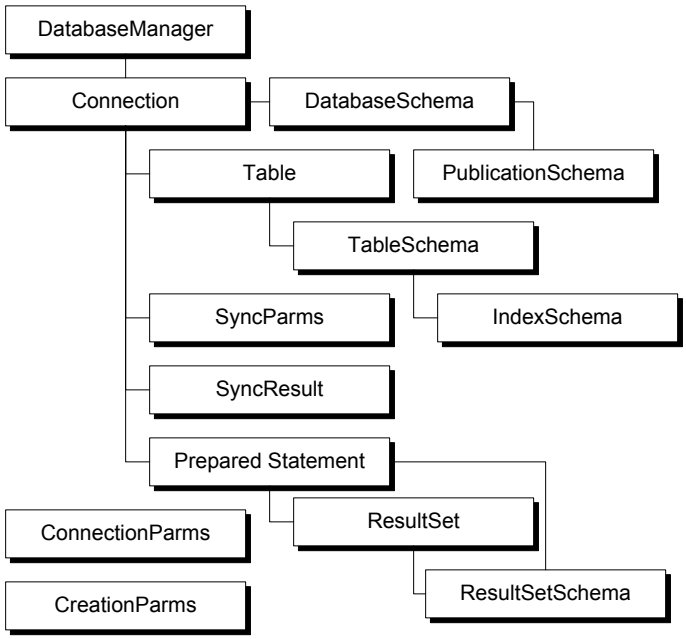
Ultra Light for M-Business Anywhere は、次のターゲット・プラットフォームをサポートしています。

- ARM プロセッサの Pocket PC に搭載した Windows CE 3.0 以降 (Windows Mobile 5.0 を含む)
- Palm OS バージョン 5.0 以降
- Windows では、M-Business Anywhere 5.5 以降

配備の詳細については、「SQL Anywhere 用 Ultra Light 展開オプション」を参照してください。

Ultra Light for M-Business Anywhere のアーキテクチャ

Ultra Light プログラミング・インタフェースは、Ultra Light データベースを使用したデータ操作のためのオブジェクト・セットを公開しています。次の図は、オブジェクト階層を示します。



次のリストは、よく使用される高度なオブジェクトの一部を示します。

- **DatabaseManager** Ultra Light データベースへの接続を管理します。

  「DatabaseManager クラス」 73 ページを参照してください。

- **ConnectionParms** 接続パラメータのセットを格納します。

  「ConnectionParms クラス」 68 ページを参照してください。

- **CreationParms** データベース作成パラメータのセットを格納します。

  「CreationParms クラス」 71 ページを参照してください。

- **Connection** データベース接続を表し、トランザクションを管理します。

  「Connection クラス」 60 ページを参照してください。

- **PreparedStatement、ResultSet、および ResultSetSchema** SQL を使用してデータベース要求とその結果を管理します。

  次の項を参照してください。

- 「PreparedStatement クラス」 85 ページ
- 「ResultSet クラス」 95 ページ
- 「ResultSetSchema クラス」 111 ページ

- **Table**　テーブル・ベースの API を使用してデータを管理します。

  「ULTable クラス」 152 ページを参照してください。

- **SyncParms と SyncResult**　Mobile Link サーバを介して同期を管理します。

  Mobile Link との同期の詳細については、「Ultra Light クライアント」『Mobile Link - クライアント管理』を参照してください。

第 2 章

Ultra Light for M-Business Anywhere 開発の概要

目次

Ultra Light for M-Business Anywhere クイック・スタート

次の手順は、付属するサンプル・アプリケーション CustDB と Simple を実行する方法を示します。

始める前に、M-Business Anywhere 6.0 以降がインストールされて実行されていることと、サーバで管理者の権限を持っていることを確認してください。また、サポートされているハンドヘルド・デバイスが必要です。

♦ **M-Business Anywhere のサンプルをインストールして実行するには、次の手順に従います。**

1. Ultra Light for M-Business Anywhere サンプル・ファイルを配備するためにインストール・ディレクトリにコピーします。

   a. コマンド・プロンプトを開き、SQL Anywhere インストール環境の *samples-dir¥UltraLiteForMBusinessAnywhere¥CustDB* サブディレクトリに移動します。

   b. 次のコマンドを実行します。

      build.bat *deploy-dir*

      ここで *deploy-dir* は、CustDB ファイルと Ultra Light ファイルを配備するディレクトリです。たとえば、*C:¥tutorial¥mba* を選択します。

      このバッチ・ファイルは、必要なファイルを指定したロケーションにコピーします。コピーされるファイルを確認するには、テキスト・エディタで *samples-dir¥UltraLiteForMBusinessAnywhere¥CustDB¥build.bat* ファイルを開きます。

2. Web サーバで仮想ディレクトリを作成します。この仮想ディレクトリは、手順 1 で指定したディレクトリ *deploy-dir* を指すようにします。次に示すのは、Microsoft IIS の場合の手順です。

   a. IIS の管理ツールを開きます。

   b. Web サイトを右クリックし、[新規作成] – [仮想ディレクトリ] を選択します。この仮想ディレクトリに **CustDB** という名前を付け、コンテンツ・ディレクトリとして配備ディレクトリ *deploy-dir* を指定します。他の設定は、デフォルト値のままにします。

   c. 作成した仮想ディレクトリを右クリックして、[プロパティ] を選択します。[HTTP ヘッダー] タブで [ファイルの種類] をクリックし、次のファイル拡張子をタイプ application/octet-stream として登録します。

      - Windows と Windows CE の場合：cab、dll
      - Palm OS の場合：pdb、prc
      - udb

   d. この仮想ディレクトリにあるファイル *main.htm* にアクセスするための URL をメモしておきます。デフォルトのインストール環境では、*http://localhost/CustDB/main.htm* です。

3. M-Business Anywhere にユーザを追加します。

新しいユーザを M-Business Anywhere に追加するには、新しいユーザ・プロファイルの作成、ユーザによる自己登録の許可、CSV ファイルのインポート、という 3 つの方法があります。ここでは、新しいユーザ・プロファイルを作成する方法について説明します。詳細については、M-Business Anywhere のマニュアルを参照してください。

a. 管理者として M-Business Anywhere にログインします。

   デフォルトの管理者アカウントの設定は、ユーザ ID が **Admin** で、パスワードは空です。

b. 左ウィンドウ枠で [Users] をクリックします。

c. [Create User] をクリックします。[Create User] ページが表示されます。

d. [User Name] フィールドにユニークなユーザ名を入力します。

e. [Password] フィールドと [Confirm Password] フィールドに同じパスワードを入力します。

f. [Create] をクリックしてユーザを追加します。

4. M-Business Anywhere クライアントをハンドヘルド・デバイスまたは PC に配備します。

   a. M-Business Anywhere ログイン・ページの [Download Client Software Only] リンクをクリックします。インストール・プログラムを実行して、クライアントをインストールします。

   b. ハンドヘルド・デバイスまたは PC で、M-Business Anywhere Server と同期するように M-Business Connect を設定します。

      作成した新しい M-Business ユーザ・アカウントのユーザ ID とパスワードを入力します。

   c. M-Business Anywhere Server と同期させます。

      このときに接続の問題が発生した場合は、ホスト名ではなく IP アドレスを使用してホストを指定します (ActiveSync の一部バージョンで発生する名前解決の問題を回避するため)。

   詳細については、M-Business Anywhere のマニュアルを参照してください。

5. M-Business Anywhere にグループを追加します。

   このグループは、Ultra Light for M-Business Anywhere をテストするために使用します。

   a. Web ブラウザから M-Business Anywhere に接続します。

      デフォルト URL は、*http://localhost* または *http://localhost:8091* です。

   b. 管理者アカウントを使用してログインします。

   c. 左のナビゲーション・ウィンドウ枠で [Groups] オプションをクリックし、[Create Group] をクリックします。

   d. グループに **UltraLite Samples** という名前を付けます。

6. M-Business Anywhere チャネルを設定します。

   a. 左ウィンドウ枠の [Users] オプションの [Edit Group] で、手順 3 で作成したユーザをグループ UltraLite Samples に追加します。

   b. 左のナビゲーション・ウィンドウ枠でグループの [Channels] オプションを使用し、次のチャネルを作成します。

設定	値
[Title]	CustDB
[Location]	*http://localhost/CustDB/main.htm* または手順 2 でメモした URL
[Channel Size Limit]	1000
[Link depth]	3
[Allow binary distribution]	チェックする
[Hide From Users]	チェックしない

   [Location] の値を設定したら、[View] をクリックして、入力した値が正しいことを確認してください。

7. クライアントを同期します。

   最初の同期では、Ultra Light for M-Business Anywhere ファイルがハンドヘルド・デバイスにダウンロードされます。

♦ 設定を確認するには、次の手順に従います。

1. 必要なファイルが存在することをチェックします。

   - Windows CE の場合は、デバイスを同期したら、次のファイルが *¥Program Files¥AvantGo ¥Pods* フォルダに存在することをチェックします。

     - *ulpod10.dll*
     - *custdb.udb*

     いずれかのファイルが存在しない場合は、そのファイルを手動でデバイスにコピーする必要がある場合があります。

   - Palm OS の場合は、デバイスを同期したら、次の項目が存在することを Palm OS のアプリケーション情報でチェックします。

     - ulpod
     - custdb

いずれかの項目が存在しない場合は、Palm のインストール・ユーティリティを使用して、Ultra Light M-Business Anywhere のランタイム *.prc* ファイルと サンプル・スキーマ *.pdb* ファイルをデバイスにインストールする必要がある場合があります。

- Windows デスクトップの場合は、デバイスを同期したら、次のファイルが *AvantGo Connect* フォルダの *AvantGo¥Pods* サブディレクトリに存在することをチェックします。
  - *ulpod10.dll*
  - *custdb.udb*

  いずれかのファイルが存在しない場合は、そのファイルを手動でデバイスにコピーする必要がある場合があります。

2. M-Business Client を起動します。

   ハンドヘルド・デバイスまたは PC で、[About] 画面に Ultra Light for M-Business Anywhere のバージョン番号が表示されることをチェックします。これにより、Ultra Light for M-Business Anywhere が正常にインストールされたことが確認できます。

3. CustDB サンプル・アプリケーションを実行します。

   a. デスクトップ・コンピュータで、Mobile Link サーバを起動します。

      [スタート] – [プログラム] – [SQL Anywhere 10] – [Mobile Link] – [Mobile Link サーバのサンプル] を選択します。

   b. M-Business Client で CustDB アプリケーションを起動します。

      CustDB アプリケーションは、M-Business ホームページ上のリンクです。

   c. ユーザ ID を入力します。

      デフォルト値は **50** です。

   d. 同期を実行します。

      [Do you have a network connection now?] というプロンプトに対して [はい] と応答するか、CustDB アプリケーションで [同期] をクリックします。これにより Mobile Link とデータが同期されますが、この操作は M-Business Anywhere との同期とは独立しています。

      CustDB のフィールドにデータが表示されます。これで、CustDB アプリケーションを利用できるようになりました。

「Mobile Link CustDB サンプルの解説」『Mobile Link - クイック・スタート』を参照してください。

複数データベースとの HotSync

Palm OS デバイス上の各 Ultra Light データベースには、HotSync で適切に処理されるように、固有の作成者 ID が必要です。また、その作成者 ID が設定されたアプリケーションが Palm OS デバイス上に存在する必要があります。

HotSync マネージャは、各アプリケーションの作成者 ID と識別子を使用して同期を処理します。同期を実行するために、適切に設定された各 Ultra Light アプリケーションは、Mobile Link コンジットに渡されます。このコンジットは、アプリケーションと同じ作成者 ID が設定されたデータベースを検索して同期を実行します。

コンジットの設定については、「HotSync 同期の概要」 『Mobile Link - クライアント管理』を参照してください。

すべての Ultra Light for M-Business Anywhere アプリケーションは、M-Business Client の作成者 ID である **AvGo** を継承します。この継承により、作成者 ID が **AvGo** である Ultra Light データベース 1 つだけが同期でき、異なる作成者 ID をデータベースに割り当てた場合は、対応する作成者 ID が設定されたアプリケーションが存在しないために、HotSync でそのデータベースを見つけられない、ということになります。

この制限は、2 つのサンプル・アプリケーション (CustDB と Simple) では問題になりません。これは、共通のデータベース・スキーマを共有しているためです。ただし、CustDB データベースの同期時に、Simple サンプルも同期されてしまうという影響はあります。

この問題の解決については、「Palm 作成者 ID を登録する」 『Ultra Light - C/C++ プログラミング』を参照してください。

Ultra Light データベースへの接続

データベースのデータを操作するには、Ultra Light アプリケーションをデータベースに接続する必要があります。

接続を確立する最も簡単な方法は、次のとおりです。次の項では、この方法の応用について説明します。

```
var DatabaseMgr;
var Connection;
DatabaseMgr = CreateObject("iAnywhere.UltraLite.DatabaseManager.CustDB");
Connection = DatabaseMgr.openConnection("dbf=" + DatabaseMgr.directory + "¥¥mydb.udb");
```

Connection オブジェクトの使用

次に示す Connection オブジェクトのプロパティは、アプリケーションのグローバルな動作を管理します。

Connection オブジェクトの詳細については、「Connection クラス」 60 ページを参照してください。

- **コミット動作** デフォルトでは、Ultra Light アプリケーションは AutoCommit モードに設定されています。insert 文、update 文、delete 文はすべて、すぐにデータベースにコミットされます。Connection.AutoCommit を false に設定し、アプリケーションにトランザクションを構築します。AutoCommit を off に設定しコミットを実行すると、アプリケーションのパフォーマンスを直接向上できます。

  「commit メソッド」 62 ページを参照してください。

- **ユーザ認証** grantConnectTo メソッドと revokeConnectFrom メソッドを使用すると、アプリケーションのユーザ ID とパスワードをデフォルト値の DBA と sql から別の値に変更できます。

  「ユーザの認証」 31 ページを参照してください。

- **同期** 同期を管理するオブジェクトのセットは、Connection オブジェクトからアクセスできます。

  「データの同期」 32 ページを参照してください。

- **テーブル** Connection.getTable メソッドを使用して、Ultra Light テーブルにアクセスします。

  「getTable メソッド」 64 ページを参照してください。

ページ間の接続とアプリケーション状態の管理

JavaScript 変数のスコープは、Web ページ 1 つに制限されます。ほとんどの Web アプリケーションでは複数のページが要求されるため、アプリケーションのページ間でオブジェクトを永続させるメカニズムが必要です。

Ultra Light for M-Business Anywhere では、ULTable、ResultSet、PreparedStatement の各オブジェクトについて永続性を用意しています。これらのオブジェクトをページ間で永続させるには、オブジェクトの作成時にパラメータとして「**永続的な名前**」を指定します。それ以降のページで永続的な名前を使用できます。

接続オブジェクトをページ間で使用するには、各ページでその接続を開き直します。それには reOpen メソッドを使用する方法があります。各 Web ページで JavaScript ファイルをインクルードして設定を初期化することにより、各ページで open メソッドを指定するという方法もあります。この方法の例については、サンプル・ファイル *samples-dir¥UltraLiteForMBusinessAnywhere¥CustDB¥main.htm* と *samples-dir¥UltraLiteForMBusinessAnywhere¥Simple¥main_page.htm* を参照してください。

ページ間で接続を開き直すための要件により、Ultra Light アプリケーションに対してセキュリティ機能が提供されます。セキュリティ機能を使用すると、ページ間を移動するときにユーザにパスワードなどの情報を確認させることを要求できます。

別の Web ページで Ultra Light オブジェクトが不要な場合は、メモリを節約するために、アプリケーションではそのオブジェクトに対して close メソッドを発行する必要があります。

参照

- 「 reOpenConnection メソッド」 75 ページ
- 「 PreparedStatement クラス」 85 ページ
- 「 ResultSet クラス」 95 ページ
- 「 ULTable クラス」 152 ページ
- 「 PreparedStatement クラス」 85 ページ

M-Business Anywhere アプリケーションにおける永続的な名前

HTML では、制御が新しいページに移ると、古いページで割り付けられていた JavaScript オブジェクトへのすべてのハンドルが失われます。たとえば main.html には、M-Business Anywhere データベース接続オブジェクトがあります。

```
conn = dbMgr.openConnection("...");
```

main.html のリンクをクリックして *insert.html* などの別のページに移動すると、*insert.html* ではオブジェクト "conn" が見つかりません。この接続オブジェクトを取得し直すには、dbMgr.openConnection("...") をもう一度呼び出す必要がある可能性があります。ただし、接続オブジェクトはメモリ内にまだ存在するため、そのようにする必要はありません。そのオブジェクトに対する JavaScript ハンドルが失われたにすぎません。

DataManager、Connection、ULTable、PreparedStatement、ResultSet に対するすべての M-Business Anywhere API 呼び出しに persistName 引数が存在するのはこのためです。たとえば、M-Business Anywhere ランタイムが Ultra Light 接続オブジェクトに対する JavaScript からの呼び出しを受け取ると、M-Business Anywhere は、同じ persistName を持つ接続オブジェクトがメモリ内に存在するかどうかを最初にチェックします。一致するオブジェクトが見つかると、ランタイムはその接続オブジェクトを返します。見つからない場合は、M-Business Anywhere は通常の手順に移り、新しい Ultra Light データベース接続を作成して返します。

永続的な名前の使用

M-Business Anywhere オブジェクト間の階層には 2 種類あります。どちらの場合も、次のように DatabaseManager と Connection から始まります。

- DatabaseManager -> Connection -> Table (テーブル API の場合)
- DatabaseManager -> Connection -> PreparedStatement -> ResultSet (動的 SQL API の場合)

これらの M-Business Anywhere オブジェクトを永続的な名前で取得するには、最上位レベルのオブジェクトを永続的な名前で取得し、次に必要な M-Business Anywhere オブジェクトに至るまで、階層ツリーで上位レベルにあるすべての M-Business Anywhere オブジェクトを取得する必要があります。

たとえば、既存の ULTable オブジェクトを insert.html から取得する場合は、*main.html* で dbMgr、conn、table の各オブジェクトに永続的な名前を付けてから、*insert.html* で永続的な名前を使用してそれらすべてを取得します。

main.html のコード・セグメント：

```
var dbMgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.simple" );
// "simple" is the persistent name here. A real database manager object is allocated

var conn = dbMgr.openConnection( "CON=simple_con;..." );
// "simple_con" is the persistent name here. A real database connection is made.

var custTable = conn.getTable( "ULCustomer", "simpleCustTable" );
// a real table is allocated
```


insert.html のコード・セグメント：

```
var dbMgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.simple" );
// "simple" is the persistent name here.
// The allocated database manager object from main.html is returned

var conn = dbMgr.openConnection( "CON=simple_con;..." );
// "simple_con" is the persistent name here.
// The existing connection object from memory is returned.

var custTable = conn.getTable( "ULCustomer", "simpleCustTable" );
// the existing table object is returned.

var newTable = conn.getTable( "ULOrder", "simpleOrderTable" );
// since there is no order table from main.html,
// it does not exist in memory. A real order table object is allocated.
```

永続的な名前の適切な使用

共通で使用されるコードは、JavaScript ファイルに配置します。M-Business Anywhere アプリケーションのほとんどの HTML ページでは、DatabaseManager オブジェクト、Connection オブジェクト、主要な ULTable オブジェクトを参照する必要があるため、これらのオブジェクトを作成する (または、永続的な名前でこれらのオブジェクトを取得する) コードを共通の JavaScript ファイルに配置し、オブジェクトを使用する HTML ページの最初で、そのファイルをインクルードする方が便利です。M-Business Anywhere サンプル・プログラムの "Simple" と "CustDB" の両方で、その方法が示されています。

別のページからオブジェクトを使用する予定がない場合は、そのオブジェクトを閉じます。M-Business Anywhere アプリケーションに HTML ページが 1 つしかない場合は、永続的な名前を使用する必要はありません。永続的な名前の引数を NULL に設定することもできます。一方、各 HTML ページで PreparedStatement オブジェクトと ResultSet オブジェクトが多数開かれている場合は、開発者は、これらのオブジェクトをメモリ内に保持することで永続名を使用して別の HTML ページから簡単に取得できるという便利さと、これらのオブジェクトが常に存在するために生じるメモリの浪費とのバランスを取る必要があります。たとえば、5 個の PreparedStatement オブジェクトと 10 個の ResultSet オブジェクトが *main.html* で作成されたとします。これらはメモリを大量に占有しています。アプリケーションで *insert.html* に移動する場合、これらのオブジェクトのうち永続的な名前で参照する必要があるのが一部だけであれば、必要なくなったオブジェクトによってメモリが浪費されています。*insert.html* で新しく PreparedStatement オブジェクトと ResultSet オブジェクトを作成しようとすると、メモリが不足する可能性があります。これを解決するには、*insert.html* でこれらの PreparedStatment オブジェクトや ResultSet オブジェクトを必要としないことがわかっている場合は、*main.html* の最後に明示的にオブジェクトを閉じるようにします。

各 M-Business Anywhere オブジェクトの状態は、永続的な名前で取得される場合も保持されます。1 ページ目からの永続的な ULTable オブジェクトが存在する状態で、2 ページ目から同じ永続名を使用して openTable メソッドを呼び出すと、1 ページ目での状態と同じ状態でその ULtable オブジェクトを取得します。1 ページ目から移動するときにカーソルがテーブルで n 番目のローにある場合、2 ページ目でこのオブジェクトを取得したときも、カーソルは n 番目のローにあるままです。「最初のローの前」にカーソルが来ることはありません。

ResultSet で永続的な名前を使用する場合は注意してください。PreparedStatment にプレースホルダがある場合は、ResultSet に永続名を付けるかどうかについて慎重に考慮する必要があります。たとえば、*main.html* に次のコードがあるとします。

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" );

OrderStmt.setInt(1, 5000);

var OrderResultSet = OrderStmt.executeQuery( "order_query_result" );
```

insert.html で同じ ResultSet オブジェクトが必要な場合は、次のようにする必要があります。

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" );

//OrderStmt.setInt(1, 5000); // no need to do this since both the OrderStmt and
OrderResultSet are retrieve from "cache" without any SQL statement being
actually executed

var OrderResultSet = OrderStmt.executeQuery( "order_query_result" );
```

この OrderResultSet オブジェクトには、"order_id" が 5000 に設定された場合と同じ結果が格納されます。

しかし、別の状況も考えられます。Order テーブルで同じクエリを実行するために、同じ PreparedStatement が必要だとします。ただし、クエリでは 5000 以外の order_id を使用します。この場合、永続的な名前を PreparedStatement に割り当てることはできますが、ResultSet には永続的な名前は必要ありません。この状況では order_id が異なるため、前の例とは違う結果セットになります。*main.html* では、次のような同様のコードになります。

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" ); // with persistent name

OrderStmt.setInt(1, 5000);

var OrderResultSet = OrderStmt.executeQuery( null ); // notice here, no persistent name
```

insert.html では、次のようにして新しい ResultSet を取得します。

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" ); // get the prepared statement from memory with persistent name

OrderStmt.setInt(1, 6000); // set a different place holder value

var OrderResultSet = OrderStmt.executeQuery( null ); // a real query is executed
here!
```


この例では、プレースホルダの値が異なるか、Order テーブルで返される結果セットが異なると予期される他の操作が実行されるため、executeQuery の呼び出し時に ResultSet に永続的な名前を使用する必要がありません。

データベースの暗号化と難読化

Ultra Light for M-Business Anywhere を使用して、Ultra Light データベースを暗号化したり、難読化したりできます。

データベース暗号化の詳細については、「Ultra Light でのセキュリティの考慮事項」『Ultra Light - データベース管理とリファレンス』を参照してください。

暗号化

Ultra Light データベースは、暗号化しないことも、暗号化や難読化を適用することもできます。データベースを暗号化や難読化する場合は、データベースの作成時に選択する必要があります。

Ultra Light データベースの暗号化では、強力な業界標準技術を使用して、データベース内のデータを暗号化します。暗号化は、データベースの作成時に指定するキー・フレーズに基づいて行われます。このキー・フレーズは、データベースに接続するときにも指定する必要があります。

Ultra Light データベースを暗号化した場合、そのデータベースに接続するときに正しい暗号化キーを指定しなければ、接続に失敗します。

EncryptionKey プロパティの詳細については、「 ConnectionParms クラス」 68 ページと「 ChangeEncryptionKey メソッド」 61 ページを参照してください。

難読化

難読化とは、非常に弱い形態の暗号化のことで、ファイルやディスクの閲覧プログラムからデータベースの内容を不用意に閲覧されないように、単にデータベース内のデータをマスクします。データベースを難読化するには、creationParms.obfuscate ブール値を true に設定します。次に例を示します。

```
var create_parms = dbMgr.createCreationParms();
create_parms.obfuscate = true;
```

例

暗号化キーを変更するには、新しい暗号化キーを Connection オブジェクトで指定します。changeEncryptionKey メソッドを呼び出す前に、アプリケーションで既存の暗号化キーを使用して、暗号化されたデータベースに接続する必要があります。次のコード例では "apricot" が新しい暗号化キーです。

```
conn.changeEncryptionKey("apricot")
```


SQL を使用したデータの使用

Ultra Light アプリケーションは、SQL またはテーブル API を使用してテーブル・データにアクセスできます。この項では、SQL を使用したデータ・アクセスについて説明します。

テーブル API の詳細については、「テーブル API を使用したデータの使用」 23 ページを参照してください。

この項では、SQL を使用して次の操作を行う方法を説明します。

- ローの挿入、削除、更新
- クエリの実行
- 結果セットのローのスクロール

この項では、SQL 言語そのものについては説明しません。SQL 機能の詳細については、SQL Anywhere サーバ - SQL リファレンス 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

データ操作：INSERT、UPDATE、DELETE

Ultra Light では、SQL データ操作言語の操作や DDL 操作を実行できます。これらの操作は、PreparedStatement クラスのメンバである ExecuteStatement メソッドを使用して実行します。

PreparedStatement クラスの詳細については、「 PreparedStatement クラス」 85 ページを参照してください。

準備文のパラメータ・マーカ
Ultra Light は、パラメータ・マーカ '?' を使用して変数値を処理します。INSERT、UPDATE、DELETE で、準備文での並び順に従ってそれぞれの '?' が参照されます。たとえば、最初の '?' は 1、2 番目の '?' は 2 のようになります。

♦ ローを挿入するには、次の手順に従います。

1. PreparedStatement オブジェクトを宣言します。

   ```
   var PrepStmt;
   ```

2. INSERT 文を準備文オブジェクトに割り当てます。次のコードでは、TableName と ColumnName がテーブルとカラムの名前です。

   ```
   PrepStmt = conn.prepareStatement(
      "INSERT into TableName(ColumnName) values (?)", null );
   ```

   null パラメータは、文に永続的な名前がないことを示します。

3. その文にパラメータ値を割り当てます。

```
var NewValue;
NewValue = "Bob";
PrepStmt.setStringParameter(1, NewValue);
```

4. 文を実行します。

```
PrepStmt.executeStatement( null );
```

♦ **ローを更新するには、次の手順に従います。**

1. PreparedStatement オブジェクトを宣言します。

```
var PrepStmt;
```

2. UPDATE 文を準備文オブジェクトに割り当てます。次のコードでは、TableName と ColumnName がテーブルとカラムの名前です。

```
PrepStmt = conn.prepareStatement(
  "UPDATE TableName SET ColumnName = ? WHERE ID = ?", null);
```

null パラメータは、文に永続的な名前がないことを示します。

3. データ型に適切なメソッドを使用して、文にパラメータ値を割り当てます。

```
var NewValue;
NewValue = "Bob";
PrepStmt.setStringParameter(1, NewValue);
PrepStmt.setIntParameter(2, 6);
```

4. 文を実行します。

```
PrepStmt.executeStatement( );
```

♦ **ローを削除するには、次の手順に従います。**

1. PreparedStatement オブジェクトを宣言します。

```
var PrepStmt;
```

2. DELETE 文を準備文オブジェクトに割り当てます。

```
PrepStmt = conn.prepareStatement(
   "DELETE FROM customer WHERE ID = ?", null );
```

null パラメータは、文に永続的な名前がないことを示します。

3. その文にパラメータ値を割り当てます。

```
var IDValue;
IDValue = 6;
PrepStmt.setIntParameter( 1, IDValue );
```

4. 文を実行します。

```
PrepStmt.executeStatement( );
```


データ検索：SELECT

SELECT 文を実行すると、PreparedStatement.executeQuery メソッドは ResultSet オブジェクトを返します。ResultSet クラスには、結果セット内をナビゲーションするためのメソッドや、ResultSet を使用してデータを更新するためのメソッドが含まれています。

ResultSet オブジェクトの詳細については、「ResultSet クラス」 95 ページを参照してください。

例

次のコードでは、クエリの結果に ResultSet としてアクセスします。最初に割り当てられたとき、ResultSet は最初のローの前に配置されます。次に ResultSet.moveFirst メソッドが呼び出され、結果セットの最初のレコードをナビゲーションします。

```
var MyResultSet;
var PrepStmt;
PrepStmt = conn.prepareStatement("SELECT ID, Name FROM customer", null );
MyResultSet = PrepStmt.executeQuery( null );
MyResultSet.moveFirst();
```

例

次のコードは、現在のローのカラム値を取得する方法を説明します。この例では、文字データを使用します。その他のデータ型でも同様のメソッドを利用できます。

getString メソッドは構文 MyResultSetName.getString(*Index*) を使用します。*Index* は SELECT 文でのカラム名の並び順です。

```
if ( MyResultSet.getRowCount() == 0 ) {
} else {
  alert( MyResultSet.getString(1) );
  alert( MyResultSet.getString(2) );
  MyResultSet.moveRelative(0);
}
```

結果セットのナビゲーションの詳細については、「SQL を使用したナビゲーション」 21 ページを参照してください。

次の手順では、SELECT 文を使用して、データベースから情報を取り出します。クエリの結果は、ResultSet オブジェクトに割り当てられます。

♦ **SELECT 文を実行するには、次の手順に従います。**

1. PreparedStatement オブジェクトを宣言します。

   ```
   var OrderStmt;
   ```

2. 準備文を PreparedStatement オブジェクトに割り当てます。

   ```
   OrderStmt = Connection.prepareStatement(
     "SELECT order_id, disc, quant, notes, status, c.cust_id,
     cust_name, p.prod_id, prod_name, price
    FROM ULOrder o, ULCustomer c, ULProduct p
    WHERE o.cust_id = c.cust_id
    AND o.prod_id = p.prod_id
    ORDER BY der_id", "order_query_stmt" );
   ```


2 番目のパラメータは、ページ間 JavaScript オブジェクトの永続性を提供する永続的な名前です。

3. クエリを実行します。

```
OrderResultSet = OrderStmt.executeQuery( "order_query" );
```

クエリの使用方法の詳細については、*samples-dir¥UltraLiteForMBusinessAnywhere¥CustDB¥custdb.js* の CustDB サンプル・コードを参照してください。

SQL を使用したナビゲーション

Ultra Light for M-Business Anywhere は、幅広いナビゲーション作業を行うため、結果セットをナビゲーションする方法を多数提供します。

次の ResultSet オブジェクトのメソッドを使うと、結果セット内をナビゲーションできます。

- **moveAfterLast** 最後のローの後に移動します。
- **moveBeforeFirst** 最初のローの前に移動します。
- **moveFirst** 最初のローに移動します。
- **moveLast** 最後のローに移動します。
- **moveNext** 次のローに移動します。
- **movePrevious** 前のローに移動します。
- **moveRelative** いくつかのローを、現在のローを基準にして相対的に移動します。正のインデックス値は結果セット内を前に移動し、負のインデックス値は結果セット内を後ろに移動し、0 はカーソルを移動しません。ロー・バッファを再移植する場合は、0 が便利です。

例

次のコード・フラグメントは、moveFirst メソッドを使用して、結果セット内をナビゲーションする方法を説明します。

```
PrepStmt = conn.prepareStatement(
   "SELECT ID, Name FROM customer", null );
MyResultSet = PrepStmt.executeQuery( null );
MyResultSet.moveFirst();
```

すべての move メソッドで同じ方法を使用できます。

これらのナビゲーション・メソッドの詳細については、「ResultSet クラス」 95 ページを参照してください。

ResultSetSchema オブジェクト

ResultSet.schema プロパティを使うと、クエリのカラムに関する情報を取り出すことができます。

次の例は、ResultSetSchema を使用して、スキーマ情報を取得する方法を示しています。


```
var i;
var MySchema = rs.schema ;
for ( i = 1; i <= MySchema.columnCount; i++) {
  colname = MySchema.getColumnName(i);
  coltype = MySchema.getColumnSQLType(colname).toString();
  alert ( colname + " " + coltype );
}
```

テーブル API を使用したデータの使用

Ultra Light アプリケーションは、SQL またはテーブル API を使用してテーブル・データにアクセスできます。この項では、テーブル API を使用したデータ・アクセスについて説明します。

SQL の詳細については、「SQL を使用したデータの使用」 18 ページを参照してください。

この項では、テーブル API を使用して次の操作を行う方法について説明します。

- テーブルのローのスクロール
- 現在のローの値へのアクセス
- find メソッドと lookup メソッドを使用したテーブルのローの検索
- ローの挿入、削除、更新

テーブル API を使用したナビゲーション

Ultra Light for M-Business Anywhere は、幅広いナビゲーション作業を行うために、テーブルをナビゲーションする複数のメソッドを提供します。

次の ULTable オブジェクトのメソッドを使うと、結果セット内をナビゲーションできます。

- **moveAfterLast**　最後のローの後に移動します。
- **moveBeforeFirst**　最初のローの前に移動します。
- **moveFirst**　最初のローに移動します。
- **moveLast**　最後のローに移動します。
- **moveNext**　次のローに移動します。
- **movePrevious**　前のローに移動します。
- **moveRelative**　いくつかのローを、現在のローを基準にして相対的に移動します。正のインデックス値はテーブル内を前に移動し、負のインデックス値はテーブル内を後ろに移動し、0 はカーソルを移動しません。ロー・バッファを再移植する場合は、0 が便利です。

例

次のコードは、customer テーブルを開き、そのローをスクロールします。次に、各顧客の姓を示す警告を表示します。

```
var tCustomer;
tCustomer = conn.getTable( "customer", null );
tCustomer.open();
tCustomer.moveBeforeFirst();
While (tCustomer.moveNext()) {
  alert( tCustomer.getString(3) );
}
```

インデックスの指定

テーブル・オブジェクトを開くと、テーブルのローがアプリケーションに公開されます。デフォルトでは、ローはプライマリ・キー値の順に公開されますが、インデックスを指定すると特定の順序でローにアクセスできます。

例

次のコードは、ix_name インデックスで順序付けられた Customer テーブルの最初のローに移動します。

```
tCustomer = conn.getTable("customer", null );
tCustomer.openWithIndex("ix_name");
tCustomer.moveFirst();
```

現在のローの値へのアクセス

ULTable オブジェクトは、次のいずれかの位置に常に置かれています。

- テーブルの最初のローの前
- テーブルのいずれかのローの上
- テーブルの最後のローの後ろ

ULTable オブジェクトがローの上に置かれている場合は、ULTable の get メソッドを使用して、現在のローの各カラムの値を取得できます。

例

次のコード・フラグメントは、tCustomer ULTable オブジェクトから 3 つのカラムの値を取り出して、テキスト・ボックスに表示します。

```
var colID, colFirstName, colLastName;
colID = tCustomer.schema.getColumnID( "id" );
colFirstName = tCustomer.schema.getColumnID( "fname" );
colLastName = tCustomer.schema.getColumnID( "lname" );
alert( tCustomer.getInt( colID ) );
alert( tCustomer.getString( colFirstName ) );
alert( tCustomer.getString( colLastName ) );
```

値を設定するには、ULTable のメソッドを使用することもできます。

```
tCustomer.setString( colLastName, "Kaminski" );
```

これらのプロパティへの値の割り当てによって、データベース内のデータの値が変更されることはありません。

位置がテーブルの最初のローの前または最後のローの後ろにある場合でも、プロパティに値を割り当てることができます。しかし、カラムから値を取得することはできません。たとえば、次のコード・フラグメントはエラーを生成します。

```
tCustomer.moveBeforeFirst();
id = tCustomer.getInt( colID );
```

find と lookup を使用したローの検索

Ultra Light には、データを操作するための操作モードがいくつかあります。これらのモードのうちの 2 つ (検索モードとルックアップ・モード) は、検索に使用されます。ULTable オブジェクトには、テーブル内の特定のローを検索するために、これらのモードに対応するメソッドがあります。

注意
find メソッドや lookup メソッドを使用して検索されるカラムは、テーブルを開くのに使用されたインデックスにあることが必要です。

- **find メソッド** ULTable オブジェクトを開いたときに指定したソート順に基づいて、指定された検索値と正確に一致する最初のローに移動します。

  find メソッドの詳細については、「ULTable クラス」 152 ページを参照してください。

- **lookup メソッド** ULTable オブジェクトを開いたときに指定したソート順に基づいて、指定された検索値と一致するか、それより大きい値の最初のローに移動します。

  lookup メソッドの詳細については、「ULTable クラス」 152 ページを参照してください。

- **ローを検索するには、次の手順に従います。**

1. 検索モードまたはルックアップ・モードを開始します。

   FindBegin メソッドまたは LookupBegin メソッドを呼び出します。たとえば、次のコード・フラグメントは ULTable.findBegin を呼び出します。

   ```
   tCustomer.findBegin();
   ```

2. 検索値を設定します。

   検索値は、現在のローの値を設定することで設定します。これらの値を設定すると、バッファに影響しますが、データベースには影響しません。たとえば、次のコード・フラグメントは、バッファの姓のカラムを Kaminski に設定します。

   ```
   tCustomer.setString(3, "Kaminski" );
   ```

   マルチカラム・インデックスの場合は、最初のカラムの値が必要ですが、ほかのカラムは省略できます。

3. ローを検索します。

   適切なメソッドを使用して検索を実行します。たとえば、次の指示は、現在のインデックスで指定された値と正確に一致する最初のローを検索します。

   ```
   tCustomer.findFirst();
   ```


ローの挿入、更新、削除

Ultra Light は、テーブルのローを一度に 1 つずつアプリケーションに公開します。ULTable オブジェクトにはカレント・ポジションがあります。カレント・ポジションは、テーブルのローの上、最初のローの前、または最後のローの後ろになります。

アプリケーションがロケーションを変更すると、Ultra Light はバッファにそのローのコピーを作成します。値を取得または設定する操作はすべて、このバッファにあるデータのコピーにのみ影響します。データベースのデータには影響しません。

例

次の文は、バッファの最初のカラムの値を 3 に変更します。

```
tCustomer.setInt( 1 , 3 );
```

Ultra Light のモードの使用

Ultra Light モードによって、バッファ内の値を使用する目的が決まります。Ultra Light には、デフォルト・モードに加えて、次の 4 つの操作モードがあります。

- **挿入モード**　ULTable.insert メソッドを呼び出すと、バッファ内のデータが新しいローとしてテーブルに追加されます。
- **更新モード**　ULTable.update メソッドを呼び出すと、現在のローがバッファ内のデータに置き換えられます。
- **検索モード**　ULTable.find メソッドの 1 つが呼び出されたときに、値がバッファ内のデータに正確に一致するローの検索に使用されます。
- **ルックアップ・モード**　いずれかの ULTable.lookup メソッドが呼び出されたときに、バッファ内のデータと一致するか、それより大きい値のローを検索します。

- **ローを更新するには、次の手順に従います。**

1. 更新するローに移動します。

   テーブルをスクロールするか、find メソッドや lookup メソッドを使用して検索し、ローに移動できます。

2. 更新モードを開始します。

   たとえば、次の指示は、tCustomer テーブル上で更新モードを開始します。

   ```
   tCustomer.updateBegin();
   ```

3. 更新するローの新しい値を設定します。

   たとえば、次の指示は新しい値を Elizabeth に設定します。

   ```
   tCustomer.setString( 2, "Elizabeth" );
   ```

4. Update を実行します。

   ```
   tCustomer.update();
   ```

更新操作が終了すると、直前に更新したローが現在のローになります。ULTable オブジェクトを開いたときに指定したインデックスのカラム値を変更した場合は、現在の位置は不確定です。

デフォルトでは、Ultra Light は AutoCommit モードで動作するため、更新は永続的な記憶領域のローに即時適用されます。AutoCommit モードを無効にした場合は、コミット操作を実行するまで、更新は適用されません。AutoCommit モードの詳細については、「トランザクションの管理」 28 ページを参照してください。

警告
ローのプライマリ・キーを更新しないでください。代わりに、ローを削除して新しいローを追加してください。

ローの挿入

ローの挿入手順は、ローの更新手順とほぼ同じです。ただし、挿入操作の場合は、テーブル内の特定のローにあらかじめ指定する必要はありません。ローは、テーブルを開くときに使用したインデックスで自動的にソートされます。

♦ ローを挿入するには、次の手順に従います。

1. 挿入モードを開始します。

   たとえば、次の指示は、CustomerTable テーブル上で更新モードを開始します。

   ```
   tCustomer.insertBegin();
   ```

2. 新しいローの値を設定します。

   カラムの値を設定しない場合、そのカラムにデフォルト値があるときはデフォルト値が使用されます。カラムにデフォルト値がない場合は、NULL が使用されます。カラムが NULL を許可しない場合は、次のデフォルトが使用されます。

   - 数値カラムの場合は 0
   - 文字カラムの場合は空の文字列

   明示的に値を NULL に設定するには、setNull メソッドを使用します。

   ```
   colID = tCustomer.schema.getColumnID( "id" );
   colFirstName = tCustomer.schema.getColumnID( "fname" );
   colLastName = tCustomer.schema.getColumnID( "lname" );
   tCustomer.setInt( colID, 42 );
   tCustomer.setString( colFirstName, "Mitch" );
   tCustomer.setString( colLastName, "McLeod" );
   ```

3. 挿入を実行します。

   挿入されたローは、Commit を実行したときに永続的にデータベースに保存されます。AutoCommit モードでは、Insert メソッドの一部として Commit が実行されます。

   ```
   tCustomer.insert();
   ```


ローの削除

挿入モードや更新モードに対応する削除モードはありません。

次のプロシージャは、ローを削除します。

♦ **ローを削除するには、次の手順に従います。**

1. 削除するローに移動します。
2. 削除を実行します。

```
tCustomer.deleteRow();
```

BLOB データの処理

GetByteChunk メソッドを使用して、BINARY または LONG BINARY と宣言された、カラムの BLOB データをフェッチできます。

「 getStringChunk メソッド」 161 ページを参照してください。

トランザクションの管理

Ultra Light のトランザクション処理は、データベース内のデータの整合性を保証します。トランザクションは、作業の論理単位です。トランザクション全体が実行されるか、トランザクション内の文がどれも実行されないかのいずれかです。

デフォルトでは、Ultra Light は AutoCommit モードで動作します。AutoCommit モードでは、挿入、更新、削除はそれぞれ独立したトランザクションとして実行されます。操作が完了すると、データベースに変更が加えられます。

Connection.AutoCommit プロパティを false に設定すると、複数文のトランザクションを使用できます。たとえば、2 つの口座間で資金を移動するアプリケーションでは、振り込み元の口座からの引き落としと振り込み先口座への振り込みが、1 つのトランザクションを構成します。

AutoCommit が false に設定されている場合は、Connection.commit 文を実行してトランザクションを完了し、データベースへの変更を永続的なものにするか、Connection.rollback 文を実行してトランザクションのすべての処理をキャンセルしてください。AutoCommit をオフにすると、パフォーマンスが向上します。

> **注意**
> Autocommit を False に設定していても、同期はコミットを実行します。

スキーマ情報へのアクセス

Connection、ULTable、ResultSet オブジェクトにはそれぞれ、スキーマ・プロパティが含まれます。これらのスキーマ・オブジェクトは、データベース内のテーブル、カラム、インデックス、パブリケーションに関する情報を提供します。

- **DatabaseSchema**　データベース内のテーブルの数と名前、日付と時刻のフォーマットなどのグローバル・プロパティ。

  DatabaseSchema オブジェクトを取得するには、Connection.databaseSchema プロパティにアクセスします。

- **TableSchema**　テーブル内のカラムの数と名前、テーブルの Indexes コレクション。

  TableSchema オブジェクトを取得するには、ULTable.schema プロパティにアクセスします。

- **IndexSchema**　インデックス内のカラムに関する情報。インデックスには直接対応するデータがないので、個別の Index オブジェクトはなく、IndexSchema オブジェクトだけが存在します。

  IndexSchema オブジェクトは、TableSchema.getIndex メソッドを使用してアクセスできます。

- **PublicationSchema**　パブリケーションに含まれるテーブルとカラムの数と名前。パブリケーションもスキーマのみで構成されているため、Publication オブジェクトではなく、PublicationSchema オブジェクトが存在します。

  PublicationSchema オブジェクトは、DatabaseSchema.getPublicationSchema メソッドを使用してアクセスできます。

- **ResultSetSchema**　結果セットのカラムの数と名前。

  ResultSetSchema オブジェクトは、PreparedStatement.getResultSetSchema メソッドまたは ResultSet.schema プロパティを使用してアクセスできます。

エラー処理

通常の操作では、Ultra Light for M-Business Anywhere はスクリプト環境でキャッチされて処理されることを意図したエラーをスローすることがあります。「SQLError クラス」 115 ページを参照してください。

エラーは SQLCODE 値として表現され、負の数字は特定の種類のエラーを示します。

Ultra Light for M-Business Anywhere は、DatabaseManager オブジェクトと Connection オブジェクトからのみエラーをスローします。DatabaseManager の次のメソッドは、エラーをスローできます。

- createDatabase
- dropDatabase
- openConnection

Ultra Light for M-Business Anywhere 内での他のエラーや例外はすべて、Connection オブジェクトを経由します。

エラー番号には DatabaseManager オブジェクトと Connection オブジェクトからアクセスできます。次の項を参照してください。

- 「Connection クラス」 60 ページ
- 「DatabaseManager クラス」 73 ページ

ユーザの認証

新しいユーザは既存の接続から追加します。Ultra Light のすべてのデータベースは、デフォルトのユーザ ID DBA とパスワード sql を使用して作成されるため、まずは初期ユーザとして接続します。

ユーザ ID の変更はできません。ユーザを 1 人追加して既存のユーザを削除します。Ultra Light ではデータベースごとにユーザ ID が 4 つまで許可されます。

接続権限の付与または取り消しの詳細については、「grantConnectTo メソッド」 64 ページと「revokeConnectFrom メソッド」 65 ページを参照してください。

♦ **ユーザを追加する、または既存のユーザのパスワードを変更するには、次の手順に従います。**

1. 既存のユーザとしてデータベースに接続します。
2. 希望するパスワードでユーザに接続権限を付与します。

```
conn.grantConnectTo("Robert", "newPassword");
```

♦ **既存のユーザを削除するには、次の手順に従います。**

1. 既存のユーザとしてデータベースに接続します。
2. 次のように、ユーザの接続権限を取り消します。

```
conn.revokeConnectFrom("Robert");
```


データの同期

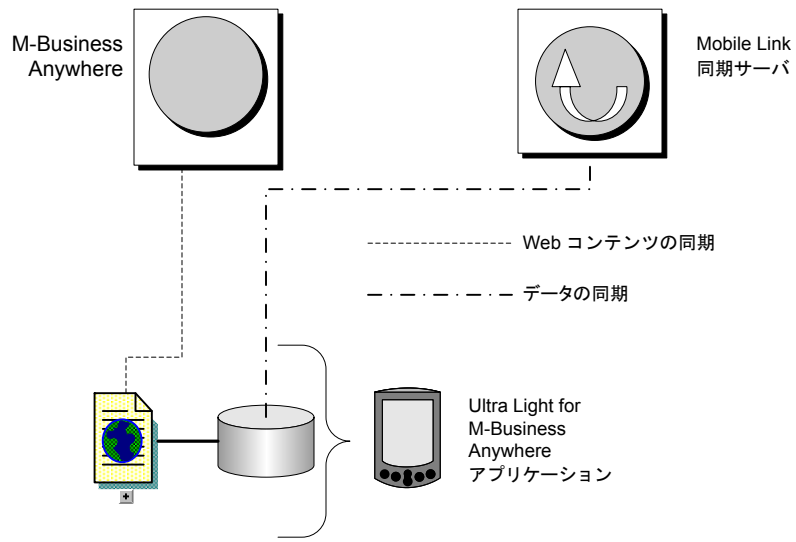
Ultra Light for M-Business Anywhere アプリケーションには、一般に 2 種類の同期があります。

- **Web コンテンツの同期**　Web コンテンツ (アプリケーション自体を定義する HTML ページを含む) が M-Business Anywhere で同期されます。
- **データの同期**　Ultra Light データベースが Mobile Link サーバと同期されます。

これら 2 種類の同期は異なりますが、「**ワンタッチ同期**」という手法を使用して一緒に開始することができます。ワンタッチ同期は、ほとんどのアプリケーションで推奨されるモデルです。ただし、データと Web コンテンツの同期を完全に分離しておく必要のある場合もあり、その場合の手法については後述します。

ワンタッチ同期

ワンタッチ同期は、Web コンテンツの同期 (M-Business Anywhere を使用) と Ultra Light データの同期 (Mobile Link を使用) を 1 回の操作で開始するための手法です。この手法は、Windows CE と Windows でのみ利用できます。ワンタッチ同期のアーキテクチャは、次のとおりです。



ワンタッチ同期では、次のような一連のイベントが発生します。

1. ユーザは、クレードルに置くなどして Web アプリケーションを同期します。
2. M-Business Client は、Web コンテンツを同期します。
3. M-Business Client の MBConnect コンポーネントは、*ulconnect.exe* アプリケーションを呼び出します。
4. *ulconnect.exe* は、Ultra Light データベースの同期を開始します。

5. データが Mobile Link と同期されます。

ワンタッチ同期を実装するには、次の手順を実行します。

1. アプリケーションで、Mobile Link 同期用の同期パラメータを設定します。

   M-Business Anywhere を使用して同期を行う場合は、SyncParms.setMBAServer メソッドを使用してホストとポートの同期パラメータを設定できます。「setMBAServer メソッド」 132 ページを参照してください。

   そうでない場合は、標準の方法で同期パラメータを設定してください。「SyncParms クラス」 126 ページを参照してください。

2. *ulconnect.exe* で読み込めるように、同期パラメータを保存します。

   Connection.saveSyncParms メソッドを呼び出して、同期パラメータを保存します。「saveSyncParms メソッド」 66 ページを参照してください。

データの同期

ほとんどのユーザは、データの同期と Web コンテンツの同期の両方を開始する ワンタッチ同期を使用する方が便利です。詳細については、「ワンタッチ同期」 32 ページを参照してください。

この項は、Web コンテンツの同期と独立してデータを同期したいユーザを対象としています。

同期には、Mobile Link サーバと適切なライセンスが必要です。CustDB サンプル・アプリケーションには、同期の実例もあります。

Ultra Light for M-Business Anywhere は、TCP/IP、HTTP、HTTPS、HotSync 同期をサポートしています。同期は、Ultra Light アプリケーションによって開始されます。いずれの場合でも、Connection オブジェクトのメソッドとプロパティを使用して同期を制御します。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 承認の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。
「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 10 - 紹介』を参照してください。

♦ TCP/IP または HTTP で同期するには、次の手順に従います。

1. 同期情報を準備します。

   Connection.syncParms オブジェクトの必須プロパティに値を割り当てます。

   設定するプロパティと値の詳細については、「Ultra Light クライアント」 『Mobile Link - クライアント管理』を参照してください。

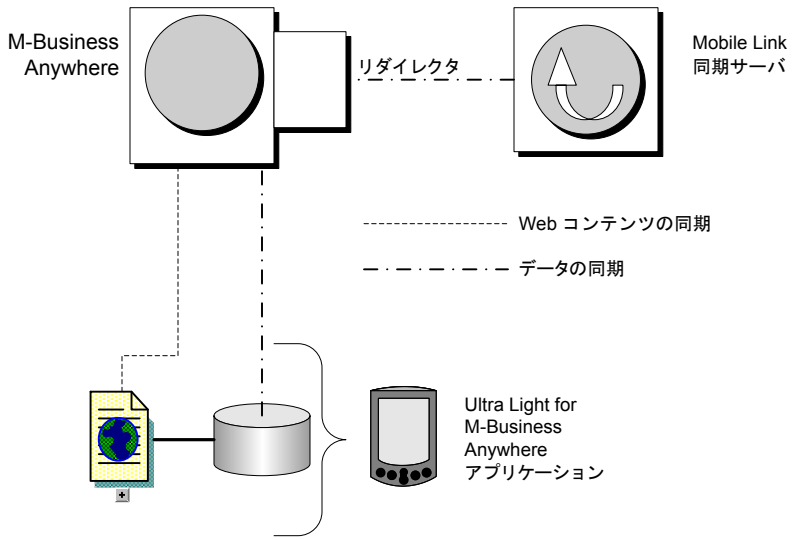
2. 同期を実行します。

   Connection.synchronize メソッドを呼び出します。

M-Business Anywhere を使用したデータの同期

ワンタッチ同期と個別のデータ同期のどちらを使用するかに関わらず、M-Business Anywhere Server が Mobile Link サーバとの間でデータを送受信するように設定するために、Mobile Link リダイレクタを使用することができます。ファイアウォールの外部からの同期の場合、これによって、外部からアクセス可能であることが必要なポート数が少なくて済みます。

次の図は、ワンタッチ同期の場合のアーキテクチャを示しています。



♦ **M-Business Anywhere を使用してデータを同期するには、次の手順に従います。**

1. サーバ側で、M-Business Anywhere と Mobile Link サーバとの間でデータを送受信するように Mobile Link リダイレクタを設定します。「M-Business Anywhere リダイレクタ」 『Mobile Link - サーバ管理』を参照してください。
2. クライアントで同期パラメータを設定して、Ultra Light 同期が M-Business Anywhere のホストとポート番号に送信されるようにします。SyncParms.setMBAServer メソッド使用してこの作業を行うことができます。「 setMBAServer メソッド」 132 ページを参照してください。
3. クライアント・アプリケーションから、ワンタッチ同期または個別のデータ同期を使用して、同期を開始します。次の項を参照してください。

   ♦ 「ワンタッチ同期」 32 ページ
   ♦ 「データの同期」 33 ページ

Ultra Light for M-Business Anywhere アプリケーションの配備

アプリケーションが完了したら、またはアプリケーションをテストしたい場合、アプリケーションをデバイスに配備する必要があります。この項では、デバイスに Ultra Light アプリケーションを配備するための手順について説明します。

Windows CE と Windows デスクトップへのアプリケーションの配備

Ultra Light アプリケーションを Windows CE デバイスに配備するには、次の手順を実行する必要があります。

- アプリケーションと Ultra Light コンポーネントを配備します。「Ultra Light for M-Business Anywhere クイック・スタート」 6 ページを参照してください。
- Ultra Light データベースの初期コピーを配備します。「Ultra Light for M-Business Anywhere クイック・スタート」 6 ページを参照してください。

  多くの場合、Ultra Light データベースを配備すれば十分です。このため、同期を使用してデータの初期コピーをロードできます。

  データベースは、アプリケーションが特定できるロケーションに配備する必要があります。Database On CE 接続パラメータは、Windows CE 用のロケーションを定義します。Database on Desktop 接続パラメータは、Windows 用のロケーションを定義します。次の項を参照してください。

  - 「Ultra Light CE_FILE 接続パラメータ」 『Ultra Light - データベース管理とリファレンス』
  - 「Ultra Light DBF 接続パラメータ」 『Ultra Light - データベース管理とリファレンス』.

ワンタッチ同期を使用するアプリケーションの配備

ワンタッチ同期では、*ulconnect.exe*、*ulconnect.udb*、*ulpod10.dll*、*ulrt10.dll* を含む一連のファイルが必要です。Windows CE の場合、これらのファイルはディレクトリ *install-dir¥ultralite¥UltraLiteForMBusinessAnywhere¥ce¥arm¥* のファイル ulpod.cab 内にあります。この cab ファイルを Windows CE デバイスに配備すると、cab ファイルの内容が適切なロケーションに自動的にインストールされます。Windows の場合、必要なファイルは、ディレクトリ *install-dir¥ultralite¥UltraLiteForMBusinessAnywhere¥win32¥386¥* から手動で配備する必要があります。

Palm OS へのアプリケーションの配備

Ultra Light アプリケーションを Palm OS デバイスに配備するには、次の手順を実行する必要があります。

- アプリケーションと Ultra Light コンポーネントを配備します。「Ultra Light for M-Business Anywhere クイック・スタート」 6 ページを参照してください。

- Ultra Light データベースの初期コピーを配備します。「Ultra Light for M-Business Anywhere クイック・スタート」 6 ページを参照してください。

  多くの場合、適切に初期化された Ultra Light データベース・ファイルのみを配備すれば十分です。このため、同期を使用してデータの初期コピーをロードできます。

  Palm OS へ配備する *.pdb* ファイルは、ulxml と ulinit を含む任意の Ultra Light ユーティリティを使用して作成できます。

  データベースは、アプリケーションがロケーションを特定できるように正しい作成者 ID を使用して指定する必要があります。Database On Palm 接続パラメータは、作成者 ID を使用してデータベースを検索します。「Ultra Light PALM_FILE 接続パラメータ」 『Ultra Light - データベース管理とリファレンス』を参照してください。

- HotSync 用 Mobile Link 同期コンジットを配備します。

  この手順が必要なのは、HotSync を使用してアプリケーションを同期する場合のみです。「Palm OS の HotSync」 『Mobile Link - クライアント管理』を参照してください。

第 3 章

チュートリアル：M-Business Anywhere 用のサンプル・アプリケーション

目次

M-Business Anywhere の開発チュートリアルの概要

このチュートリアルでは、プラットフォームを問わない Ultra Light アプリケーションを構築する方法について説明します。チュートリアルを終了すると、アプリケーションと小規模なデータベースが構築されます。これを、中央の統合データベースと同期します。

所要時間

このチュートリアルは、コードをコピーして貼り付ける場合、約 30 分で終了します。自分でコードを入力する場合、これより長い時間が必要です。

前提条件

このチュートリアルは、コンピュータに M-Business Anywhere がすでにインストールされていること、ファイルを配信するために使用できる Web サーバが用意されていることを前提としています。

アプリケーションをテストおよび実行するために、M-Business Client にアクセスできる必要もあります。

また、JavaScript プログラミング言語や M-Business Anywhere アプリケーション開発についての基本的な知識が必要です。

チュートリアルでは、Sybase Central で Ultra Light を使用するか、Ultra Light ユーティリティ ulcreate を使用して、Ultra Light データベースを作成する方法を理解していることも前提としています。

参照

- 「Sybase Central からの Ultra Light データベースの作成」『Ultra Light - データベース管理とリファレンス』
- 「Ultra Light データベースの作成と設定」『Ultra Light - データベース管理とリファレンス』

レッスン 1：プロジェクト・アーキテクチャの作成

最初のレッスンでは、プロジェクト・アーキテクチャを設定し、チュートリアル用の Ultra Light データベースを作成する方法を説明します。

♦ プロジェクト・アーキテクチャと空の Ultra Light データベースを作成するには、次の手順に従います。

1. このチュートリアル用のディレクトリを作成します。

   このチュートリアルでは、保存先ディレクトリを *c:¥Tutorial¥mbus* とします。別の名前のディレクトリを作成した場合は、チュートリアルを通じてそのディレクトリを使用してください。

   プラットフォーム固有のファイル用に、次のサブディレクトリを作成します。

   - *c:¥Tutorial¥mbus¥PALM_OS*
   - *c:¥Tutorial¥mbus¥WIN32_OS*
   - *c:¥Tutorial¥mbus¥WINCE_OS*
   - *c:¥Tutorial¥mbus¥WINCE_OS¥arm*

2. Web サーバを設定します。

   a. Web サーバ上の仮想ディレクトリ tutorial を *c:¥Tutorial¥mbus* にマップします。このディレクトリにアクセスするための URL は、*http://localhost/tutorial* になります。

      Microsoft IIS の場合は、管理ツールを使用してこの変更を行えます。

      Apache の場合は、ドキュメント・ルートから *c:¥Tutorial¥mbus* ディレクトリへのシンボリック・リンク *tutorial* を作成するか、または Apache ドキュメント・ルートにチュートリアル・ファイルをコピーします。

   b. Web サーバが次のファイルを MIME タイプ application/octet-stream で配信することを確認します。

      - *.cab*
      - *.dll*
      - *.prc*
      - *.pdb*
      - *.udb*

      Microsoft IIS の場合は、管理ツールを使用してこの変更を行えます。仮想ディレクトリのプロパティに移動し、[HTTP ヘッダー] と [ファイルの種類] で変更します。

      Apache の場合は、*conf* ディレクトリ内のファイル *mime.types* を編集します。

3. Sybase Central で Ultra Light を使用して、データベースを作成します。

   データベースの作成については、「Sybase Central からの Ultra Light データベースの作成」『Ultra Light - データベース管理とリファレンス』を参照してください。

- **テーブル名**　Customer
- **Customer のカラム**

カラム名	データ型 (サイズ)	カラムの NULL 値の許可	デフォルト値
ID	integer	いいえ	オートインクリメント
FName	char (15)	いいえ	なし
LName	char (20)	いいえ	なし
City	char (20)	はい	なし
Phone	char (12)	はい	555-1234

通常、同期している環境では、グローバル・オートインクリメントまたは UUID 値をプライマリ・キーに使用する方が適しています。ここでは、チュートリアルを簡潔にするため、デフォルトのオートインクリメントを使用しています。

- **プライマリ・キー**　昇順 ID

4. データベースを保存します。

Windows または Windows CE 用のアプリケーションを開発している場合は、[ファイル] – [保存] を選択し、ファイル名としてチュートリアル・ディレクトリの *WINCE_OS* または *WIN32_OS* サブディレクトリ内の *tutcustomer.udb* を選択します。

Palm OS のアプリケーションを開発している場合は、次の手順に従います。

a. [ファイル] - [Palm 用にスキーマをエクスポート] を選択します。

b. [作成者 ID] に Syb3 と入力します。

c. チュートリアル・ディレクトリの *PALM_OS* サブディレクトリに、そのファイルを *tutcustomer.pdb* として保存します。

Palm 作成者 ID に関する注意
作成者 ID は、Palm によってユーザに割り当てられます。サンプル・アプリケーションを作成する場合は、Syb3 を作成者 ID として使用できます。ただし、運用アプリケーションを作成する場合は、独自の作成者 ID を使用してください。

プラットフォームを問わないアプリケーションを開発している場合は、上記のロケーションすべてにデータベース・ファイルを保存します。

レッスン 2：アプリケーション・ファイルの作成

次の手順は、フォームを使用してユーザ・インタフェースを作成します。この例では、入力用と出力用にテキスト・ボックスを使用します。

♦ アプリケーション・ファイルの作成

1. ファイル *c:¥Tutorial¥mbus¥main.html* を作成します。

   このファイルは、アプリケーションのメイン・ファイルになります。チュートリアルの後半で、このファイルにコンテンツを追加します。ここでは、プラットフォーム固有のファイル *ul_deps.html* をインクルードするように設定します。次の内容をファイルに追加します。

   ```
   <html>
   <body>
   <a href="AG_DEVICEOS/ul_deps.html"></a>
   </body>
   </html>
   ```

2. プラットフォーム固有のファイルを作成します。

   これらのファイルはそれぞれ適切な Ultra Light ランタイム・ライブラリとデータベース・ファイルを参照します。次のようにして、チュートリアル・ディレクトリ内にあるオペレーティング・システムのサブディレクトリごとにファイル *ul_deps.html* を作成します。

   ```
   <!-- PALM_OS¥ul_deps.html -->
   <html>
     <a href="ulpod10.prc"></a>
     <a href="tutCustomer.pdb"></a>
   </html>

   <!-- WINCE_OS¥ul_deps.html -->
   <html>
     <a href="AG_DEVICEPROCESSOR/ulpod10.dll"></a>
     <a href="tutCustomer.udb"></a>
   </html>

   <!-- WIN32_OS¥ul_deps.html -->
   <html>
     <a href="ulpod10.dll"></a>
     <a href="tutCustomer.udb"></a>
   </html>
   ```

3. Ultra Light ランタイム・ファイルをチュートリアル・ディレクトリにコピーします。

   ul_deps.html ファイルでは、Ultra Light ランタイム・ライブラリとデータベースがチュートリアル・ディレクトリに対して適切なロケーションに存在していることが必要です。データベース・ファイルは、このチュートリアルの始めのほうで配置済みです。ここでは、Ultra Light ランタイム・ライブラリを適切なロケーションにコピーしてください。

   - Palm OS の場合は、*ulpod10.prc* を *install-dir¥ultralite¥UltraLiteForMBusinessAnywhere¥palm¥68k* から *c:¥Tutorial¥mbus¥PALM_OS¥* にコピーします。
   - Windows CE の場合は、*ulpod10.dll* を *install-dir¥ultralite¥UltraLiteForMBusinessAnywhere¥ce¥arm* から *c:¥Tutorial¥mbus¥WINCE_OS¥arm¥* にコピーします。

- Windows デスクトップの場合は、*ulpod10.dll* を *install-dir¥ultralite ¥UltraLiteForMBusinessAnywhere¥win32¥386* から *c:¥Tutorial¥mbus¥WIN32_OS¥* にコピーします。

すべてのアプリケーション・ファイルが配置されました。

レッスン 3：M-Business Anywhere サーバとクライアントの設定

このレッスンでは、アプリケーションで使用する M-Business Anywhere ユーザ、グループ、チャネルを作成します。この情報は、M-Business Anywhere 6.5 用です。

♦ M-Business Anywhere の設定

1. M-Business Anywhere 管理コンソールを開き、管理ユーザとしてログインします。

   デフォルトのユーザ ID は **Admin** で、パスワードは空です。

2. 新しいユーザを作成します。

   このチュートリアルの後半では、この手順で作成するユーザ名とパスワードを使用して、M-Business Client から同期を実行します。このサーバ用に設定された M-Business Client がすでに用意されている場合は、既存のユーザ名を使用することもできます。

   a. 左のナビゲーション・ウィンドウ枠で [Users] メニュー・オプションをクリックし、[Create User] リンクをクリックします。[Create User] ページが表示されます。

   b. [User Name] フィールドにユーザ名を入力し、[Password] フィールドと [Confirm Password] フィールドに同じパスワードを入力します。その他のフィールドはオプションです。[Create] をクリックします。

3. グループとチャネルを作成します。

   a. [Groups] 見出しをクリックし、[Create Group] をクリックします。

   b. 新しいグループに **UltraLite Samples** という名前を付け、[Create] をクリックします。

   c. [Edit Group] - [Channels] で [Create] をクリックします。

   d. チャネルに次の設定を使用します。お使いの Web ブラウザに応じて適切な URL に置き換えてください。

      - **[Title]** Ultra Light チュートリアル
      - **[Location]** *http://localhost:8091/tutorial/main.html*.

        このロケーションは、Web サーバがサービスを行う、チュートリアルの *main.html* ページの URL です。
      - **[Channel Size Limit]** 1000 KB
      - **[Link Depth]** 3
      - **[Allow Binary Distribution]** チェックする
      - **[Hide From Users]** チェックしない

4. ユーザをグループに追加します。

a. [Users] - [List User] をクリックし、手順 2 で作成したユーザを見つけます。

b. [User Name] をクリックして、ユーザのプロパティを表示します。

c. [Edit User] - [Channels] - [Groups] をクリックします。

d. **_UltraLite Samples_** グループをオンにし、[Save] をクリックしてユーザをこのグループに追加します。

ユーザ、グループ、チャネルが M-Business Anywhere に設定されました。次の手順では、このチャネルの内容を M-Business Client に同期します。これは、使用するどのプラットフォームからでも実行できます。

次の手順では、M-Business Client がインストール済みであることを前提としています。[ツール] – [オプション] をクリックし、[JavaScript エラーを表示する] クライアント・オプションを設定することをおすすめします。このように設定することで、アプリケーションで発生するあらゆるエラーのデバッグが簡単になります。

♦ デバイス用のチャネルの同期

· M-Business Client を、M-Business Anywhere Server 上の Ultra Light チャネルと同期します。

この時点では、アプリケーションのコンテンツはないため、ブランクのページが表示されます。

レッスン 4：アプリケーションへの起動コードの追加

このレッスンでは、Ultra Light データベースに接続するための起動コードをアプリケーションに追加します。このために、HTML をメイン・ページに追加し、アプリケーションを制御するための JavaScript 論理を追加します。

♦ アプリケーションへのコンテンツの追加

1. コンテンツを *main.html* に追加します。

   次のフォームをアプリケーションのメイン・ページ *c:¥Tutorial¥mbus¥main.html* で <a> タグの直後に追加します。

```
<form name="custForm">
<input type="text" name="fname" size="15"><br>
<input type="text" name="lname" size="20"><br>
<input type="text" name="city" size="20"><br>
<input type="text" name="phone" size="12"><br>
<input type="text" name="custid" size="10"><br>
<br><br>


<table>
   <tr>
 <td><input type="button"
     value="Insert" onclick="ClickInsert();">
 </td>
 <td><input type="button"
     value="Update" onclick="ClickUpdate();">
 </td>
 <td>
          <input type="button"
    value="Delete" onclick="ClickDelete();">
 </td>
  </tr>

 <tr>
 <td>
  <input type="button"
   value="Next" onclick="ClickNext();">
 </td>
 <td>
  <input type="button"
   value="Prev" onclick="ClickPrev();">
 </td>
 <td></td>
  </tr>

 <tr>
  <td colspan=3>
       <input type="button"
   value="Synchronize" onclick="ClickSync();">
  </td>
   </tr>
</table>
</form>
```

2. アプリケーションの論理を提供する JavaScript ファイル *c:¥Tutorial¥mbus¥tutorial.js* を作成します。

3. コンテンツを JavaScript ファイルに追加します。

   必要な Ultra Light オブジェクトを宣言する次のコードをファイルの先頭に追加します。

```
var DatabaseMgr;
var Connection;
var CustomerTable;
```

   接続コードを追加します。

```
function Connect()
{
  DatabaseMgr = CreateObject( "iAnywhere.Data.UltraLite.DatabaseManager.Tutorial" );
  if( DatabaseMgr == null ) {
    alert( "Error, make sure POD is on device" );
    return;
  }

  var connParms = DatabaseMgr.createConnectionParms();
  var dir = DatabaseMgr.directory;

  connParms.schemaOnPalm = "tutCustomer";
  connParms.databaseOnPalm = "Syb3";

  connParms.databaseOnCE = dir + "¥¥tutCustomer.udb";
  connParms.databaseOnDesktop = dir + "¥¥tutCustomer.udb";

  connParms.userID = "DBA";
  connParms.password = "sql";

  try {
    // try to connect to an existing database
    Connection = DatabaseMgr.openConnection( connParms.ToString() );
    alert("Connected to an existing database");
  } catch( ex ) {
    if( DatabaseMgr.sqlCode !=
DatabaseMgr.SQLError.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
      alert( ex.getMessage() );
      return;
    } else {
      try {
        // the database does not exist, create one
        Connection = DatabaseMgr.createDatabase( connParms.ToString() );
        alert("Created a new database");
      } catch( ex2 ) {
        alert( ex2.getMessage() );
        return;
      }
    }
  }
}
```

4. onload イベント・ハンドラを使用して、アプリケーションの起動時にデータベースに接続します。

   a. 次の行を <body> タグの直前に追加して、*tutorial.js* を *main.html* にインポートします。

```
<script src="tutorial.js"></script>
```

   b. *main.html* を編集して、<body> タグを次のように変更します。


```
<body onload="Connect();">
```

5. アプリケーションをテストします。

   M-Business Client を同期し、アプリケーションを起動します。アプリケーションによって Ultra Light データベースが作成されるときに、メッセージ・ボックスが表示されます。正常に動作したら、次のレッスンに進むことができます。

レッスン 5：アプリケーションへの挿入の追加

このレッスンでは、アプリケーションにデータ操作やナビゲーション論理を追加する方法を説明します。

♦ テーブルを開く

1. データベースの Customer テーブルを表す CustomerTable を初期化するコードを記述します。

   次のコードを *tutorial.js* の Connect 関数の末尾に追加します。

```
try {
  CustomerTable = Connection.getTable( "customer", null );
  CustomerTable.open();
} catch( ex3 ) {
  alert("Error: " + ex3.getMessage() );
}
```

2. データベースと Web フォームの間でデータを移動するための変数を追加します。

   次の宣言を *tutorial.js* の先頭で Connect 関数の前に追加します。

```
var Cust_FName = "";
var Cust_LName = "";
var Cust_City = "";
var Cust_Phone = "";
var Cust_Id = "";
```

3. 顧客のデータをフェッチして表示するためのプロシージャを作成します。

   次の関数を *tutorial.js* の Connect 関数の直後に追加します。この関数は、顧客の現在のローをフェッチして、NULL カラムが空の文字列として表示されるようにします。

```
function FetchCurrentRow()
{
  var id;
  if( CustomerTable.getRowCount() == 0 ) {
    Cust_FName = "";
    Cust_LName = "";
    Cust_City = "";
    Cust_Phone = "";
    Cust_Id = "";
  } else {
    id = CustomerTable.schema.getColumnID("FName");
    Cust_FName = CustomerTable.getString(id);
    id = CustomerTable.schema.getColumnID("LName");
    Cust_LName = CustomerTable.getString(id);
    id = CustomerTable.schema.getColumnID("city");
    if( CustomerTable.isNull(id) ) {
      Cust_City = "";
    } else {
      Cust_City = CustomerTable.getString(id);
    }
    id = CustomerTable.schema.getColumnID("phone");
    if( CustomerTable.isNull(id) ) {
      Cust_Phone = "";
    } else {
      Cust_Phone = CustomerTable.getString(id);
    }
```

```
    id = CustomerTable.schema.getColumnID("id");
    Cust_Id = CustomerTable.getString(id);
  }
}
```

次の JavaScript コードを *main.html* の終了 </body> タグの直前に追加します。DisplayCurrentRow は、データベースから値を取得して、Web フォームに表示します。FetchForm は、Web フォームの現在の値を取得して、データベース・コードで利用できるようにします。

```
<script>
function DisplayCurrentRow()
{
  FetchCurrentRow();
  document.custForm.fname.value = Cust_FName;
  document.custForm.lname.value = Cust_LName;
  document.custForm.city.value = Cust_City;
  document.custForm.phone.value = Cust_Phone;
  document.custForm.custid.value = Cust_Id;
}

function FetchForm()
{
  Cust_FName = document.custForm.fname.value;
  Cust_LName = document.custForm.lname.value;
  Cust_City = document.custForm.city.value;
  Cust_Phone = document.custForm.phone.value;
}
</script>
```

4. アプリケーションのロード時に DisplayCurrentRow を呼び出します。

   main.html の先頭の <body> タグを次のように変更します。

   ```
   <body onload="Connect(); DisplayCurrentRow();">
   ```

データベースにはデータがないためにローは表示されませんが、M-Business Client を同期してバグが生じていないことを確認するにはよいタイミングです。

♦ ローを挿入するコードの追加

- [Insert] ボタンを実装するためのコードを記述します。

  次のプロシージャでは、InsertBegin を呼び出すと、アプリケーションが挿入モードになり、現在のローのすべての値がデフォルトに設定されます。たとえば、ID カラムは、次のオートインクリメント値を受け取ります。カラム値が設定されると、新しいローが挿入されます。

  次の関数を *tutorial.js* の FetchCurrentRow の直後に追加します。

  ```
  function Insert()
  {
    var id;

    try {
      CustomerTable.insertBegin();
      id = CustomerTable.schema.getColumnID("FName");
      CustomerTable.setString(id, Cust_FName);
      id = CustomerTable.schema.getColumnID("LName");
      CustomerTable.setString(id, Cust_LName);
  ```



```
    id = CustomerTable.schema.getColumnID("city");
    if( Cust_City.length > 0 ) {
      CustomerTable.setString(id, Cust_City);
    }
    id = CustomerTable.schema.getColumnID("phone");
    if( Cust_Phone.length > 0 ) {
      CustomerTable.setString(id, Cust_Phone);
    }
    CustomerTable.insert();
    CustomerTable.moveLast();
  } catch( ex ) {
    alert( "Insert error: " + ex.getMessage() );
  }
}
```

次の関数を *main.html* の FetchForm 関数の直後に追加します。

```
function ClickInsert()
{
  FetchForm();
  Insert();
  DisplayCurrentRow();
}
```

これで、アプリケーションをテストできるようになりました。

♦ アプリケーションのテスト

1. M-Business Client を同期します。
2. アプリケーションを実行します。

   最初のメッセージ・ボックスの後にフォームが表示されます。
3. テーブルにローを 2 つ挿入します。

   a. 先頭のテキスト・ボックスに名前 Jane を入力し、2 つ目のテキスト・ボックスに姓 Doe を入力します。[Insert] をクリックします。

      テーブルに、これらの値を持つローが追加されます。アプリケーションはテーブルの最後のローに移動し、そのローを表示します。ラベルには、Ultra Light がローに割り当てた ID カラムの自動的にインクリメントされた値が表示されます。

   b. 先頭のテキスト・ボックスに名前 John を入力し、2 つめのテキスト・ボックスに姓 Smith を入力します。[Insert] をクリックします。

次の手順では、ナビゲーション・ボタンを追加します。

レッスン 6：アプリケーションへのナビゲーションの追加

このレッスンでは、結果セットのローを前後にスクロールするためのコードを説明します。

♦ アプリケーションへのナビゲーション・コードの追加

1. MoveNext 関数を *tutorial.js* の Insert 関数の直後に追加します。

```
function MoveNext()
{
  if( ! CustomerTable.moveNext() ) {
    CustomerTable.moveLast();
  }
}
```

2. MovePrev 関数を *tutorial.js* の MoveNext 関数の直後に追加します。

```
function MovePrev()
{
  if( ! CustomerTable.movePrevious() ) {
    CustomerTable.moveFirst();
  }
}
```

3. 次のプロシージャを *main.html* に追加します。

```
function ClickNext()
{
  MoveNext();
  DisplayCurrentRow();
}

function ClickPrev()
{
  MovePrev();
  DisplayCurrentRow();
}
```

4. アプリケーションを同期し、ナビゲーション・ボタンをテストします。

   最初にフォームが表示されると、現在位置が最初のローの前にあるため、コントロールは空です。フォームが表示されたら、[Next] と [Prev] をクリックして、テーブルのローの間を移動します。

レッスン 7：アプリケーションへの更新と削除の追加

このレッスンでは、ローを更新、削除するコードを説明します。

♦ アプリケーションへの更新関数と削除関数の追加

1. Update 関数を *tutorial.js* に追加します。

```
function Update()
{
  var id;
  try {
    CustomerTable.updateBegin();

    id = CustomerTable.schema.getColumnID("fname");
    CustomerTable.setString(id, Cust_FName);
    id = CustomerTable.schema.getColumnID("lname");
    CustomerTable.setString(id, Cust_LName);
    id = CustomerTable.schema.getColumnID("city");
    if( Cust_City.length > 0 ) {
      CustomerTable.setString(id, Cust_City);
    } else {
      CustomerTable.setNull(id);
    }
    id = CustomerTable.schema.getColumnID("phone");
    if( Cust_Phone.length > 0 ) {
      CustomerTable.setString(id, Cust_Phone);
    } else {
      CustomerTable.setNull(id);
    }
    CustomerTable.update();
  } catch( ex ) {
    alert( "Update error: " + ex.getMessage() );
  }
}
```

2. Delete 関数を *tutorial.js* に追加します。

```
function Delete()
{
  if( CustomerTable.getRowCount() == 0 ) {
    return;
  }
  CustomerTable.deleteRow();
  CustomerTable.moveRelative(0);
}
```

3. 次の関数を *main.html* に追加します。

```
function ClickUpdate()
{
  FetchForm();
  Update();
  DisplayCurrentRow();
}

function ClickDelete()
{
  Delete();
  DisplayCurrentRow();
}
```

4. M-Business Client を同期し、アプリケーションを実行します。

レッスン 8：アプリケーションへの同期の追加

次の手順は、同期を実装します。

♦ アプリケーションへの同期関数の追加

1. Synchronize 関数を *tutorial.js* に追加します。

   同期パラメータは、SyncParms オブジェクトに格納されます。たとえば SyncParms.userName プロパティは、Mobile Link が検索するユーザ名を指定します。SyncParms.sendColumnNames プロパティは、カラム名が Mobile Link に送信されることを指定し、アップロード・スクリプトとダウンロード・スクリプトを生成できるようにします。

   この関数は、TCP/IP 同期ストリームと、デフォルトのネットワーク通信オプション (ストリーム・パラメータ) を使用します。これらのデフォルトのオプションは、Mobile Link サーバを実行しているコンピュータに ActiveSync を使用して接続した Windows CE クライアントから、または Mobile Link と同じコンピュータで動作している 32 ビット Windows デスクトップ・クライアントから同期を実行していることを想定しています。これ以外の場合は、同期ストリーム・タイプを変更し、ネットワーク通信オプションを適切な値に設定してください。次の項を参照してください。

   - 「setStream メソッド」 135 ページ
   - 「setStreamParms メソッド」 135 ページ


```
function Synchronize()
{
  var SyncParms = Connection.syncParms;

  SyncParms.setUserName("user-name");
  SyncParms.setStream(SyncParms.STREAM_TYPE_TCPIP);
  SyncParms.setVersion("ul_default");
  SyncParms.setSendColumnNames(true);

  try {
    Connection.synchronize();
  } catch( ex ) {
    alert( "Sync error: " + ex.getMessage() );
  }
}
```

2. 次の関数を *main.html* に追加します。

```
function ClickSync()
{
  window.showBusy = true;
  Synchronize();
  window.showBusy = false;
  DisplayCurrentRow();
}
```

3. M-Business Client を同期します。

   この同期では、アプリケーションの最新バージョンをダウンロードします。データベースは同期しません。

このチュートリアルの最後の手順では、Ultra Light データベースを同期します。SQL Anywhere サンプル・データベースの Customer テーブルは、作成した Ultra Light データベースの **Customer** テーブルとカラムが一致しています。次の手順では、データベースを SQL Anywhere サンプル・データベースと同期します。

♦ **アプリケーションを同期するには、次の手順に従います。**

1. コマンド・プロンプトから、次のコマンドを実行して、Mobile Link サーバを起動します。

   ```
   mlsrv10 -c "dsn=SQL Anywhere 10 Demo" -v+ -zu+
   ```

   -zu+ オプションを指定すると、ユーザ・スクリプトの自動追加が行われます。このオプションの詳細については、「Mobile Link サーバのオプション」『Mobile Link - サーバ管理』を参照してください。

2. M-Business Client で [削除] を繰り返しクリックして、テーブルのすべてのローを削除します。

   テーブルにローがあると、SQL Anywhere サンプル・データベースの Customer テーブルに、ローがアップロードされます。

3. アプリケーションを同期します。

   [同期] をクリックします。Mobile Link サーバのウィンドウでは、同期の進行状況が表示されます。

4. 同期が完了したら、[次へ] と [前へ] をクリックしてテーブルのローの間を移動します。

これでチュートリアルが完了しました。

第 4 章

Ultra Light for M-Business Anywhere API リファレンス

目次

Ultra Light for M-Business Anywhere のデータ型

JavaScript には、数値データ型と DATE データ型がそれぞれ 1 つだけあります。

この API リファレンスのプロトタイプでは、メソッドやプロパティの説明にその他のデータ型が含まれています。これらの型は、M-Business Anywhere の内部データ型です。UInt32 (32 ビット符号なし整数) などの個別の数値データ型に対しては、渡される可能性のあるデータのサイズと精度について例を記載しています。日付と時間に関連する個別のデータ型 (DATA、TIME、TIMESTAMP) については、渡されるデータから必要な情報を抽出するコードを必要に応じて記述できるような説明を記載しています。

AuthStatusCode クラス

Mobile Link ユーザ認証の実行中にレポートされる可能性のあるステータス・コードを列挙します。

このオブジェクトを DatabaseManager から取得するには、次のように記述します。

```
var authStatus = dbMgr.AuthStatusCode;
```

プロパティ

AuthStatusCode のプロパティは次の定数です。

定数	値	説明
UNKNOWN	0	認証ステータスが不明です。接続がまだ同期を実行していない可能性があります。
VALID	1	ユーザ ID とパスワードは、同期時には有効でした。
VALID_BUT_EXPIRES_SOON	2	ユーザ ID とパスワードは、同期時には有効でしたが、まもなく有効期限が切れます。
EXPIRED	3	ユーザ ID またはパスワードの有効期限が切れているため、認証に失敗しました。
INVALID	4	ユーザ ID またはパスワードが正しくないため、認証に失敗しました。
IN_USE	5	ユーザ ID がすでに使用されているため、認証に失敗しました。

toString メソッド

認証ステータス・コード定数の文字列名を生成します。

構文

String **toString()**;

戻り値

コードの名前、または認識されたコードでない場合は **unknown**。

Connection クラス

Ultra Light データベースへの接続を示します。

次のいずれかのメソッドを使用して、接続がインスタンス化されます。

- DatabaseManager.openConnection
- DatabaseManager.createDatabase

接続は、他の操作の実行前に開きます。また、その接続での操作がすべて終了したら、接続を閉じてからアプリケーションを終了します。

接続で開いたテーブルをすべて閉じてから、その接続を閉じます。

Ultra Light データベース操作の失敗が原因で JavaScript エラーがスローされた場合、SQL エラー・コードが Connection オブジェクトの sqlCode フィールドに設定されます。

プロパティ

プロトタイプ	説明
Boolean autoCommit	各文 (insert、update、delete) の後でコミットを行うかどうかを指定する。 **autoCommit** が false の場合、コミットまたはロールバックが実行されるのは、ユーザが **commit()** メソッドまたは **rollback()** メソッドを呼び出したときだけです。 デフォルトでは、各文の処理が成功した後でデータベースのコミットが実行されます。コミットに失敗する場合は、追加の SQL 文を実行してからコミットを再度実行するか、ロールバック文を実行できます。
String openParms (読み込み専用)	name=value の組み合わせをセミコロンで区切ったリストで、接続パラメータの文字列を取得する。 「Ultra Light の接続文字列パラメータのリファレンス」『Ultra Light - データベース管理とリファレンス』を参照してください。
DatabaseSchema databaseSchema (読み込み専用)	データベース・スキーマを取得する。このプロパティが有効なのは、接続が開かれている間だけです。
Boolean skipMBASync (読み込み／書き込み)	ワンタッチ同期中にデータベースを同期するか (false)、スキップするか (true) を指定する。 デフォルトは false です。 「ワンタッチ同期」 32 ページを参照してください。

プロトタイプ	説明
Int32 sqlCode (読み込み専用)	この接続で行った最後の操作の SQL コードを取得する。 SQL コードは、SQL Anywhere の標準のコードであり、この接続の以後の Ultra Light データベース操作によってリセットされます。
SyncParms syncParms (読み込み専用)	この接続の同期設定を取得する。 「Ultra Light の同期パラメータ」『Mobile Link - クライアント管理』を参照してください。
SyncResult syncResult (読み込み専用)	この接続の最後の同期の結果を取得する。 「Ultra Light の同期パラメータ」『Mobile Link - クライアント管理』を参照してください。
INVALID_DATABASE_ID (読み込み専用)	無効なデータベースを示す定数。

ChangeEncryptionKey メソッド

データベースの暗号化キーを、指定した新しいキーに変更します。

構文

changeEncryptionKey(String *newKey*)

パラメータ

- **newkey**　データベースの新しい暗号化キー。

備考

暗号化キーが失われた場合は、データベースを開くことができません。

close メソッド

この接続を閉じます。

構文

close()

備考

接続を閉じると、その接続をもう一度開くことはできません。接続をもう一度開くには、新しい接続オブジェクトを作成して開く必要があります。

閉じた接続に関連付けられたオブジェクト (テーブル、スキーマなど) を使用するとエラーになります。

JavaScript では、閉じた接続オブジェクトは接続が閉じた後で自動的に NULL に設定されません。接続を閉じた後で、明示的に接続オブジェクトを NULL に設定することをおすすめします。

commit メソッド

未処理の変更をデータベースにコミットします。

構文

commit()

CountUploadRow メソッド

次回の同期でアップロードするロー数を返します。

構文

UInt32 **countUploadRow(** UInt32 *mask*, UInt32 *threshold***)**

パラメータ

- **mask**　チェック対象のパブリケーションのセット。

  PublicationSchema クラスを参照してください。

- **threshold**　カウントするローの最大数を決定する値。呼び出しにかかる時間を制限します。値 0 は制限が最大であることを示します。値 1 は、同期の必要なローがあるかどうかを判別する場合に使用します。**threshold** は、**[0,0x0fffffff]** の範囲内である必要があります。

getDatabaseID メソッド

setDatabaseID() で設定された現在のデータベース ID 値を取得します。

構文

UInt32 **getDatabaseID( )**

備考

値が設定されていない場合は、定数 Connection.INVALID_DATABASE_ID が返されます。

getGlobalAutoIncrementUsage メソッド

利用可能なグローバル・オートインクリメントの値の使用済み比率 (%) を返します。

構文

UInt16 **getGlobalAutoIncrementUsage( )**

備考

比率が 100% になったら、**setDatabaseID** を使用して、使用中のアプリケーションに新しいグローバル・データベース ID 値を設定してください。

getLastDownloadTime メソッド

最後のダウンロードのタイムスタンプを返します。

構文

Date **getLastDownloadTime(** UInt32 *mask* **)**

パラメータ

- **mask**　チェック対象のパブリケーションのセット。

備考

パラメータ mask は、単一のパブリケーションを参照するか、データベース全体の最後のダウンロード時刻の特殊定数 PublicationSchema.SYNC_ALL_DB である必要があります。

参照

- 「 PublicationSchema クラス」 94 ページ

getLastIdentity メソッド

直前に使用した identity の値を返します。

構文

UInt64 **getLastIdentity()**

備考

この関数は、次の SQL 文と同義です。

```
SELECT @@identity
```

この関数は、グローバル・オートインクリメント・カラムで使うと特に便利です。戻り値は、符号なし 64 ビット整数であるデータベース・データ型 UNSIGNED BIGINT です。この文では最後に割り当てられたデフォルト値がわかるだけなので、間違った結果を取らないために INSERT 文を実行した直後にこの値を取り出してください。

ときには、1 つの INSERT 文にグローバル・オートインクリメント型のカラムが複数含まれていることがあります。この場合、戻り値は生成されたデフォルト値のいずれか 1 つですが、そのうちのどの値であるかを判別する信頼できる方法はありません。このため、このような状況を回避するようなデータベースの設計と INSERT 文の記述を行ってください。

getNewUUID メソッド

新しい UUID 値を返します。

構文

UUID **getNewUUID()**

getTable メソッド

データベース内の要求されたテーブルへの参照を作成して返します。

構文

Table **getTable(**String *name*, String *persistName* **)**

パラメータ

- **name**　フェッチするテーブルの名前。
- **persistName**　ページ間 JavaScript オブジェクト持続性の名前。持続性が必要ない場合 (たとえばアプリケーションに単一の HTML ページしかない場合) は null を設定します。

grantConnectTo メソッド

指定されたパスワードを持つユーザ ID に、Ultra Light データベースへのアクセスを許可します。

構文

grantConnectTo(String *uid*, String *pwd***)**

パラメータ

- **uid**　アクセスを許可するユーザ ID。最大長は 16 文字です。
- **pwd**　ユーザ ID のパスワード。

備考

既存のユーザ ID が指定されていれば、この関数を使用してそのユーザのパスワードを更新します。Ultra Light では、最大で 4 人のユーザがサポートされます。

isOpen メソッド

接続が開いている場合は true、閉じている場合は false を返します。

構文

Boolean **isOpen()**;

prepareStatement メソッド

IN パラメータあり、または IN パラメータなしで、SQL 文を事前にコンパイルして PreparedStatement オブジェクトに格納します。

構文

PreparedStatement **prepareStatement(**String *sql*, String *persistName***)**

パラメータ

- **sql**　IN パラメータのプレースホルダ '?' が 1 つまたは複数含まれている可能性のある SQL 文。
- **persistName**　ページ間 JavaScript オブジェクト持続性の名前。持続性が必要ない場合 (たとえばアプリケーションに単一の HTML ページしかない場合) は null を設定します。

備考

このオブジェクトを使用すると、SQL 文を効率的に何回も実行できます。

resetLastDownloadTime メソッド

最後のダウンロードの時刻をリセットします。

構文

resetLastDownloadTime(UInt32 *mask***)**

パラメータ

- **mask**　リセット対象のパブリケーションのセット。

revokeConnectFrom メソッド

指定したユーザ ID に対して、Ultra Light データベースへのアクセス権を取り消します。

構文

revokeConnectFrom(String *uid* **)**

パラメータ

- **uid**　データベース・アクセスから除外されるユーザ ID。最大長は 16 文字です。

rollback メソッド

未処理の変更をデータベースにロールバックします。

構文

rollback( )

rollbackPartialDownload メソッド

失敗した同期から変更をロールバックします。

構文

rollbackPartialDownload( )

備考

同期のダウンロード時に通信エラーが発生した場合、Ultra Light では、ダウンロードした変更を適用し、同期が中断した時点から同期を再開することができます。ダウンロードした変更が不要な場合 (ダウンロードが中断した時点での再開を望まない場合)、RollbackPartialDownload を使用することで、失敗したダウンロード・トランザクションをロールバックします。

saveSyncParms メソッド

HotSync または ワンタッチ同期で使用される同期パラメータを保存します。

構文

saveSyncParms( )

備考

saveSyncParms メソッドと Connection.SyncParms プロパティを混同しないでください。SyncParms プロパティは、この接続の同期パラメータを定義するために使用されます。setSyncParms メソッドは、HotSync で使用できるようにそれらのパラメータを保存するだけです。

参照

- 「ワンタッチ同期」 32 ページ

setDatabaseID メソッド

グローバル・オートインクリメント・カラムに使用するデータベース ID の値を設定します。

構文

setDatabaseID(UInt32 *value* **)**

パラメータ

- **value**　データベース ID 値。**value** は、**[0,0x0fffffff]** の範囲内である必要があります。

startSynchronizationDelete メソッド

同期用に、この接続によって行われる以後のすべての削除にマークを付けます。

構文

startSynchronizationDelete( )

備考

この関数が呼び出されると、すべての削除操作がもう一度同期されます。

stopSynchronizationDelete メソッド

この関数が呼び出されると、削除操作が同期されなくなります。

構文

stopSynchronizationDelete( )

備考

領域を節約するために、Ultra Light データベースから古い情報を削除して、統合データベースにはそれを残しておく場合に使用すると便利です。

synchronize メソッド

現在の SyncParms オブジェクトを使用してデータベースを同期します。

構文

synchronize( )

備考

結果の詳細なステータスは、この接続の SyncResult オブジェクトでレポートされます。この接続の Connection.SyncParms オブジェクトで定義される同期プロパティを使用して、同期が実行されます。

synchronizeWithParm メソッド

指定された SyncParms オブジェクトを使用してデータベースを同期します。

構文

synchronizeWithParm(SyncParms *parms***)**

パラメータ

- **parms**　この同期で使用される SyncParms オブジェクト。

備考

このメソッドでは、接続間で同期パラメータを共有できます。

結果の詳細なステータスは、この接続の SyncResult オブジェクトでレポートされます。

ConnectionParms クラス

Ultra Light データベースへの接続を開くためのパラメータを指定します。

データベースは、1 人の認証済みユーザ DBA で作成されます。このユーザの最初のパスワードは sql です。デフォルトでは、ユーザ ID DBA とパスワード sql を使用して、接続が開かれます。デフォルトのユーザを無効にするには、Connection.revokeConnectFrom を使用します。ユーザを追加したりユーザのパスワードを変更するには、Connection.grantConnectTo を使用します。

現在のところ、一度に開くことができる接続は 1 つのみです。一度にアクティブにできるデータベースは 1 つのみです。他の接続が開いているときに別のデータベースへの接続を開こうとすると、エラーが発生します。

プロパティ

このクラスのプロパティは、次のとおりです。

プロトタイプ	説明
String additionalParms (読み込み／書き込み)	セミコロンで区切られた **name**=*value* の組み合わせとして指定される追加のパラメータ。
String cacheSize (読み込み／書き込み)	キャッシュのサイズ。CacheSize の値はバイト単位で指定します。キロバイトの単位を示すにはサフィックス k または K を使用し、メガバイトの単位を示すにはサフィックス M または m を使用します。デフォルトのキャッシュ・サイズは 16 ページです。デフォルトのページ・サイズは 4 KB なので、デフォルトのキャッシュ・サイズは 64 KB です。 「Ultra Light CACHE_SIZE 接続パラメータ」『Ultra Light - データベース管理とリファレンス』を参照してください。
String connectionName (読み込み／書き込み)	接続の名前。接続名は、複数の Web ページ間で 1 つの接続を共有するために使用されます。 「Ultra Light CON 接続パラメータ」『Ultra Light - データベース管理とリファレンス』と「ページ間の接続とアプリケーション状態の管理」 12 ページを参照してください。
String creatorIdOnPalm	Palm デバイス上の Ultra Light データベースの作成者 ID。
String databaseOnCE (読み込み／書き込み)	Windows CE 上のデータベースのファイル名。 「Ultra Light CE_FILE 接続パラメータ」『Ultra Light - データベース管理とリファレンス』を参照してください。
String databaseOnDesktop (読み込み／書き込み)	Windows に配備されるデータベースのファイル名。 「Ultra Light DBF 接続パラメータ」『Ultra Light - データベース管理とリファレンス』を参照してください。

プロトタイプ	説明
String databaseOnPalm (読み込み／書き込み)	Palm 上の Ultra Light データベースのファイル名。 「Ultra Light PALM_FILE 接続パラメータ」『Ultra Light - データベース管理とリファレンス』を参照してください。
String databaseOnSymbian (読み込み／書き込み)	Symbian 上の Ultra Light データベースのファイル名。 「Ultra Light SYMBIAN_FILE 接続パラメータ」『Ultra Light - データベース管理とリファレンス』を参照してください。
String encryptionKey (読み込み／書き込み)	データベースを暗号化するためのキー。OpenConnection は、データベース作成中に指定されたキーと同じキーを使用する必要があります。キーは以下の条件を満たしていることが推奨されます。 1. 任意の長い文字列を選択します。 2. キーを見破られる可能性を減らすため、多様な数字、文字、特殊文字を使用した文字列を選択します。 「Ultra Light DBKEY 接続パラメータ」『Ultra Light - データベース管理とリファレンス』を参照してください。
String password (読み込み／書き込み)	認証ユーザのパスワード。データベースは最初、1 つの認証されたユーザ (DBA) とパスワード *sql* を使用して、作成されます。データベースで大文字と小文字が区別されない場合は、パスワードの大文字と小文字は区別されません。データベースで大文字と小文字が区別される場合は、パスワードの大文字と小文字も区別されます。デフォルト値は *sql* です。 「Ultra Light PWD 接続パラメータ」『Ultra Light - データベース管理とリファレンス』を参照してください。
String userID (読み込み／書き込み)	データベースで認証されたユーザ。データベースは最初、1 つの認証されたユーザ DBA を使用して、作成されます。UserID では大文字と小文字は区別されません。デフォルト値は *DBA* です。 「Ultra Light UID 接続パラメータ」『Ultra Light - データベース管理とリファレンス』を参照してください。

toString メソッド

認証ステータス・コード定数の文字列名を生成します。

構文

String **toString()**;

戻り値

コードの名前、または認識されたコードでない場合は **unknown**。

CreationParms クラス

Ultra Light データベースの作成時に指定できるパラメータを定義します。

一部の Ultra Light データベース・オプションは、データベースの作成時に設定する必要があります。createDatabase メソッドを使用してデータベースを作成するときは、以下のパラメータを指定できます。「 createDatabase メソッド」 73 ページを参照してください。

プロパティ

このクラスのプロパティは、次のとおりです。対応する説明の詳細については、「Ultra Light で使用する作成時のデータベース・プロパティの選択」『Ultra Light - データベース管理とリファレンス』を参照してください。

プロトタイプ	説明
Boolean caseSensitive	Ultra Light データベースの文字列を比較するときに大文字と小文字を区別するかどうかを設定する。
UInt32 checksumLevel	データベースのチェックサム検証のレベルを設定する。 デフォルトは 0 です。
String dateFormat	日付がデータベースから取り出されるときのデフォルトの文字列フォーマットを設定する。
String dateOrder	年、月、日の日付の順序を解釈する方法を制御する。
UInt32 maxHashSize	Ultra Light のインデックスのハッシュに使用する最大バイト数を設定する。 デフォルトは 0 です。
UInt32 nearestCentury	文字列から日付への変換で、2 桁の年の解釈を制御する。
Boolean obfuscate	データベース内のデータを難読化するかどうかを制御する。難読化は単純暗号化です。
UInt32 pageSize	データベース・ページ・サイズを定義する。有効な値は、1024、2048、4096、8192、16384 です。 デフォルトは 4096 です。
UInt32 precision	10 進法計算での結果の最大桁数を指定する。
UInt32 scale	計算結果が最大 precision にトランケートされる場合の、小数点以下の最小桁数を指定する。
String timeFormat	データベースから取り出した時刻のフォーマットを設定する。

プロトタイプ	説明
String timestampFormat	Ultra Light でのタイムスタンプのフォーマットを指定する。
String timestampIncrement	Ultra Light でのタイムスタンプのトランケート方法を指定する。
Boolean utf8Encoding	ユニコード用の 8 ビット・マルチバイト・エンコードである UTF-8 フォーマットでデータをエンコードする。

DatabaseManager クラス

Ultra Light データベースへの接続を管理します。

接続は、他の操作の実行前に開きます。また、その接続での操作がすべて終了したら、接続を閉じてからアプリケーションを終了します。接続で開いたテーブルをすべて閉じてから、その接続を閉じます。

プロパティ

このクラスのプロパティは、次のとおりです。

プロパティ	説明
AuthStatusCode **AuthStatusCode** (読み込み専用)	最後の同期に関連付けられた AuthStatusCode オブジェクトを取得する。
String **directory** (読み込み専用)	M-Business Anywhere が実行しているディレクトリ。 Palm OS の場合は NULL です。
UInt32 **runtimeType** (読み込み専用)	ランタイム・タイプ。Ultra Light ランタイム (スタンドアロン) ライブラリまたは Ultra Light データベース・エンジンのいずれかです。値は一覧で、次のいずれかです。 ◆ DatabaseManager.UL_STANDALONE ◆ DatabaseManager.UL_ENGINE_CLIENT
Int32 **sqlCode** (読み込み専用)	最後の操作に関連付けられた SQL コード値を取得する。
SQLError **SQLError** (読み込み専用)	SQLError オブジェクトを取得する。
SQLType **SQLType** (読み込み専用)	SQLType オブジェクトを取得する。
PODSUInt32 **UL_STANDALONE** (読み込み専用)	ランタイム・タイプが Ultra Light ランタイム・ライブラリであることを示す定数。
PODSUInt32 **UL_ENGINE_CLIENT** (読み込み専用)	ランタイム・タイプが Ultra Light データベース・エンジンであることを示す定数。

createDatabase メソッド

データベースを作成し、**access_parms** によって指定されたデータベースへの接続を開きます。

構文

Connection **createDatabase(**
String *access_parms* ,
PODSArray **coll_bytes*,
String *create_parms*
)

パラメータ

- **access_parms**　データベースに接続するためのパラメータ。access_parms は、接続パラメータ (データベースのファイル名やロケーションなど) を指定したり、接続を開いたりするために使用されます。

  「Ultra Light データベースへの接続」『Ultra Light - データベース管理とリファレンス』を参照してください。

- **coll_bytes**　作成されるデータベースで使用するデータベース照合を定義するバイト配列。Ultra Light には多くのソース・ファイルが付属しています。これらは *install-dir¥src¥ulcollations¥* にある JavaScript ソース・ファイル (.js) で、ファイル名の形式は **Collation_XXXXX.js** です。XXXXX は照合名を表します。例：*coll_1250LATIN2.js*

  .js ファイルは、メインの html ファイルでデータベース論理の前にインクルードする必要があります。バイト配列変数は .js ファイルで定義されます。

- **create_parms**　データベースを作成するためのパラメータ。パラメータ・キーワードは大文字と小文字を区別しませんが、ほとんどの値は大文字と小文字を区別します。create_parms は、データベースの作成時のみ指定できる特定のパラメータを指定するために使用されます。

  「Ultra Light で使用する作成時のデータベース・プロパティの選択」『Ultra Light - データベース管理とリファレンス』を参照してください。

戻り値

戻り値なし。

備考

すでにデータベースが存在する場合は、SQLE_DATABASE_NOT_CREATED 例外がスローされます。

一度にアクティブにできるデータベースは 1 つのみです。データベースへの接続が開いているときに、別のデータベースへの接続を開こうとすると、エラーが発生します。

dropDatabase メソッド

指定したデータベースを削除します。

構文

dropDatabase(String *parms***)**

パラメータ

- **parms**　データベースを識別するためのパラメータ。

備考

parms は、keyword=value の組み合わせをセミコロンで区切ったリスト (**"param1=value1;param2=value2"**) で指定します。パラメータ・キーワードは大文字と小文字を区別しませんが、ほとんどの値は大文字と小文字を区別します。

接続が開かれているデータベースを削除することはできません。

getDatabaseOptions メソッド

構文

Connection **openConnection(** String *parms***)**

openConnection メソッド

parms によって指定されたデータベースへの接続を開きます。

構文

Connection **openConnection(** String *parms***)**

パラメータ

- **parms**　接続を開くためのパラメータ (keyword=value の組み合わせのセット) を保持する文字列。パラメータ・キーワードは大文字と小文字を区別しませんが、ほとんどの値は大文字と小文字を区別します。

戻り値

開かれた接続。

備考

データベースが存在しない場合は、エラーがスローされます。エラーをキャッチするコード内で Connection.sqlCode をチェックすると、エラーの原因を特定できます。

一度にアクティブにできるデータベースは 1 つのみです。他の接続が開いているときに別のデータベースを作成しようとすると、エラーが発生します。

reOpenConnection メソッド

開かれた Connection オブジェクトを返します。

構文

Connection **reOpenConnection(** String *connectionName* **)**

パラメータ

- **connectionName**　ConnectionParms.connectionName プロパティで指定される、もう一度開く接続の名前。

戻り値

このメソッドを使用して、複数の Web ページ間の接続を管理します。

DatabaseSchema クラス

Ultra Light データベースのスキーマを表します。**DatabaseSchema** オブジェクトは、接続にアタッチされ、接続が開いている間のみ有効です。

定数

定数	説明
SYNC_ALL_DB	データベース内のすべてのテーブルを同期する。
SYNC_ALL_PUBS	データベース内のすべてのパブリケーションを同期する。

このクラスのメンバは、次のとおりです。

getCollationName メソッド

このデータベースで使用される文字セットとソート順を示す文字列を返します。

構文

String **getCollationName()**

getDatabaseProperty メソッド

指定したデータベースのプロパティの値を返します。

構文

String **getDatabaseProperty(**String *name***)**

パラメータ

- **name**　データベースのプロパティの名前。

備考

識別されるプロパティは次のとおりです。

- **"date_format"**　データベースによる文字列変換に使用される日付フォーマット。
- **"date_order"**　データベースによる文字列変換に使用される日付順。
- **"nearest_century"**　データベースによる文字列変換に使用される最も近い世紀。
- **"precision"**　データベースによる文字列変換に使用される浮動小数点の精度。
- **"scale"**　データベースによる文字列変換中に、計算結果が最大 precision にトランケートされるときの小数点以下の最小桁数。

- **"time_format"**　データベースによる文字列変換に使用される時刻フォーマット。
- **"timestamp_format"**　データベースによる文字列変換に使用されるタイムスタンプのフォーマット。
- **"timestamp_increment"**　2 つのユニークなタイムスタンプ間のナノ秒 (1,000,000 分の 1 秒) 単位の最小差。

getDateFormat メソッド

文字列変換に使用される日付フォーマットを返します。

構文

String **getDateFormat()**

getDateOrder メソッド

文字列変換に使用される日付順を返します。

構文

String **getDateOrder()**

getNearestCentury メソッド

文字列変換に使用される最も近い世紀を返します。

構文

String **getNearestCentury()**

getPrecision メソッド

文字列変換に使用される浮動小数点の精度を返します。

構文

String **getPrecision( )**

getPublicationCount メソッド

データベース内のパブリケーションの数を返します。

構文

UInt16 **getPublicationCount( )**

備考

パブリケーション ID の範囲は、1 ー **getPublicationCount()** です。パブリケーション ID は、パブリケーション・マスクではありません。

注意：パブリケーションの ID、マスク、カウントは、スキーマのアップグレード中に変更されることがあります。パブリケーションを正しく識別するには、名前でアクセスするか、キャッシュされている ID、マスク、カウントをスキーマのアップグレード後にリフレッシュします。

getPublicationName メソッド

指定したパブリケーション ID で識別されたパブリケーションの名前を返します。

構文

String **getPublicationName(** UInt16 *publID* **)**

パラメータ

- **publID**　パブリケーションの ID。**publID** は、**[1,getPublicationCount()]** の範囲内である必要があります。

備考

パブリケーション ID は、パブリケーション・マスクではありません。

注意：パブリケーションの ID、マスク、カウントは、スキーマのアップグレード中に変更されることがあります。パブリケーションを正しく識別するには、名前でアクセスするか、キャッシュされている ID、マスク、カウントをスキーマのアップグレード後にリフレッシュします。

getPublicationSchema メソッド

指定したパブリケーションに対応するパブリケーション・スキーマを返します。

構文

PublicationSchema **getPublicationSchema(** String *name* **)**

パラメータ

- **name**　パブリケーションの名前。

getSignature メソッド

このデータベースのシグニチャを返します。

構文

String **getSignature( )**

getTableCount メソッド

データベース内のテーブルの数を返します。

構文

UInt16 **getTableCount( )**

戻り値

テーブル数。接続が開いていない場合は 0。

備考

テーブル ID の範囲は、1 ー **getTableCount()** です。

getTableCountInPublications メソッド

指定したパブリケーション・マスクに含まれるテーブルの数を返します。

構文

UInt16 **getTableCountInPublications(** UInt32 *mask* **)**

パラメータ

- **mask**　チェック対象のパブリケーションのセット。

備考

カウントには、名前が _nosync で終わるテーブルは含まれません。

getTableName メソッド

指定したテーブル ID で識別されたテーブルの名前を返します。

構文

String **getTableName(** UInt16 *tableID***)**

パラメータ

- **tableID**　テーブルの ID。**tableID** は、**[1,getTableCount()]** の範囲内である必要があります。

備考

注意：テーブル ID は、スキーマのアップグレード中に変更されることがあります。テーブルを正しく識別するには、名前でアクセスするか、キャッシュされている ID をスキーマのアップグレード後にリフレッシュします。

getTimeFormat メソッド

文字列変換に使用される時刻フォーマットを返します。

構文

String **getTimeFormat()**

getTimestampFormat メソッド

文字列変換に使用されるタイムスタンプのフォーマットを返します。

構文

String **getTimestampFormat()**

isCaseSensitive メソッド

データベースで大文字と小文字が区別される場合は true、区別されない場合は false を返します。

構文

Boolean **isCaseSensitive()**

isOpen メソッド

データベース・スキーマが開いている場合は true、閉じている場合は false を返します。

構文

Boolean **isOpen()**

IndexSchema クラス

Ultra Light テーブルのインデックスのスキーマを表します。

このオブジェクトを直接インスタンス化することはできません。インデックス・スキーマは、TableSchema.getPrimaryKey、TableSchema.getIndex、TableSchema.getOptimalIndex の各メソッドを使用して作成されます。

getColumnCount メソッド

インデックス内のカラム数を返します。

構文

UInt16 **getColumnCount( )**

備考

インデックス内のカラム ID の範囲は、1 ー getColumnCount() です。

getColumnName メソッド

このインデックス内の *colIDInIndex* カラムの名前を返します。

構文

String **getColumnName(**UInt16 *colIDInIndex* **)**

パラメータ

- **colIDInIndex**　このカラムのインデックス内の ID。colIDInIndex は、[1, getColumnCount()] の範囲内である必要があります。

getName メソッド

このインデックスの名前を返します。

構文

String **getName()**

getReferencedIndexName メソッド

このインデックスが外部キーである場合、参照されるプライマリ・インデックスの名前を返します。

構文

String **getReferencedIndexName()**

getReferencedTableName メソッド

インデックスが外部キーである場合、参照されるプライマリ・テーブルの名前を返します。

構文

String **getReferencedTableName()**

isColumnDescending メソッド

カラムが降順で使用される場合は true、昇順で使用される場合は false を返します。

構文

Boolean **isColumnDescending(**String *name***)**

パラメータ

- **name**　カラムの名前。

isForeignKey メソッド

インデックスが外部キーである場合は true、外部キーでない場合は false を返します。

構文

Boolean **isForeignKey()**

備考

外部キー内のカラムは、別のテーブルの null 以外のユニーク・インデックスを参照することができます。

isForeignKeyCheckOnCommit メソッド

参照整合性がコミット時にチェックされる場合は true、挿入時および更新時にチェックされる場合は false を返します。

構文

Boolean **isForeignKeyCheckOnCommit()**

isForeignKeyNullable メソッド

この外部キーが null 入力可能であれば true、null 入力不可であれば false を返します。

構文

Boolean **isForeignKeyNullable()**

isPrimaryKey メソッド

インデックスがプライマリ・キーである場合は true、プライマリ・キーでない場合は false を返します。

構文

Boolean **isPrimaryKey()**

備考

プライマリ・キー内のカラムでは null は許可されません。

isUniqueIndex メソッド

インデックスがユニークである場合は true、ユニークでない場合は false を返します。

構文

Boolean **isUniqueIndex()**

備考

ユニークなインデックス内のカラムは null であることがあります。

isUniqueKey メソッド

インデックスがユニーク・キーである場合は true、ユニーク・キーでない場合は false を返します。

構文

Boolean **isUniqueKey()**

備考

ユニーク・キー内のカラムでは null は許可されません。

PreparedStatement クラス

IN パラメータあり、または IN パラメータなしで事前にコンパイルされた SQL 文を表します。実行時に Connection.prepareStatement を使用して作成されます。

このオブジェクトを使用すると、この文を効率的に何回も実行できます。

準備文が閉じられると、それに関連する ResultSet オブジェクトと ResultSetSchema オブジェクトもすべて閉じられます。リソース管理の理由により、準備文を使用し終わったら、その準備文を明示的に閉じることをおすすめします。

AppendBytesParameter メソッド

指定したバイト配列の指定したサブセットを、指定した SQLType.LONGBINARY カラムの新しい値に追加します。

構文

AppendBytesParameter(
 UInt16 *parameterID*,
 Array *value*,
 UInt32 *srcOffset*,
 UInt32 *count*
)

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの現在の新しい値に追加する値。
- **srcOffset**　ソース配列の開始位置。
- **count**　コピーされるバイト数。

備考

配列 **value** の srcOffset (0 から始まります) から srcOffset+count-1 までの位置のバイトが、指定したパラメータの値に追加されます。挿入時には、**insertBegin** は新しい値をパラメータのデフォルト値に初期化します。

次のいずれかに該当する場合、コード SQLError.SQLE_INVALID_PARAMETER とともにエラーがスローされ、追加先は修正されません。

- **value** 引数が null である
- **srcOffset** 引数が負の値である
- **count** 引数が負の値である
- **srcOffset+count** がソース配列の長さ **value.length** よりも大きい

AppendStringChunkParameter メソッド

文字列を指定した SQLType.LONGVARCHAR の新しい値に追加します。

構文

AppendStringChunkParameter(
  UInt16 *parameterID*,
  String *value*,
)

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの現在の新しい値に追加する値。

例

次の文は、文字列 **XYZ** のインスタンス 100 個を最初のパラメータに追加します。

```
for ( i = 0; i < 100; i++ ){
  stmt.AppendStringChunkParameter( 1, "XYZ" );
}
```

close メソッド

準備文を閉じます。

構文

close( )

備考

準備文が閉じられると、それに関連する ResultSet オブジェクトと ResultSetSchema オブジェクトもすべて閉じられます。

preparedStatement オブジェクトを閉じたら、ただちに null に設定することをおすすめします。

executeQuery メソッド

SQL SELECT 文を実行し、結果セットを返します。

構文

ResultSet **executeQuery(** String *persistName* **)**

パラメータ

- **persistName**　ページ間 JavaScript オブジェクト持続性の名前。持続性が必要ない場合 (たとえばアプリケーションに単一の HTML ページしかない場合) は null を設定します。

戻り値

クエリの結果セット (ローのセット)。

executeStatement メソッド

SQL INSERT 文、DELETE 文、UPDATE 文のように、結果セットを返さない文を実行します。

構文

Int32 **executeStatement( )**

戻り値

文の影響を受けるローの数。

備考

Connection.autoCommit が true の場合は、文が 1 つまたは複数のローに影響するときだけ、文がコミットされます。

getPlan メソッド

クエリを実行するのに Ultra Light が使用するアクセス・プランを記述する文字列を返します。

構文

String **getPlan( )**

備考

このメソッドは、主に開発中の使用を目的とします。

参照

- 「Ultra Light のクエリ・アクセス・プラン」 『Ultra Light - データベース管理とリファレンス』

getResultSetSchema メソッド

このクエリ文の結果セットを記述するスキーマを返します。

構文

ResultSetSchema **getResultSetSchema()**

hasResultSet メソッド

この文が実行されたときに結果セットが生成される場合は true、生成されない場合は false を返します。

構文

Boolean **hasResultSet( )**

isOpen メソッド

準備文が開いている場合は true、閉じている場合は false を返します。

構文

Boolean **isOpen( )**

setBooleanParameter メソッド

指定したパラメータの値を、Boolean を使用して設定します。

構文

setBooleanParameter(UInt16 *parameterID*, Boolean *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を設定します。

```
stmt.setBooleanParameter(1, false);
```

setBytesParameter メソッド

指定したパラメータの値を、バイト配列を使用して設定します。

構文

setBytesParameter(UInt16 *parameterID*, Array *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

備考

SQLType.BINARY 型、SQLType.LONGBINARY 型のカラムにのみ適しています。

例

次の文は、最初のパラメータの値を設定します。

```
var blob = new Array( 3 );
blob[ 0 ] = 78;
blob[ 1 ] = 0;
blob[ 2 ] = 68;
stmt.setBytesParameter( 1, blob );
```

setDateParameter メソッド

指定した SQLType.DATE 型のパラメータの値を、日付を使用して設定します。

構文

setDateParameter(UInt16 *parameterID*, Date *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

備考

Date オブジェクトの年、月、日のフィールドだけが関係します。

例

次の文は、最初のパラメータの値を 2004/09/27 に設定します。

```
stmt.setDateParameter(
    1, new Date( 2004,9,27,0,0,0,0 )
);
```

setDoubleParameter メソッド

指定したパラメータの値を、**double** を使用して設定します。

構文

setDoubleParameter(UInt16 *parameterID*, Double *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を設定します。

```
stmt.setDoubleParameter( 1, Number.MAX_VALUE );
```

setFloatParameter メソッド

指定した SQLType.REAL パラメータの値を設定します。

構文

setFloatParameter(UInt16 *parameterID*, Float *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの浮動小数点値を設定します。

```
stmt.setFloatParameter( 1,
   (2 - Math.pow(2,-23)) * Math.pow(2,127)
);
```

setIntParameter メソッド

指定したパラメータの値を、UInt16 を使用して設定します。

構文

setUInt16Parameter(UInt16 *parameterID*, UInt16 *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を **2147483647** に設定します。

```
stmt.setIntParameter( 1, 2147483647 );
```

setLongParameter メソッド

指定したパラメータの値を設定します。

構文

setLongParameter(UInt16 *parameterID*, Int64 *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を **9223372036854770000** に設定します。

```
stmt.setLongParameter( 1, 9223372036854770000 );
```

setNullParameter メソッド

指定したパラメータの値を SQL NULL に設定します。

構文

setNullParameter(UInt16 *parameterID* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。

setShortParameter メソッド

指定したパラメータの値を設定します。

構文

setUInt16Parameter(UInt16 *parameterID*, UInt16 *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を **32767** に設定します。

```
stmt.setShortParameter( 1, 32767 );
```

setStringParameter メソッド

指定したパラメータの値を設定します。

構文

setStringParameter(UInt16 *parameterID*, String *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を **ABC** に設定します。

```
stmt.setStringParameter( 1, "ABC" );
```

setTimeParameter メソッド

指定した SQLType.TIME 型のパラメータの値を、日付を使用して設定します。

構文

setTimeParameter(UInt16 *parameterID*, Date *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

備考

Date オブジェクトの時、分、秒のフィールドだけが関係します。

例

次の文は、最初のパラメータの値を 18:02:13:0000 に設定します。

```
stmt.setTimeParameter(
   1, new Date( 1966,4,1,18,2,13,0 )
);
```

setTimestampParameter メソッド

指定したパラメータの値を、**Timestamp** を使用して設定します。

構文

setTimestampParameter(UInt16 *parameterID*, Date *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を 1966/04/01 18:02:13:0000 に設定します。

```
stmt.setTimestampParameter(
   1, new Date( 1966,4,1,18,2,13,0 )
);
```

setULongParameter メソッド

指定したパラメータの値を、符号なし値として扱われる Double を使用して設定します。

構文

setULongParameter(UInt16 *parameterID*, UInt64 *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。64 ビット符号なし整数値を表す Double を使用します。

備考

Unsigned64 クラスを参照してください。

例

次の文は、最初のパラメータの値を設定します。

```
stmt.setLongParameter( 1, 9223372036854770000 * 4096 );
```

setUUIDParameter メソッド

指定したパラメータの値を、**UUID** を使用して設定します。

構文

setUUIDParameter(UInt16 *parameterID*, UUID *value***)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

PublicationSchema クラス

Ultra Light パブリケーションのスキーマを表します。

このクラスを直接インスタンス化することはできません。パブリケーションのスキーマは DatabaseSchema.getPublicationSchema メソッドを使用して作成されます。

パブリケーション・マスクが必要な Ultra Light のメソッドには、実際にはチェック対象のパブリケーションのセットが必要です。セットは、個々のパブリケーションのパブリケーション・マスクの論理和で形成されます。次に例を示します。

```
pub1.getMask() | pub2.getMask()
```

特別な 2 つのマスク値は、DatabaseSchema オブジェクトによって提供されます。SYNC_ALL_DB は、データベース全体に対応します。SYNC_ALL_PUBS は、すべてのパブリケーションに対応します。

パブリケーション・マスクは、スキーマのアップグレード中に変更されることがあります。パブリケーションを正しく識別するには、名前でアクセスするか、キャッシュされているマスクをスキーマのアップグレード後にリフレッシュします。

getMask メソッド

このパブリケーションのパブリケーション・マスクを返します。

構文

UInt32 **getMask()**

備考

注意：パブリケーションの ID、マスク、カウントは、スキーマのアップグレード中に変更されることがあります。パブリケーションを正しく識別するには、名前でアクセスするか、キャッシュされているマスク、カウントをスキーマのアップグレード後にリフレッシュします。

getName メソッド

このパブリケーションの名前を返します。

構文

String **getName()**

ResultSet クラス

Ultra Light データベースの結果セットを示します。実行時に PreparedStatement.executeQuery を使用して作成されます。

プロパティ

このクラスのプロパティは、次のとおりです。

プロパティ	説明
ResultSetSchema schema (読み込み専用)	この結果セットのスキーマ。このプロパティが有効なのは、その準備文が開かれている間だけです。
NULL_TIMESTAMP_VAL	タイムスタンプ値が NULL であることを示す定数。

appendBytes メソッド

指定したバイト配列の指定したサブセットを、指定した SQLType.LONGBINARY カラムの新しい値に追加します。

構文

```
appendBytes(
  UInt16 columnID,
  Array value,
  UInt32 srcOffset,
  UInt32 count
)
```

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。
- **srcOffset**　カラムの現在の新しい値に追加する値。
- **count**　コピーされるバイト数。

備考

配列 **value** の srcOffset (0 から始まります) から **srcOffset+count-1** までの位置のバイトが、指定したカラムの値に追加されます。挿入時には、insertBegin は新しい値をカラムのデフォルト値に初期化します。ローのデータは、**insert** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

次のいずれかに該当する場合、コード SQLCode.SQLE_INVALID_PARAMETER とともにエラーがスローされ、追加先は修正されません。

- **value** 引数が null である
- **srcOffset** 引数が負の値である
- **count** 引数が負の値である
- **srcOffset+count** がソース配列の長さ **value.length** よりも大きい

その他のエラーの場合は、それに応じたエラー・コードとともに **SQLException** がスローされます。

appendStringChunk メソッド

指定した文字列を指定した SQLType.LONGVARCHAR カラムの新しい値に追加します。

構文

```
appendStringChunk(
  UInt16 columnID,
  String value
)
```

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

例

次の文は、文字列 **XYZ** のインスタンス 100 個を最初のカラムの値に追加します。

```
for ( i = 0; i < 100; i++ ){
  t.AppendStringChunk( 1, "XYZ" );
}
```

close メソッド

このオブジェクトに関連付けられているリソースを解放します。

構文

close()

deleteRow メソッド

現在の行を削除します。

構文

deleteRow( )

備考

deleteRow を行う前に updateBegin を呼び出してください。

getBoolean メソッド

指定したカラムの値を Boolean として返します。

構文

Boolean **getBoolean(** UInt16 *index* **)**

パラメータ

- **index**　カラムの ID 番号。結果セットの最初のカラムの ID は 1 です。

getBytes メソッド

指定したカラムの値のバイト配列を返します。

構文

Array **getBytes(** UInt16 *index* **)**

パラメータ

- **index**　カラムの ID 番号。結果セットの最初のカラムの ID は 1 です。

備考

SQLType.BINARY 型、SQLType.LONGBINARY 型のカラムの場合にのみ有効です。

getBytesSection メソッド

指定したソース・オフセットで始まる、指定した SQLType.LONGBINARY または SQLType.BINARY カラムの内容のサブセットを、コピー先のバイト配列の指定したオフセットにコピーします。

構文

```
UInt32 getBytesSection(
  UInt16 index,
  UInt32 srcOffset,
  Array  dst,
  UInt32 dstOffset,
  UInt32 count
)
```

パラメータ

index　バイナリ・データを含むカラムの 1 から始まる序数。

srcOffset　ソース・バイト配列への、ゼロから始まる相対オフセット。ソース・オフセットは、0 以上であることが必要です。それ以外の場合は、SQLE_INVALID_PARAMETER エラーが発生します。64K を超えるバッファも許されます。

dst　コピー先バイト配列。

dstOffset　コピー先バイト配列への、ゼロから始まる相対オフセット。コピー先オフセットは、0 以上であることが必要です。それ以外の場合は、SQLE_INVALID_PARAMETER エラーが発生します。64K を超えるバッファも許されます。

count　移動するバイト数。count は 0 以上であることが必要です。

戻り値

読み込まれたバイト数。

備考

コピー元の配列の srcOffset (0 から始まります) から srcOffset+count-1 までの位置のバイトが、コピー先の配列の dstOffset から dstOffset+count-1 までの位置に、それぞれコピーされます。指定されたバイト数がコピーされる前に、ソース値の末尾が検出された場合は、コピー先の配列の残りは変更されないままになります。

次のいずれかに該当する場合、エラーがスローされ、SQLError コードが SQLE_INVALID_PARAMETER に設定され、コピー先は修正されません。

- dst 引数が null である
- srcOffset 引数が負の値である
- dstOffset 引数が負の値である
- count 引数が負の値である
- dstOffset + count がコピー先の配列の長さ dst.length よりも長い

エラー・セット

SQLE_CONVERSION_ERROR　カラムのデータ型が BINARY でも LONG BINARY でもない場合、エラーが発生します。

SQLE_INVALID_PARAMETER　カラムのデータ型が BINARY でオフセットが 0 でも 1 でもない、またはデータ長が 0 より小さい場合、エラーが発生します。

カラムのデータ型が LONG BINARY で、オフセットが 1 より小さい場合も、エラーが発生します。

getDate メソッド

Date として値を返します。

構文

Date **getDate(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getDouble メソッド

Double として値を返します。

構文

Double **getDouble(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getFloat メソッド

指定されたカラムの値を返します。

構文

Float **getFloat(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getInt メソッド

指定されたカラムの値を返します。

構文

UInt32 **getInt(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getLong メソッド

指定されたカラムの値を返します。

構文

Int64 **getLong(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getRowCount メソッド

結果セット内のロー数を返します。

構文

UInt32 **getRowCount( )**

getShort メソッド

Int16 として値を返します。

構文

Int16 **getShort(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getString メソッド

String として値を返します。

構文

String **getString(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getStringChunk メソッド

指定したオフセットで始まる、指定した SQLType.LONGVARCHAR カラムの値のサブセットを String オブジェクトにコピーします。

構文

String **getStringChunk(**
  UInt16 *index*,
  UInt32 *srcOffset*,
  UInt32 *count*
)

パラメータ

- **index**　結果セットで取得する 1 から始まる序数。
- **srcOffset**　文字列値で 0 から始まる開始位置。
- **count**　コピーされる文字数。

戻り値

指定された文字数がコピーされた文字列。

getTime メソッド

Date として値を返します。

構文

Date **getTime(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getTimestamp メソッド

Date として値を返します。

構文

Date **getTimestamp(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getULong メソッド

64 ビット符号なし整数として値を返します。

構文

UInt64 **getULong(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getUUID メソッド

UUID としてカラムの値を返します。

構文

UUID **getUUID(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

備考

カラムは長さが 16 の SQLType.BINARY 型である必要があります。

isBOF メソッド

現在のローの位置が最初のローの前なら **true** を返し、そうでなければ **false** を返します。

構文

Boolean **isBOF( )**

isEOF メソッド

現在のローの位置が最後のローの後なら **true** を返し、そうでなければ **false** を返します。

構文

Boolean **isEOF( )**

isNull メソッド

値が null なら **true** を返し、そうでなければ **false** を返します。

構文

Boolean **isNull(** Uint16 *index* **)**

パラメータ

index　カラムのインデックス値。

isOpen メソッド

ResultSet が開いている場合は **true**、閉じている場合は **false** を返します。

構文

Boolean **isOpen( )**

moveAfterLast メソッド

ULResultSet の最後のローの後に移動します。

構文

moveAfterLast( )

moveBeforeFirst メソッド

最初のローの前に移動します。

構文

moveBeforeFirst( )

moveFirst メソッド

最初のローに移動します。

構文

Boolean **moveFirst( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveLast メソッド

最後のローに移動します。

構文

Boolean **moveLast( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveNext メソッド

次のローに移動します。

構文

Boolean **moveNext( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

movePrevious メソッド

前のローに移動します。

構文

Boolean **movePrevious( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveRelative メソッド

いくつかのローを、現在のローを基準にして相対的に移動します。

構文

Boolean **moveRelative(** Int32 *index* **)**

パラメータ

index　移動するローの数。値は、正、負、または 0 です。

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

備考

結果セットでのカーソルの現在位置を基準にして、正のインデックス値は結果セット内を前に移動し、負のインデックス値は結果セット内を後ろに移動し、0 はカーソルを移動しません。

setBoolean メソッド

指定したカラムの値を、**boolean** を使用して設定します。

構文

setBoolean(short *columnID*, boolean *value*)

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、**update** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setBytes メソッド

指定したカラムの値を、**byte** 配列を使用して設定します。

構文

setBytes(UInt16 *columnID*, Array *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

SQLType.BINARY 型、**SQLType.LONGBINARY** 型のカラムにのみ適しています。ローのデータは、**update** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setDate メソッド

指定したカラムの値を、**Date** を使用して設定します。

構文

setDate(UInt16 *columnID*, Date *value***)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setDateTime メソッド

指定したカラムの値を、**Date** を使用して設定します。

構文

setDateTime(UInt16 *columnID*, Date *value***)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setDouble メソッド

指定したカラムの値を、**double** を使用して設定します。

構文

setDouble(UInt16 *columnID*, Double *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setFloat メソッド

指定したカラムの値を、**float** を使用して設定します。

構文

setFloat(UInt16 *columnID*, Float *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setInt メソッド

指定したカラムの値を、Integer を使用して設定します。

構文

setInt(UInt16 *columnID*, Int32 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setLong メソッド

指定したカラムの値を、Int64 を使用して設定します。

構文

setLong(UInt16 *columnID*, Int64 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setNull メソッド

カラムに SQL NULL を設定します。

構文

setNull(UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

データは、update を実行するまでは、実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setShort メソッド

指定したカラムの値を、UInt16 を使用して設定します。

構文

setShort(UInt16 *columnID*, Int16 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setString メソッド

指定したカラムの値を、String を使用して設定します。

構文

setString(UInt16 *columnID*, String *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setTime メソッド

指定したカラムの値を、Date を使用して設定します。

構文

setTime(UInt16 *columnID*, Date *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setTimestamp メソッド

指定したカラムの値を、Date を使用して設定します。

構文

setTimestamp(UInt16 *columnID*, Date *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setULong メソッド

指定したカラムの値を、符号なし値として扱われる 64 ビット整数を使用して設定します。

構文

setULong(UInt16 *columnID*, UInt64 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setUUID メソッド

指定したカラムの値を、UUID を使用して設定します。

構文

setUUID(UInt16 *columnID*, UUID *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。長さが 16 の SQLType.BINARY 型のカラムの場合にのみ有効です。

参照

- 「UUID の使用」 『Mobile Link - サーバ管理』

update メソッド

現在のカラム値 (set メソッドを使用して指定されます) で新しいローを更新します。

構文

update()

備考

update を行う前に updateBegin を呼び出してください。

updateBegin メソッド

この結果セットの現在のローを更新する準備を行います。

構文

updateBegin()

備考

カラム値は、適切な set*Type* メソッドを呼び出すことによって修正します。

データは、update を実行するまでは、実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

ResultSetSchema クラス

Ultra Light の結果セットのスキーマを表します。

getColumnCount メソッド

このカーソル内のカラム数を返します。

構文

UInt16 **getColumnCount()**;

備考

カラム ID の範囲は、1 ー getColumnCount です。

カラムの ID とカウントは、スキーマのアップグレード中に変更されることがあります。カラムを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getColumnID メソッド

指定したカラムのカラム ID を返します。

構文

UInt16 **getColumnID(**String *name***)**

パラメータ

- **name**　カラムの名前。

備考

カラム ID の範囲は、1 ー getColumnCount() です。

カラムの ID とカウントは、スキーマのアップグレード中に変更されることがあります。カラムを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getColumnName メソッド

指定したカラム ID で識別されたカラムの名前を返します。

構文

String **getColumnName(**UInt16 *columnID***)**

パラメータ

- **columnID**　カラムの ID。**columnID** は、**[1,getColumnCount()]** の範囲内である必要があります。

備考

カラムの ID とカウントは、スキーマのアップグレード中に変更されることがあります。カラムを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getColumnPrecision メソッド

指定したカラムの精度を返します。

構文

Int32 **getColumnPrecision(**String *name***)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnPrecisionByColID メソッド

カラムの精度を返します。

構文

Int32 **getColumnPrecisionByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。結果セットの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnScale メソッド

カラムの位取りを返します。

構文

Int32 **getColumnScale(**String *name***)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnScaleByColID メソッド

カラムの位取りを返します。

構文

Int32 **getColumnScaleByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。結果セットの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnSize メソッド

指定したカラムのサイズを返します。

構文

UInt32 **getColumnSize(**String *name***)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnSizeByColID メソッド

カラムのサイズを返します。

構文

UInt32 **getColumnSizeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。結果セットの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnSQLType メソッド

指定したカラムの SQL データ型を返します。

構文

UInt16 **getColumnSQLType(**String *name***)**

パラメータ

- **name**　カラムの名前。

getColumnSQLTypeByColID メソッド

カラムの SQLType を示す SQLType 列挙整数を返します。

構文

UInt16 **getColumnSQLTypeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。結果セットの最初のカラムの ID 値は 1 です。

isOpen メソッド

結果セットが開いている場合は **true**、閉じている場合は **false** を返します。

構文

Boolean **isOpen()**;

SQLError クラス

Ultra Light for M-Business Anywhere でレポートされる可能性のある SQL コードを列挙します。このクラスは静的定数を提供し、直接インスタンス化することはできません。

メンバ	説明
SQLE_AGGREGATES_NOT_ALLOWED	「集合関数の不正な使用です。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ALIAS_NOT_UNIQUE	「エイリアス '%1' がユニークではありません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ALIAS_NOT_YET_DEFINED	「エイリアス '%1' の定義は、最初の参照前に記述する必要があります。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_AMBIGUOUS_INDEX_NAME	「インデックス名 '%1' があいまいです。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_BAD_ENCRYPTION_KEY	「暗号化キーが不正であるか、見つかりません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_BAD_PARAM_INDEX	「入力パラメータ・インデックスが範囲外です。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CANNOT_ACCESS_FILESYSTEM	「デバイス上のファイルシステムにアクセスできません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CANNOT_CHANGE_USER_NAME	「前回のアップロードのステータスが不明の場合、同期の user_name は変更できません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CANNOT_CONVERT	「不正なデータ変換」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CANNOT_EXECUTE_STMT	「文を実行することができませんでした。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CANNOT_MODIFY	「テーブル '%2' のカラム '%1' を変更できません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CLIENT_OUT_OF_MEMORY	「クライアントでメモリが不足しています。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_COLUMN_AMBIGUOUS	「カラム '%1' が複数のテーブルで見つかりました。相関名が必要です。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_COLUMN_CANNOT_BE_NULL	「テーブル '%2' のカラム '%1' は NULL 値を持つことはできません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_COLUMN_IN_INDEX	「インデックスのカラムを変更することはできません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。

メンバ	説明
SQLE_COLUMN_NOT_FOUND	「カラム '%1' が見つかりません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_COLUMN_NOT_INDEXED	「カラム '%1' は、それを含んでいるテーブルのどのインデックスにも属していません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_COLUMN_NOT_STREAMABLE	「操作が失敗しました。カラム '%1' のタイプがストリーミングをサポートしていません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_COMMUNICATIONS_ERROR	「通信エラーが発生しました。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CONNECTION_ALREADY_EXISTS	「この接続はすでに存在します。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CONNECTION_NOT_FOUND	「接続が見つかりません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CONNECTION_RESTORED	「Ultra Light の接続がリストアされました。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CONSTRAINT_NOT_FOUND	「制約 '%1' が見つかりません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CONVERSION_ERROR	「値 %1 をデータ型 %2 に変換できません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_COULD_NOT_FIND_FUNCTION	「ダイナミック・ライブラリ '%2' に '%1' が見つかりませんでした。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_COULD_NOT_LOAD_LIBRARY	「ダイナミック・ライブラリ '%1' をロードできませんでした。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CURSOR_ALREADY_OPEN	「カーソルはすでに開いています。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CURSOR_NOT_OPEN	「カーソルが開きません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CURSOR_RESTORED	「Ultra Light のカーソル (あるいは結果セットまたはテーブル) がリストアされました。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_CURSOROP_NOT_ALLOWED	「不正なカーソル処理をしようとしました。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DATABASE_ERROR	「データベースの内部エラー %1 -- トランザクションはロールバックされました。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。

メンバ	説明
SQLE_DATABASE_NAME_REQUIRED	「サーバを起動するには、データベース名が必要です。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DATABASE_NOT_CREATED	「データベースの作成に失敗しました : %1」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DATATYPE_NOT_ALLOWED	「式にサポートされていないデータ型があります。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DBSPACE_FULL	「データベース領域が最大ファイル・サイズに達しています。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DESCRIBE_NONSELECT	「SELECT 文以外は記述できません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DIV_ZERO_ERROR	「ゼロで除算しようとしました。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DOWNLOAD_CONFLICT	「既存のローと競合しているため、ダウンロードに失敗しました。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DOWNLOAD_RESTART_FAILED	「ダウンロードをリトライできません。アップロードが完了していません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DROP_DATABASE_FAILED	「データベース '%1' の削除に失敗しました。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DUPLICATE_CURSOR_NAME	「カーソル名 '%1' はすでに存在します。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DUPLICATE_FOREIGN_KEY	「テーブル '%2' の外部キー '%1' は、既存の外部キーと重複しています。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DUPLICATE_OPTION	「オプション '%1' が複数回指定されています。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_DYNAMIC_MEMORY_EXHAUSTED	「動的メモリが足りません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ENCRYPTION_INITIALIZATION_FAILED	「暗号化 DLL を初期化できませんでした : '%1'」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ENGINE_ALREADY_RUNNING	「データベース・サーバはすでに起動しています。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ENGINE_NOT_MULTIUSER	「データベース・サーバはマルチユーザ・モードで実行していません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ERROR	「実行時 SQL エラーです -- %1」『SQL Anywhere 10 - エラー・メッセージ』を参照。

メンバ	説明
SQLE_ERROR_CALLING_FUNCTION	「外部関数の呼び出しのためのリソースを割り付けられませんでした。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ERROR_IN_ASSIGNMENT	「割り当てのエラー」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_EXPRESSION_ERROR	「'%1' 付近に無効な式があります。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_FEATURE_NOT_ENABLED	「呼び出そうとしたメソッドは、お使いのアプリケーションでは使用できません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_FILE_BAD_DB	「指定されたデータベースを開始できません。'%1' は有効なデータベース・ファイルではありません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_FILE_IN_USE	「指定されたデータベース・ファイルはすでに使用されています。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_FILE_NOT_DB	「指定されたデータベースを開始できません。'%1' はデータベースではありません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_FILE_VOLUME_NOT_FOUND	「データベース '%1' に対して指定したファイルシステム・ボリュームが見つかりません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_FILE_WRONG_VERSION	「指定されたデータベースを開始できません。'%1' は異なるバージョンのソフトウェアで作成されています。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_FOREIGN_KEY_NAME_NOT_FOUND	「外部キー '%1' は見つかりません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_IDENTIFIER_TOO_LONG	「識別子 '%1' が長すぎます。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INCORRECT_VOLUME_ID	「'%1' のボリューム ID が不正です。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INDEX_NAME_NOT_UNIQUE	「インデックス名 '%1' はユニークではありません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INDEX_NOT_FOUND	「インデックス '%1' が見つかりません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INDEX_NOT_UNIQUE	「テーブル '%2' のインデックス '%1' はユニークでなければなりません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INTERRUPTED	「文の実行がユーザによって中断させられました。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。

メンバ	説明
SQLE_INVALID_CONSTRAINT_REF	「制約 '%1' への参照または操作が無効です。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_DESCRIPTOR_INDEX	「記述子のインデックスが正しくありません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_DESCRIPTOR_NAME	「SQL 記述子名が正しくありません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_DISTINCT_AGGREGATE	「グループ化されたクエリに、複数の異なる集合関数が含まれています。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_FOREIGN_KEY	「テーブル '%2' の外部キー '%1' に対応するプライマリ・キーの値がありません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_FOREIGN_KEY_DEF	「外部キーのカラム '%1' にプライマリー・キーと異なる定義があります。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_GROUP_SELECT	「'%1' に対する関数またはカラムの参照も GROUP BY 句に記述する必要があります。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_LOGON	「ユーザ ID またはパスワードが無効です。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_OPTION_SETTING	「不正なオプション '%1' を設定しています。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_OPTION_VALUE	「'%1' は '%2' に対して無効な値です。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_ORDER	「ORDER BY 句の指定が不正です。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_PARAMETER	「不正なパラメータです。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_PARSE_PARAMETER	「解析エラーです : %1」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_PUBLICATION_MASK	「指定されたパブリケーション・マスクが不正です。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_SQL_IDENTIFIER	「SQL の識別子が無効です。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_INVALID_UNION	「UNION、INTERSECT、または EXCEPT の select リストの長さが一致していません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。

メンバ	説明
SQLE_KEYLESS_ENCRYPTION	「このデータベースはキー不使用の暗号化を使用するため、要求された操作を実行できません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_LOCKED	「'%2' のローは、ユーザ '%1' によってロックされています。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_MEMORY_ERROR	「メモリエラー -- トランザクションはロールバックされました。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_NAME_NOT_UNIQUE	「アイテム '%1' はすでに存在しています。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_NO_COLUMN_NAME	「抽出されたテーブル '%1' にはカラム %2 に対する名前がありません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_NO_CURRENT_ROW	「カーソルの現在のローがありません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_NO_INDICATOR	「NULL に対して、インジケータ変数が用意されていません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_NO_MATCHING_SELECT_ITEM	「派生テーブル '%1' の select リストに '%2' と一致する式がありません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_NO_PRIMARY_KEY	「テーブル '%1' にはプライマリ・キーが定義されていません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_NOERROR	SQLE_NOERROR(0) – このコードは、エラーまたは警告がなかったことを示します。
SQLE_NON_UPDATEABLE_COLUMN	「式を更新できません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_NON_UPDATEABLE_CURSOR	「FOR UPDATE が READ ONLY カーソルに誤って指定されました。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_NOT_IMPLEMENTED	「'%1' の機能は実装されていません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_NOT_SUPPORTED_IN_ULTRALITE	「Ultra Light では使用できない機能です。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_NOTFOUND	「ローが見つかりません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ONLY_ONE_TABLE	「カーソルの INSERT/DELETE は、1 度に 1 つのテーブルにしかできません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_OVERFLOW_ERROR	「値 %1 は、対象先にとって大きすぎます。」『SQL Anywhere 10 - エラー・メッセージ』を参照。

メンバ	説明
SQLE_PAGE_SIZE_INVALID	「無効なデータベース・ページ・サイズです。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_PARTIAL_DOWNLOAD_NOT_FOUND	「部分ダウンロードが見つかりませんでした。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_PERMISSION_DENIED	「パーミッションがありません : %1」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_PRIMARY_KEY_NOT_UNIQUE	「テーブル '%1' のプライマリ・キーがユニークではありません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_PRIMARY_KEY_TWICE	「テーブルに 2 つのプライマリ・キーを定義することはできません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_PRIMARY_KEY_VALUE_REF	「テーブル '%1' 内のローのプライマリ・キーがテーブル '%3' 内の外部キー '%2' によって参照されています。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_PUBLICATION_NOT_FOUND	「パブリケーション '%1' が見つかりません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_PUBLICATION_PREDICATE_IGNORED	「パブリケーションの述部は評価されませんでした。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_RESOURCE_GOVERNOR_EXCEEDED	「%1 のリソース・ガバナーが制限を超えています。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY	「参照整合性を保つためにテーブル %1 からローが削除されました。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_SCHEMA_UPGRADE_NOT_ALLOWED	「スキーマのアップグレードは現在認められません。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_SERVER_SYNCHRONIZATION_ERROR	「サーバ %1 でエラーが発生したため、同期に失敗しました。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_START_STOP_DATABASE_DENIED	「データベースの起動／停止の要求は拒否されました。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_STATEMENT_ERROR	「SQL 文にエラーがあります。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_STRING_RIGHT_TRUNCATION	「文字列データの右側がトランケートされます。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_SUBQUERY_SELECT_LIST	「select リストの中にカラムが 2 つ以上指定されています。」『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_SYNC_INFO_INVALID	「同期の情報が不完全か無効です。'%1' を確認してください。」『SQL Anywhere 10 - エラー・メッセージ』を参照。

メンバ	説明
SQLE_SYNC_INFO_REQUIRED	「同期の情報が指定されていません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_SYNC_NOT_REENTRANT	「同期処理に戻ることができませんでした。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_SYNC_STATUS_UNKNOWN	「最後の同期アップロードのステータスは不明です。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_SYNTAX_ERROR	「'%1' %2 の近くに構文エラーがあります。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_TABLE_ALREADY_INCLUDED	「テーブル '%1' はすでにインクルードされています。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_TABLE_IN_USE	「テーブルは使用されています。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_TABLE_NOT_FOUND	「テーブル '%1' は見つかりませんでした。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_TOO_MANY_BLOB_REFS	「BLOB への参照が多すぎます。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_TOO_MANY_CONNECTIONS	「データベース・サーバに接続できる限界数を超えています。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_TOO_MANY_PUBLICATIONS	「パブリケーション・マスクに指定されているパブリケーションが多すぎます。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_TOO_MANY_TEMP_TABLES	「接続しているテンポラリ・テーブルが多すぎます。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_TOO_MANY_USERS	「データベースのユーザが多すぎます。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ULTRALITE_DATABASE_NOT_FOUND	「データベース '%1' が見つかりませんでした。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ULTRALITE_OBJ_CLOSED	「閉じられたオブジェクトに対する操作は無効です。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_ULTRALITE_WRITE_ACCESS_DENIED	「書き込みアクセスが拒否されました。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_UNABLE_TO_CONNECT	「データベースが起動できません -- %1」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_UNABLE_TO_START_DATABASE	「指定されたデータベースを起動できません : %1」 『SQL Anywhere 10 - エラー・メッセージ』を参照。

メンバ	説明
SQLE_UNABLE_TO_START_DATABASE_VER_NEWER	「指定されたデータベースを起動できません : データベース %1 を起動するにはサーバをアップグレードする必要があります。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_UNCOMMITTED_TRANSACTIONS	「コミットされていないトランザクションの同期またはアップグレードはできません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_UNKNOWN_FUNC	「関数 '%1' はありません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_UNKNOWN_OPTION	「'%1' は認識できないオプションです。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_UNKNOWN_USERID	「'%1' というユーザ ID はありません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_UNRECOGNIZED_OPTION	「オプション '%1' が認識されません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_UPLOAD_FAILED_AT_SERVER	「同期サーバがアップロードのコミットに失敗しました。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_VALUE_IS_NULL	「要求されたデータ型として NULL の結果を返すことができません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_VARIABLE_INVALID	「無効なホスト変数です。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_WRONG_NUM_OF_INSERT_COLS	「INSERT コマンドへの値が正しくありません。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。
SQLE_WRONG_PARAMETER_COUNT	「関数 '%1' のパラメータ数が誤りです。」 『SQL Anywhere 10 - エラー・メッセージ』を参照。

SQLType クラス

この列挙は、テーブル・カラムの型として使用されている、Ultra Light SQL データベースの型を定数としてリストします。

定数	Ultra Light データベースの型
BAD_INDEX	
S_LONG	INT
U_LONG	UNSIGNED INT
S_SHORT	SMALLINT
U_SHORT	UNSIGNED SMALLINT
S_BIG	BIGINT
U_BIG	UNSIGNED BIGINT
TINY	TYNY INT
BIT	BIT
TIMESTAMP	TIMESTAMP
DATE	DATE
TIME	TIMESTAMP
DOUBLE	DOUBLE
REAL	REAL
NUMERIC	NUMERIC
BINARY	BINARY
CHAR	CHAR または VARCHAR
LONGVARCHAR	LONG VARCHAR
LONGBINARY	LONG BINARY
MAX_INDEX	

toString メソッド

指定された SQL カラム型定数の文字列名、または認識されない型の場合は BAD_SQL_TYPE を返します。

構文

String **toString(**UInt16 *code***)**

パラメータ

- **code**　SQL カラム型定数。

SyncParms クラス

Ultra Light データベースの同期方法を定義する同期パラメータを表します。それぞれの接続には、固有の SyncParms インスタンスがあります。

定数

定数	値	説明
STREAM_TYPE_TCPIP	0	TCP/IP ストリーム
STREAM_TYPE_HTTP	1	HTTP ストリーム
STREAM_TYPE_HTTPS	2	HTTPS 同期
STREAM_TYPE_TLS	3	TLS 同期
STREAM_TYPE_HOTSYNC	4	HotSync 同期用

getAuthenticationParms メソッド

カスタム・ユーザ認証スクリプトに渡されたパラメータ、またはパラメータが指定されなかった場合は null を返します。

構文

Array **getAuthenticationParms()**

getCheckpointStore メソッド

クライアントが追加のチェックポイントを実行する場合は true、必須のチェックポイントだけを実行する場合は false を返します。

構文

Boolean **getCheckpointStore()**

getDisableConcurrency メソッド

同時同期が無効な場合は true、有効な場合は false を返します。

構文

Boolean **getDisableConcurrency()**

getDownloadOnly メソッド

アップロードが無効な場合は true、有効な場合は false を返します。

構文

Boolean **getDownloadOnly()**

getKeepPartialDownload メソッド

部分的なダウンロードが保持される場合は true、ロールバックされる場合は false を返します。

構文

Boolean getKeepPartialDownload**()**

getNewPassword メソッド

次回の同期以降、Mobile Link ユーザに関連付けられる新しいパスワードを返します。

構文

String **getNewPassword()**

getPartialDownloadRetained メソッド

通信エラーが原因でダウンロードが失敗し、部分的なダウンロードが保持された場合は true、ダウンロードが中断されなかった場合または部分的なダウンロードがロールバックされた場合は false を返します。

構文

Boolean **getPartialDownloadRetained( )**

getPassword メソッド

setUserName で指定されたユーザの Mobile Link パスワードを返します。

構文

String **getPassword()**;

getPingOnly メソッド

クライアントがサーバに ping のみを実行する場合は true、クライアントが同期を実行する場合は false を返します。

構文

Boolean **getPingOnly()**

getPublicationMask メソッド

同期させるパブリケーションを返します。

構文

UInt32 **getPublicationMask()**;

備考

PublicationSchema クラスを参照してください。

getResumePartialDownload メソッド

前回の部分的なダウンロードが再開される場合は true、ロールバックされる場合は false を返します。

構文

Boolean **getResumePartialDownload( )**

getSendColumnNames メソッド

クライアントが同期中にカラム名を Mobile Link サーバに送信する場合は true、送信しない場合は false を返します。

構文

Boolean **getSendColumnNames()**

getSendDownloadAck メソッド

クライアントが Mobile Link サーバにダウンロード確認を送信する場合は true、送信しない場合は false を返します。

構文

Boolean **getSendDownloadAck()**

getStream メソッド

同期に使用する Mobile Link 同期ストリームのタイプを返します。

構文

UInt16 **getStream()**;

getStreamParms メソッド

同期ストリームに使用されるすべてのネットワーク・プロトコル・オプションを含む文字列を返します。

構文

String **getStreamParms()**;

getUploadOnly メソッド

ダウンロードが無効な場合は true、有効な場合は false を返します。

構文

Boolean **getUploadOnly()**

getUserName メソッド

Mobile Link ユーザ名を返します。

構文

String **getUserName()**

getVersion メソッド

使用される同期スクリプトを示すバージョン文字列を返します。

構文

String **getVersion()**

setAuthenticationParms メソッド

カスタム・ユーザ認証スクリプト (Mobile Link authenticate_parameters 接続イベント) のパラメータを指定します。

構文

setAuthenticationParms(Array *value* **)**

パラメータ

- **value**　それぞれに認証パラメータが格納された、文字列の配列 (配列のエントリが null であると、同期エラーになります)。

備考

最初の 255 文字のみが使用されます。また、各文字列は 128 文字以下である必要があります (長すぎる文字列は、Mobile Link に送信されるときにトランケートされます)。

setCheckpointStore メソッド

クライアントが、追加のストア・チェックポイントを実行して、同期中にデータベース・ストアのサイズの増加を制御するかどうかを指定します。

構文

setCheckpoint16Store(Boolean *value* **)**

パラメータ

- **value**　追加のチェックポイントを実行する場合は true、必須のチェックポイントだけを実行する場合は false に設定します。

備考

チェックポイント操作は、アプリケーションの I/O 操作を増やすため、同期が低速になります。このオプションは、多くの更新を伴う大量のダウンロードに最適です。低速なフラッシュ・メモリを使用するデバイスでは、パフォーマンスの低下が望ましくないことがあります。

setDisableConcurrency メソッド

同期の実行中に Ultra Light への同時アクセスを無効にするか、有効にするかを指定します。

構文

setDisableConcurrency(Boolean *value* **)**;

パラメータ

- **value**　同時同期を無効にする場合は true、有効にする場合は false に設定します。

備考

デフォルトでは、1 つのスレッドが同期しているときに、別のスレッドが Ultra Light の操作を行うことがあります。同時同期が無効になっていると、別のスレッドは同期が完了するまで Ultra Light の呼び出しをブロックします。

setDownloadOnly メソッド

同期時のアップロードを無効にするか、有効にするかを指定します。

構文

setDownloadOnly(Boolean *value* **)**

パラメータ

- **value**　アップロードを無効にする場合は true、有効にする場合は false に設定します。

setKeepPartialDownload メソッド

同期時の部分的なダウンロードを無効にするか、有効にするかを指定します。

構文

setKeepPartialDownload(Boolean *value* **)**

パラメータ

- **value**　同期中に部分的なダウンロードを保持する場合は true、破棄する場合は false に設定します。

備考

Ultra Light では、通信エラーが原因で失敗したダウンロードを再開することができます。Ultra Light は、ダウンロードを受信しながら処理します。ダウンロードが中断した場合は、部分的なダウンロード・トランザクションがデータベース内に残るため、次の同期中に再開できます。

Ultra Light で部分的なダウンロードを保存する必要があることを示すには、Connection.syncParms.setKeepPartialDownload(true); と指定します。指定しないと、エラーが発生した場合にダウンロードがロールバックされます。

部分的なダウンロードが保持された場合、connection.synchronize の終了時に、出力フィールド connection.SyncResult.getPartialDownloadRetained が true に設定されます。getPartialDownloadRetained が設定されている場合は、ダウンロードを再開できます。再開するには、connection.syncParms.setResumePartialDownload(true) を指定して connection.synchronize を呼び出します。多くの場合、別の通信エラーの発生に備えて、KeepPartialDownload も true に設定する必要があります。ダウンロードが省略された場合は、アップロードは行われません。

再開したダウンロードで受信するダウンロードは、最初にダウンロードを開始したときと同じものです。最新のデータが必要な場合は、再開された特別なダウンロードが完了した直後に、もう一度ダウンロードを行うことができます。

ダウンロードを再開する場合、SyncParms フィールドの多くは関係ありません。たとえば、PublicationMask フィールドは使用されません。受信するパブリケーションは、最初のダウンロード時に要求したものです。設定する必要があるフィールドは、setResumePartialDownload(boolean) と setUserName(String) だけです。setKeepPartialDownload(boolean) フィールド、setDownloadOnly(boolean) フィールド、setDisableConcurrency(boolean) フィールドを必要に応じて設定すると、正常に機能します。

部分的なダウンロードが存在するが、このダウンロードが必要ではなくなった場合は、Connection.rollbackPartialDownload を呼び出して、失敗したダウンロード・トランザクションをロールバックできます。また、同期をもう一度実行したときに ResumePartialDownload を指定しなかった場合は、部分的なダウンロードがロールバックされてから、次の同期が開始されます。

「同期の障害処理の方法」『Mobile Link - クイック・スタート』を参照してください。

setMBAServer メソッド

Mobile Link ホストとポートの同期パラメータを、M-Business Client によって使用される M-Business Anywhere Server の同期パラメータにすばやく設定できます。

構文

setMBAServer(String *host*, String *port*, String *url_suffix* **)**

パラメータ

- **host** M-Business Anywhere Server のホストまたは IP 値。host が null の場合、現在の M-Business Anywhere ホストに設定されます。
- **port** M-Business Anywhere Server が受信しているポート。port が null の場合、現在の M-Business Anywhere ポート値に設定されます。
- **url_suffix** これは M-Business Anywhere の *sync.conf* ファイルで設定された url_suffix パラメータに対応します。

備考

データを Mobile Link サーバとの間でルート指定するには、M-Business Anywhere 用の Mobile Link リダイレクタを使用します。

ワンタッチ同期を使用している場合は、Connection.saveSyncParms を使用して同期パラメータを保存してください。

M-Business Anywhere リダイレクタを使用して HTTP データベース・トラフィックをルート指定するように M-Business Server を設定する方法については、「M-Business Anywhere リダイレクタ」『Mobile Link - サーバ管理』を参照してください。

setMBAServerWithMoreParms メソッド

Mobile Link ホストとポートの同期パラメータを、M-Business Client によって使用される M-Business Anywhere Server の同期パラメータにすばやく設定できます。

構文

setMBAServerWithMoreParms(String *host*, String *port*, String *url_suffix*, String *additional***)**

パラメータ

- **host** M-Business Anywhere Server のホストまたは IP 値。host が null の場合、現在の M-Business Anywhere ホストに設定されます。
- **port** M-Business Anywhere Server が受信しているポート。port が null の場合、現在の M-Business Anywhere ポート値に設定されます。

- **url_suffix**　これは M-Business Anywhere の *sync.conf* ファイルで設定された url_suffix パラメータに対応します。
- **additional**　このパラメータには、先行するパラメータでは扱われない追加のストリーム・パラメータが含まれる可能性があります (プロキシ・ホスト、プロキシ・ポート、セキュリティ関連パラメータなど)。host、port、url_suffix の情報を指定する必要がある場合は、前述した setMBAServer メソッドを使用することもできます。

備考

データを Mobile Link サーバとの間でルート指定するには、M-Business Anywhere 用の Mobile Link リダイレクタを使用します。

このメソッドは setMBAServer で提供される機能を拡張しており、ユーザは additional パラメータ内で別のパラメータを指定できます。

ワンタッチ同期を使用している場合は、Connection.saveSyncParms を使用して同期パラメータを保存してください。

M-Business Anywhere リダイレクタを使用して HTTP データベース・トラフィックをルート指定するように M-Business Server を設定する方法については、「M-Business Anywhere リダイレクタ」 『Mobile Link - サーバ管理』を参照してください。

setNewPassword メソッド

setUserName で指定されたユーザの新しい Mobile Link パスワードを設定します。

構文

setNewPassword(String *value* **)**

パラメータ

- **value**　Mobile Link ユーザの新しいパスワード。

備考

新しいパスワードが有効になるのは、次の同期の後です。

setPassword メソッド

setUserName で指定されたユーザの Mobile Link パスワードを設定します。

構文

setPassword(String *value* **)**

パラメータ

- **value**　Mobile Link ユーザのパスワード。

備考

このユーザ名とパスワードは他のデータベース・ユーザ ID やパスワードとは別のもので、アプリケーションを Mobile Link サーバに対して識別し、認証するために使用されます。

setPingOnly メソッド

実際に同期を行う代わりに、クライアントが Mobile Link サーバに ping のみを行うかどうかを指定します。

構文

setPingOnly(Boolean *value* **)**;

パラメータ

- **value**　Mobile Link サーバに ping のみを実行する場合は true、同期を実行する場合は false に設定します。

setPublicationMask メソッド

同期させるパブリケーションを指定します。

構文

setPublicationMask(UInt16 *mask* **)**

パラメータ

- **mask**　同期対象のパブリケーションのセット。

備考

PublicationSchema クラスを参照してください。

setSendColumnNames メソッド

同期中に、クライアントが Mobile Link サーバにカラム名を送信するかどうかを指定します。

構文

setSendColumnNames(Boolean *value* **)**

パラメータ

- **value**　カラム名を送信する場合は true、送信しない場合は false に設定します。

備考

このパラメータは、ダイレクト・ロー・ハンドリングで使用されます。

setSendDownloadAck メソッド

同期中に、クライアントが Mobile Link サーバにダウンロード確認を送信するかどうかを指定します。

構文

setSendDownloadAck(Boolean *value* **)**

パラメータ

- **value**　ダウンロード確認 (正または負) を送信する場合は true、送信しないことをサーバに通知する場合は false に設定します。

備考

ダウンロード確認は、ダウンロードがリモートで完全に適用されてコミットされた後 (正の確認)、またはダウンロードに失敗した後 (負の確認) に送信されます。

クライアントがダウンロード確認を送信する場合、Mobile Link サーバのデータベース・ワーカ・スレッドは、クライアントがダウンロードを適用してコミットするまで待機します。クライアントがダウンロード確認を送信しない場合、Mobile Link サーバは、次の同期のため、より早く解放されます。

setStream メソッド

同期に使用するように Mobile Link 同期ストリームを設定します。

構文

setStream(UInt16 *value* **)**

パラメータ

- **value**　同期に使用する Mobile Link 同期ストリームのタイプ。有効な選択肢のリストについては、「定数」 126 ページを参照してください。

備考

ほとんどの同期ストリームでは、Mobile Link サーバのアドレスを識別したり、その他の動作を制御したりするパラメータが必要です。これらのパラメータは、**setStreamParms()** メソッドで指定します。

デフォルトのストリーム・タイプは STREAM_TYPE_TCPIP です。

setStreamParms メソッド

同期ストリームの設定パラメータを設定します。

構文

setStreamParms(String *value* **)**

パラメータ

- **value** 同期ストリームに使用されるすべてのネットワーク・プロトコル・オプションを含む文字列。オプションは、name=value の組み合わせをセミコロンで区切ったリスト (**"param1=value1;param2=value2"**) で指定します。

備考

特定のストリームのタイプの設定方法については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Mobile Link - クライアント管理』を参照してください。

setUploadOnly メソッド

同期時のダウンロードを無効にするか、有効にするかを指定します。

構文

setUploadOnly(Boolean *value* **)**

パラメータ

- **value** ダウンロードを無効にする場合は true、有効にする場合は false に設定します。

setUserName メソッド

Mobile Link サーバが Mobile Link クライアントをユニークに識別するユーザ名を設定します。

構文

setUserName(String *value* **)**

パラメータ

- **value** Mobile Link ユーザ名。

備考

Mobile Link では、この値を使用して、ダウンロードする内容の決定、同期ステータスの記録、同期中の割り込みからの復帰を行います。このユーザ名とパスワードは他のデータベース・ユーザ ID やパスワードとは別のもので、アプリケーションを Mobile Link サーバに対して識別し、認証するために使用されます。

setVersion メソッド

使用する同期スクリプト・バージョンを指定します。

構文

setVersion(String *value* **)**

パラメータ

- **value** スクリプト・バージョン文字列。

備考

統合データベースの同期スクリプトは、それぞれバージョン文字列でマーク付けされます。たとえば、異なる文字列バージョンによって特定される 2 種類の download_cursor スクリプトがあります。Ultra Light アプリケーションは、バージョン文字列により、同期スクリプトのセットから選択できます。

SyncResult クラス

前回の同期の Ultra Light for M-Business Anywhere メソッドのステータスを表します。それぞれの接続には、固有の SyncResult インスタンスがあります。

このクラスを直接インスタンス化することはできません。

getAuthStatus メソッド

前回行われた同期の認証ステータス・コードを返します。

構文

UInt16 **getAuthStatus()**

getIgnoredRows メソッド

前回の同期中にアップロードされたローが無視された場合は true、アップロードされたローが無視されなかった場合は false を返します。

構文

Boolean **getIgnoredRows()**

パラメータ

- **return** アップロードされたローが無視された場合は true、無視されなかった場合は false。

getPartialDownloadRetained メソッド

ダウンロードが中断され、部分的なダウンロードが保持された場合は true、ダウンロードが中断されなかった場合または部分的なダウンロードがロールバックされた場合は false を返します。

構文

Boolean getPartialDownloadRetained**()**

getStreamErrorCode メソッド

同期ストリーム処理によってレポートされるエラー・コードを表す整数を返します。

構文

UInt16 **getStreamErrorCode( )**

パラメータ

- **return** 同期ストリームによってレポートされるエラー・コード。

備考

「Mobile Link 通信エラー・メッセージ」 『SQL Anywhere 10 - エラー・メッセージ』を参照してください。

getStreamErrorContext メソッド

ストリーム・エラーが発生したときに実行される基本的なネットワーク・オペレーションを返します。

構文

UInt16 **getStreamErrorContext( )**

備考

次に既知のコンテキストを示します。

値	コンテキスト
0	不明
1	登録
2	登録の解除
3	作成
4	破棄
5	開く
6	閉じる
7	読み込み
8	書き込み
9	書き込みのフラッシュ
10	書き込み終了
11	読み込み終了
12	解放
13	シャットダウン

getStreamErrorID メソッド

エラーをレポートするネットワーク・レイヤを返します。

構文

UInt32 **getStreamErrorID( )**

備考

値はネットワーク・レイヤの ID です。次に既知の ID を示します。

値	説明
0	TCP/IP ストリーム
3	HotSync 同期用
7	HTTP ストリーム
8	HTTPS 同期

getStreamErrorSystem メソッド

ストリーム・エラー・システム固有のコードを返します。

構文

UInt16 **getStreamErrorSystem( )**

パラメータ

- **return** システム固有のエラー・コード。

getTimestamp メソッド

前回の同期のタイムスタンプを返します。

構文

Date **getTimestamp( )**

getUploadOK メソッド

前回のアップロード同期が成功であった場合は true、不成功であった場合は false を返します。

構文

Boolean **getUploadOK()**

TableSchema クラス

Ultra Light のテーブルのスキーマを表します。

getColumnCount メソッド

このテーブル内の 1 から始まるカラム番号を返します。

構文

UInt16 **getColumnCount( )**

備考

カラム ID の範囲は、1 ー getColumnCount() です。

getColumnDefaultValue メソッド

指定したカラムのデフォルト値、またはデフォルト値が **null** の場合は null を返します。

構文

String **getColumnDefaultValue(** String *name* **)**

パラメータ

- **name**　カラムの名前。

getColumnDefaultValueByColID メソッド

カラムのデフォルト値、またはデフォルト値が **null** の場合は null を返します。

構文

String **getColumnDefaultValueByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

getColumnID メソッド

指定したカラムの 1 から始まる ID を返します。

構文

UInt16 **getColumnID(** String *name* **)**

パラメータ

- **name** カラムの名前。

getColumnName メソッド

指定されたカラムの名前を返します。

構文

String **getColumnName(** UInt16 *colID* **)**

パラメータ

- **colID** カラムの 1 から始まるカラム ID。

getColumnPartitionSize メソッド

カラムのグローバル・オートインクリメントの分割サイズを Double で表される 64 ビット符号なし数値として返します。

構文

UInt64 **getColumnPartitionSize(** String *name* **)**

パラメータ

- **name** カラムの名前。

備考

テーブルのすべてのグローバル・オートインクリメント・カラムは、同じグローバル・オートインクリメントの分割サイズを共有します。

getColumnPartitionSizeByColID メソッド

カラムのグローバル・オートインクリメントの分割サイズを Double で表される 64 ビット符号なし数値として返します。

構文

UInt64 **getColumnPartitionSizeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID** カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

テーブルのすべてのグローバル・オートインクリメント・カラムは、同じグローバル・オートインクリメントの分割サイズを共有します。

getColumnPrecision メソッド

カラムの精度を返します。

構文

Int32 **getColumnPrecision(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnPrecisionByColID メソッド

カラムの精度を返します。

構文

Int32 **getColumnPrecisionByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnScale メソッド

カラムの位取りを返します。

構文

Int32 **getColumnScale(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnScaleByColID メソッド

カラムの位取りを返します。

構文

Int32 **getColumnScaleByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnSize メソッド

カラムのサイズを返します。

構文

UInt32 **getColumnSize(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.BINARY 型、または SQLType.CHAR 型である必要があります。

getColumnSizeByColID メソッド

カラムのサイズを返します。

構文

UInt32 **getColumnSizeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.BINARY 型、または SQLType.CHAR 型である必要があります。

getColumnSQLType メソッド

カラムの SQLType を示す SQLType 列挙整数を返します。

構文

Int16 **getColumnSQLType(** String *name* **)**

パラメータ

- **name**　カラムの名前。

getColumnSQLTypeByColID メソッド

カラムの SQLType を示す SQLType 列挙整数を返します。

構文

Int16 **getColumnSQLTypeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

getIndex メソッド

指定したインデックスのインデックス・スキーマを返します。

構文

IndexSchema **getIndex(** String *name* **)**

パラメータ

- **name**　インデックスの名前。

getIndexCount メソッド

このテーブルのインデックス数を返します。

構文

UInt16 **getIndexCount( )**

備考

インデックス ID の範囲は、1 ー **getIndexCount()** です。

注意：インデックスの ID とカウントは、スキーマのアップグレード中に変更されることがあります。インデックスを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getIndexName メソッド

指定したインデックス ID で識別されたインデックスの名前を返します。

構文

String **getIndexName(** UInt16 *indexID* **)**

パラメータ

- **indexID**　インデックスの ID。**indexID** は、**[1,getIndexCount()]** の範囲内である必要があります。

備考

注意：インデックスの ID とカウントは、スキーマのアップグレード中に変更されることがあります。インデックスを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getName メソッド

このテーブルの名前を返します。

構文

String **getName( )**

getOptimalIndex メソッド

指定したカラムを使用してテーブルを検索するために最適なインデックスを返します。

構文

IndexSchema **getOptimalIndex(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

指定したカラムは、インデックス内の最初のカラムですが、インデックスには複数のカラムがある場合があります。

getPrimaryKey メソッド

このテーブルのプライマリ・キーのインデックス・スキーマを返します。

構文

IndexSchema **getPrimaryKey( )**

getUploadUnchangedRows メソッド

テーブルが、すべてのローをアップロードするようにマーク付けされている場合は true、一部のローをアップロードしないようにマーク付けされている場合は false を返します。

構文

Boolean **getUploadUnchangedRows( )**

備考

このメソッドで true が返されるテーブルでは、その同期時に、変更済みのローと未変更のローが常にアップロードされます。このようなテーブルは、「完全同期」テーブルと呼ばれることもあります。

isColumnAutoIncrement メソッド

カラムがオートインクリメントされる場合は true、オートインクリメントされない場合は false を返します。

構文

Boolean **isColumnAutoIncrement(** String *name* **)**

パラメータ

- **name**　カラムの名前。

isColumnAutoIncrementByColID メソッド

カラムがオートインクリメントされる場合は true、オートインクリメントされない場合は false を返します。

構文

Boolean **isColumnAutoIncrementByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

isColumnCurrentDate メソッド

カラムのデフォルトが現在の日付に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentDate(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.DATE 型である必要があります。

isColumnCurrentDateByColID メソッド

カラムのデフォルトが現在の日付に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentDateByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.DATE 型である必要があります。

isColumnCurrentTime メソッド

カラムのデフォルトが現在の時刻に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentTime(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.TIME 型である必要があります。

isColumnCurrentTimeByColID メソッド

カラムのデフォルトが現在の時刻に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentTimeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.TIME 型である必要があります。

isColumnCurrentTimestamp メソッド

カラムのデフォルトが現在のタイムスタンプに設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentTimestamp(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.TIMESTAMP 型である必要があります。

isColumnCurrentTimestampByColID メソッド

カラムのデフォルトが現在のタイムスタンプに設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentTimestampByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.TIME 型である必要があります。

isColumnGlobalAutoIncrement メソッド

カラムのデフォルトがグローバル・オートインクリメントに設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnGlobalAutoIncrement(** String *name* **)**

パラメータ

- **name**　カラムの名前。
- **return**　カラムがグローバル・オートインクリメントされる場合は true、グローバル・オートインクリメントされない場合は false。

isColumnGlobalAutoincrementByColID メソッド

カラムのデフォルトがグローバル・オートインクリメントに設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnGlobalAutoincrementByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

isColumnNewUUID メソッド

カラムのデフォルトが新しい UUID に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnNewUUID(** String *name* **)**

パラメータ

- **name**　カラムの名前。

isColumnNewUUIDByColID メソッド

カラムのデフォルトが新しい UUID に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnNewUUIDByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

isColumnNullable メソッド

カラムが null 入力可能な場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnNullable(** String *name* **)**

パラメータ

- **name**　カラムの名前。

isColumnNullableByColID メソッド

カラムのデフォルトが新しい UUID に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnNullableByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

isInPublication メソッド

テーブルがパブリケーション内にある場合は true、パブリケーション内にない場合は false を返します。

構文

Boolean **isInPublication(** String *pubName* **)**

パラメータ

- **pubName**　パブリケーションの名前。

isNeverSynchronized メソッド

テーブルが、まったく同期されないようにマーク付けされている場合は true、まったく同期されないようにマーク付けされていない場合は false を返します。

構文

Boolean **isNeverSynchronized( )**

備考

このメソッドで true が返されるテーブルは、パブリケーションに含まれている場合でもまったく同期されません。このようなテーブルは、「未同期」テーブルと呼ばれることもあります。

ULTable クラス

Ultra Light テーブルを表します。

プロパティ

このクラスのプロパティは、次のとおりです。

プロパティ	説明
TableSchema schema (読み込み専用)	この結果セットのスキーマ。このプロパティが有効なのは、その準備文が開かれている間だけです。
NULL_TIMESTAMP_VAL	タイムスタンプ値が NULL であることを示す定数。

AppendBytes メソッド

指定したバイト配列の指定したサブセットを、指定した SQLType.LONGBINARY カラムの新しい値に追加します。

構文

AppendBytes(
  UInt16 *columnID*,
  Array *value*,
  UInt32 *srcOffset*,
  UInt32 *count*
)

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。
- **srcOffset**　カラムの現在の新しい値に追加する値。
- **count**　コピーされるバイト数。

備考

配列 **value** の srcOffset (0 から始まります) から **srcOffset+count-1** までの位置のバイトが、指定したカラムの値に追加されます。挿入時には、insertBegin は新しい値をカラムのデフォルト値に初期化します。ローのデータは、**insert** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

次のいずれかに該当する場合、コード SQLCode.SQLE_INVALID_PARAMETER とともにエラーがスローされ、追加先は修正されません。

- **value** 引数が null である

- **srcOffset** 引数が負の値である
- **count** 引数が負の値である
- **srcOffset+count** がソース配列の長さ **value.length** よりも大きい

その他のエラーの場合は、それに応じたエラー・コードとともに **SQLException** がスローされます。

AppendStringChunk メソッド

指定した文字列を指定した SQLType.LONGVARCHAR カラムの新しい値に追加します。

構文

```
AppendStringChunk(
  UInt16 columnID,
  String value
)
```

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

例

次の文は、文字列 **XYZ** のインスタンス 100 個を最初のカラムの値に追加します。

```
for ( i = 0; i < 100; i++ ){
  t.AppendStringChunk( 1, "XYZ" );
}
```

deleteRow メソッド

現在の行を削除します。

構文

```
deleteRow( )
```

deleteAllRows メソッド

テーブルのすべてのローを削除します。

構文

```
deleteAllRows( )
```

備考

アプリケーションによっては、テーブル内のローをすべて削除してから、新しいデータ・セットをテーブルにダウンロードする方が便利なことがあります。Connection.startSynchronizationDelete メソッドを使用すると、統合データベースからは削除しないで Ultra Light データベースからローを削除できます。

findBegin メソッド

このテーブルで新規に検索を実行する準備を行います。

構文

findBegin( )

備考

検索する値は、このテーブルを開いたインデックス内のカラムで適切な set*Type* メソッドを呼び出して指定します。

findFirst メソッド

テーブルを先頭から順方向に移動しながら、現在のインデックスの値かそのセット全体に完全に一致するローを検索します。

構文

Boolean **findFirst( )**

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値と完全に一致した最初のローで停止します。失敗すると、カーソル位置は最後のローの後ろ (isEOF) になります。

検索を行う前に findBegin メソッドを呼び出してください。

参照

- 「 findBegin メソッド」 154 ページ
- 「 isEOF メソッド」 102 ページ

findFirstForColumns メソッド

テーブルを先頭から順方向に移動しながら、現在のインデックスの値かそのセットの一部に完全に一致するローを検索します。

構文

Boolean **findFirstForColumns(**
  UInt16 *numColumns*
)

パラメータ

- **numColumns**　複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定し、**numColumns** の値を **1** に指定します。

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値と完全に一致した最初のローで停止します。失敗すると、カーソル位置は最後のローの後ろ (isEOF) になります。

検索を行う前に findBegin メソッドを呼び出してください。

参照

- 「 findBegin メソッド」 154 ページ
- 「 isEOF メソッド」 102 ページ

findLast メソッド

テーブルを最後から逆方向に移動しながら、現在のインデックスの値またはそのセット全体に完全に一致するローを検索します。

構文

Boolean **findLast( )**

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値と完全に一致した最初のローで停止します。失敗すると、カーソル位置は最初のローの前 (isBOF) になります。

検索を行う前に findBegin メソッドを呼び出してください。

参照

- 「 findBegin メソッド」 154 ページ
- 「 isBOF メソッド」 102 ページ

findLastForColumns メソッド

テーブルを最後から逆方向に移動しながら、現在のインデックスの値またはそのセットの一部に完全に一致するローを検索します。

構文

Boolean **findLastForColumns(** UInt16 *numColumns* **)**

パラメータ

- **numColumns**　複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定し、**numColumns** の値を **1** に指定します。

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値と完全に一致した最初のローで停止します。失敗すると、カーソル位置は最初のローの前 (isBOF) になります。

検索を行う前に findBegin メソッドを呼び出してください。

参照

- 「 findBegin メソッド」 154 ページ
- 「 isBOF メソッド」 102 ページ

findNext メソッド

現在の位置からテーブルを順方向に移動しながら、次のローが現在のインデックスの値かそのセット全体に完全に一致するかどうかを調べて、findFirst 検索を続行します。

構文

Boolean **findNext( )**

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

インデックスの値と完全に一致すると、カーソルは次のローで停止します。失敗すると、カーソル位置は最後のローの後ろ (isEOF) になります。

検索されるカラムの値がローの更新において修正された場合の findNext メソッドの動作は不確定です。

findNextForColumns メソッド

現在の位置からテーブルを順方向に移動しながら、次のローが現在のインデックスの値かそのセットの一部に完全に一致するかどうかを調べて、findFirst 検索を続行します。

構文

Boolean **findNextForColumns(** UInt16 *numColumns*)

パラメータ

- **numColumns**　複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定し、**numColumns** の値を **1** に指定します。

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

インデックスの値と完全に一致すると、カーソルは次のローで停止します。失敗すると、カーソル位置は最後のローの後ろ (isEOF) になります。

検索されるカラムの値がローの更新において修正された場合の findNext メソッドの動作は不確定です。

findPrevious メソッド

現在の位置からテーブルを逆方向に移動しながら、前のローが現在のインデックスの値かそのセット全体に完全に一致するかどうかを調べて、findLast 検索を続行します。

構文

Boolean **findPrevious( )**

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

インデックスの値と完全に一致すると、カーソルは前のローで停止します。失敗すると、カーソル位置は最初のローの前 (isBOF) になります。

検索されるカラムの値がローの更新において修正された場合の findPrevious メソッドの動作は不確定です。

findPreviousForColumns メソッド

現在の位置からテーブルを逆方向に移動しながら、前のローが現在のインデックスの値かそのセットの一部に完全に一致するかどうかを調べて、findLast 検索を続行します。

構文

Boolean **findPreviousForColumns(**
UInt16 *numColumns*
)

パラメータ

- **numColumns** 複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定し、**numColumns** の値を **1** に指定します。

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

インデックスの値と完全に一致すると、カーソルは前のローで停止します。失敗すると、カーソル位置は最初のローの前 (isBOF) になります。

検索されるカラムの値がローの更新において修正された場合の findPrevious メソッドの動作は不確定です。

getBoolean メソッド

指定したカラムの値を Boolean として返します。

構文

Boolean **getBoolean(** UInt16 *index* **)**

パラメータ

- **index** カラムの ID 番号。結果セットの最初のカラムの ID は 1 です。

getBytes メソッド

指定したカラムの値のバイト配列を返します。

構文

Array **getBytes(** UInt16 *index* **)**

パラメータ

- **index** カラムの ID 番号。結果セットの最初のカラムの ID は 1 です。

備考

SQLType.BINARY 型、SQLType.LONGBINARY 型のカラムの場合にのみ有効です。

getBytesSection メソッド

指定したオフセットで始まる、指定した SQLType.LONGBINARY カラムの内容のサブセットを、コピー先のバイト配列の指定したオフセットにコピーします。

構文

```
UInt32 getBytesSection(
  UInt16 index
  UInt32 srcOffset,
  Array  dst,
  UInt32 dstOffset,
  UInt32 count
)
```

パラメータ

index　バイナリ・データを含むカラムの 1 から始まる序数。

srcOffset　カラム値の開始位置。最初の値はゼロです。

dst　コピー先の配列。

dstOffset　コピー先の配列の開始位置。

count　コピーされるバイト数。

戻り値

読み込まれたバイト数。

備考

コピー元のカラムの srcOffset (0 から始まります) から srcOffset+count-1 までの位置のバイトが、コピー先の配列の dstOffset から dstOffset+count-1 までの位置に、それぞれコピーされます。count のバイト数がコピーされる前に、値の末尾が検出された場合は、コピー先の配列の残りは変更されないままになります。

次のいずれかに該当する場合、エラーがスローされ、Connection.sqlCode が SQLError.SQLE_INVALID_PARAMETER に設定され、コピー先は修正されません。

- dst 引数が null である
- srcOffset 引数が負の値である
- dstOffset 引数が負の値である
- count 引数が負の値である
- dstOffset + count がコピー先の配列の長さ dst.length よりも長い

getDate メソッド

Date として値を返します。

構文

Date **getDate(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getDouble メソッド

Double として値を返します。

構文

Double **getDouble(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getFloat メソッド

指定されたカラムの値を返します。

構文

Float **getFloat(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getInt メソッド

指定されたカラムの値を返します。

構文

Int32 **getInt(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getLong メソッド

指定されたカラムの値を返します。

構文

Int64 **getLong(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getRowCount メソッド

結果セット内のロー数を返します。

構文

UInt32 **getRowCount( )**

getShort メソッド

Int16 として値を返します。

構文

Int16 **getShort(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getString メソッド

String として値を返します。

構文

String **getString(** UInt32 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getStringChunk メソッド

指定したオフセットで始まる、指定した SQLType.LONGVARCHAR カラムの値のサブセットを String オブジェクトにコピーします。

構文

String **getStringChunk(**
  UInt16 *index*,
  UInt32 *srcOffset*,
  UInt32 *count*
)

パラメータ

- **index**　結果セットで取得する 1 から始まる序数。
- **srcOffset**　文字列値で 0 から始まる開始位置。
- **count**　コピーされる文字数。

戻り値

指定された文字数がコピーされた文字列。

getTime メソッド

Date として値を返します。

構文

Date **getTime(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getTimestamp メソッド

Date として値を返します。

構文

Date **getTimestamp(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getULong メソッド

64 ビット符号なし整数として値を返します。

構文

UInt64 **getULong(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

getUUID メソッド

UUID としてのカラムの値を返します。

構文

UUID **getUUID(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる序数。

備考

カラムは長さが 16 の SQLType.BINARY 型である必要があります。

insert メソッド

現在のカラム値 (set メソッドを使用して指定されます) で新しいローを挿入します。

構文

insert()

備考

挿入を行う前に insertBegin を呼び出してください。

insertBegin メソッド

現在のすべてのカラムをデフォルト値に設定して、このテーブルに新しいローを挿入する準備を行います。

構文

insertBegin( )

備考

適切な set*Type* メソッドを呼び出して、挿入するデフォルト以外の値を指定します。

insert メソッドが実行されないと、ローが実際に挿入されることも、ロー内のデータが実際に変更されることもありません。また、その変更がコミットされないかぎり、永続化されません。

lookupBackward メソッド

テーブルを最後から逆方向に移動しながら、現在のインデックスの値またはそのセット全体に一致するか、その値より小さい値を持つローを検索します。

構文

Boolean **lookupBackward( )**

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値に一致する最初のローか、それより少ない値の最初のローで停止します。失敗した場合 (検索する値より小さい値のローがない場合)、カーソル位置は最初のローの前 (isBOF) になります。

検索を行う前に lookupBegin メソッドを呼び出してください。

lookupBackwardForColumns メソッド

テーブルを最初から逆方向に移動しながら、現在のインデックスの値またはそのセットの一部に一致するか、その値より小さい値を持つローを検索します。

構文

Boolean **lookupBackwardForColumns(** UInt16 *numColumns* **)**

パラメータ

- **numColumns**　複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定し、**numColumns** の値を **1** に指定します。

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値に一致する最初のローか、それより少ない値の最初のローで停止します。失敗した場合 (検索する値より小さい値のローがない場合)、カーソル位置は最初のローの前 (isBOF) になります。

検索を行う前に lookupBegin メソッドを呼び出してください。

lookupBegin メソッド

このテーブルで新規に検索を実行する準備を行います。

構文

lookupBegin( )

備考

検索する値は、このテーブルを開いたインデックス内のカラムで適切な set*Type* メソッドを呼び出して指定します。

lookupForward メソッド

テーブルを最初から順方向に移動しながら、現在のインデックスの値またはそのセット全体に一致するか、その値より大きい値を持つローを検索します。

構文

Boolean **lookupForward( )**

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値に一致するか、それより大きい値の最初のローで停止します。失敗した場合 (検索する値より大きい値のローがない場合)、カーソル位置は最後のローの後ろ (isEOF) になります。

検索を行う前に lookupBegin メソッドを呼び出してください。

lookupForwardForColumns メソッド

テーブルを最初から順方向に移動しながら、現在のインデックスの値またはそのセットの一部に一致するか、その値より大きい値を持つローを検索します。

構文

Boolean **lookupForwardForColumns(** UInt16 *numColumns* **)**

パラメータ

- **numColumns**　複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定し、**numColumns** の値を **1** に指定します。

戻り値

成功した場合は **true**、失敗した場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値に一致するか、それより大きい値の最初のローで停止します。失敗した場合 (検索する値より大きい値のローがない場合)、カーソル位置は最後のローの後ろ (isEOF) になります。

検索を行う前に lookupBegin メソッドを呼び出してください。

isBOF メソッド

成功した場合は **true**、失敗した場合は **false** を返します。

構文

Boolean **isBOF( )**

isEOF メソッド

成功した場合は **true**、失敗した場合は **false** を返します。

構文

Boolean **isEOF( )**

isNull メソッド

値が null なら **true** を返し、そうでなければ **false** を返します。

構文

Boolean **isNull(** Uint16 *index* **)**

パラメータ

index　カラムのインデックス値。

isOpen メソッド

ResultSet が開いている場合は **true**、閉じている場合は **false** を返します。

構文

Boolean **isOpen( )**

moveAfterLast メソッド

ULResultSet の最後のローの後に移動します。

構文

Boolean **moveAfterLast( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveBeforeFirst メソッド

最初のローの前に移動します。

構文

Boolean **moveBeforeFirst( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveFirst メソッド

最初のローに移動します。

構文

Boolean **moveFirst( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveLast メソッド

最後のローに移動します。

構文

Boolean **moveLast( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveNext メソッド

次のローに移動します。

構文

Boolean **moveNext( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

movePrevious メソッド

前のローに移動します。

構文

Boolean **movePrevious( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveRelative メソッド

いくつかのローを、現在のローを基準にして相対的に移動します。

構文

Boolean **moveRelative(** Int32 *index* **)**

パラメータ

index　移動するローの数。値は、正、負、または 0 です。

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

備考

結果セットでのカーソルの現在位置を基準にして、正のインデックス値は結果セット内を前に移動し、負のインデックス値は結果セット内を後ろに移動し、0 はカーソルを移動しません。

open メソッド

プライマリ・キーを使用して、このテーブルをデータ・アクセス用に開きます。

構文

open()

openWithIndex メソッド

指定したインデックスを使用して、このテーブルをデータ・アクセス用に開きます。

構文

openWithIndex(String *index*)

パラメータ

- **index**　テーブルを開くインデックスの名前。null の場合は、プライマリ・キーが使用されます。

setBoolean メソッド

指定したカラムの値を、**boolean** を使用して設定します。

構文

setBoolean(short *columnID*, boolean *value*)

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、**insert** または **update** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を **false** に設定します。

```
t.setBoolean( 1, false );
```

setBytes メソッド

指定したカラムの値を、**byte** 配列を使用して設定します。

構文

setBytes(UInt16 *columnID*, Array *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

SQLType.BINARY 型、**SQLType.LONGBINARY** 型のカラムにのみ適しています。ローのデータは、**insert** または **update** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を設定します。

```
var blob = new Array( 3 );
blob[ 0 ] = 78;
```

```
blob[ 1 ] = 0'
blob[ 2 ] = 68;
t.setBytes( 1, blob );
```

setDate メソッド

指定したカラムの値を、**Date** を使用して設定します。

構文

setDate(UInt16 *columnID*, Date *value***)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 2004/09/27 に設定します。

```
t.setDate(
  1, new Date( 2004,9,27,0,0,0,0 )
);
```

setDouble メソッド

指定したカラムの値を、**double** を使用して設定します。

構文

setDouble(UInt16 *columnID*, Double *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の例は、最初のカラムの値を設定します。

```
t.setDouble( 1, Number.MAX_VALUE );
```

setFloat メソッド

指定したカラムの値を、Float を使用して設定します。

構文

setFloat(UInt16 *columnID*, Float *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を設定します。

```
t.setFloat(
  1,
  (2 - Math.pow(2,-23)) * Math.pow(2,127)
);
```

setInt メソッド

指定したカラムの値を、Integer を使用して設定します。

構文

setInt(UInt16 *columnID*, Int32 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 2147483647 に設定します。

```
t.setInt( 1, 2147483647 );
```

setLong メソッド

指定したカラムの値を、Int64 を使用して設定します。

構文

setLong(UInt16 *columnID*, Int64 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 9223372036854770000 に設定します。

```
t.setLong( 1, 9223372036854770000 );
```

setNull メソッド

カラムに SQL NULL を設定します。

構文

setNull(UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

データは、insert または update を実行するまでは、実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setShort メソッド

指定したカラムの値を、UInt16 を使用して設定します。

構文

setShort(UInt16 *columnID*, Int16 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 32767 に設定します。

```
t.setShort( 1, 32767 );
```

setString メソッド

指定したカラムの値を、String を使用して設定します。

構文

setString(UInt16 *columnID*, String *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を **abc** に設定します。

```
t.setString( 1, "abc" );
```

setTime メソッド

指定したカラムの値を、Date を使用して設定します。

構文

setTime(UInt16 *columnID*, Date *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 18:02:13:0000 に設定します。

```
t.setTime(
  1, new Date( 1966,4,1,18,2,13,0 )
);
```

setTimestamp メソッド

指定したカラムの値を、Date を使用して設定します。

構文

setTimestamp(UInt16 *columnID*, Date *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 1966/04/01 18:02:13:0000 に設定します。

```
t.setTimestamp(
  1, new Date( 1966,4,1,18,2,13,0 )
);
```

setToDefault メソッド

指定したカラムの値を、そのデフォルト値に設定します。

構文

setToDefault(UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setULong メソッド

指定したカラムの値を、符号なし値として扱われる 64 ビット整数を使用して設定します。

構文

setULong(UInt16 *columnID*, UInt64 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を設定します。

```
t.setULong(
  1, 9223372036854770000 * 4096
);
```

setUUID メソッド

指定したカラムの値を、UUID を使用して設定します。

構文

setUUID(UInt16 *columnID*, UUID *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。長さが 16 の SQLType.BINARY 型のカラムの場合にのみ有効です。

例

次の文は、テーブルで最初のカラムに新しい UUID 値を設定します。

```
t.setUUID( 1, conn.getNewUUID(); );
```

参照

- 「UUID の使用」『Mobile Link - サーバ管理』

truncate メソッド

テーブル内のすべてのローを削除し、STOP SYNCHRONIZATION DELETE を一時的にアクティブにします。

構文

truncate()

update メソッド

現在のカラム値 (set メソッドを使用して指定されます) で新しいローを更新します。

構文

update()

備考

update を行う前に updateBegin を呼び出してください。

updateBegin メソッド

このテーブルの現在のローを更新する準備を行います。

構文

updateBegin()

備考

カラム値は、適切な set*Type* メソッドを呼び出すことによって修正します。

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

テーブルを開くのに使用されるインデックス内のカラムを修正すると、アクティブな検索処理に予期しない影響を及ぼします。テーブルのプライマリ・キー内のカラムは更新できません。

UUID クラス

UUID を説明します。UUID (ユニバーサル・ユニーク識別子) または GUID (グローバル・ユニーク識別子) は、すべてのコンピュータやデータベースの間でユニークになるように生成された値です。UUID は、SQLType.BINARY(16) 値として Ultra Light データベースに格納され、ローをユニークに識別するために使用できます。UUID クラスは不変の UUID を格納します。

UUID は、その UUID を作成した Connection と関連付けされ、その接続が閉じると文字列に変換できなくなります。

equals メソッド

この UUID が比較対象の引数と同じ場合は **true**、同じでない場合は **false** を返します。

構文

Boolean **equals(** UUID *other* **)**

パラメータ

- **other**　比較する UUID。

toString メソッド

この UUID の文字列表現を返します。

構文

String **toString()**

備考

文字列は、フォーマット XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX (X は 16 進数の数字)、またはこの UUID に関連付けられた Connection が閉じている場合は null です。

索引

F

G

H

I

J

L

M

O

P

R

S

T

U

V

W

あ

え

お

か

き

け

こ

さ

し

す

せ

