SYBASE iAnywhere

Ultra Light
Java プログラミング

バージョン 12.0.1

2012 年 1 月

バージョン 12.0.1
2012 年 1 月

目次

はじめに

このマニュアルでは、Ultra Light J のプログラミングインターフェイスについて説明します。Ultra Lite J を使用して、Android および BlackBerry スマートフォン、Java ME および Java SE プラットフォーム向けのデータベースアプリケーションを開発し、配備することができます。

Ultra Light J

Ultra Light J は、次のデバイスおよびプラットフォーム用に設計された Ultra Light データベース管理システムの Java API です。

●「Android スマートフォン」

●OS 4.2 以降を実行している 「BlackBerry スマートフォン」

●コンピューターまたはデバイス上で実行する Java SE 1.6 以降

注意
Android スマートフォン用を開発する場合、Ultra Light は他のプラットフォーム用の Ultra Light と共通の C++ コードを共有し、その動作は他のプラットフォームの動作と同様です。Android で提供されていない API 用に SQL 文を使わずにテーブルとローにアクセスできるいくつかの機能が存在します。

参照

- 「Ultra Light 概要」『Ultra Light データベース管理とリファレンス』
- 「Windows Mobile 用 Ultra Light API の選択」『Ultra Light データベース管理とリファレンス』

システムの要件とサポート対象プラットフォーム

開発プラットフォーム

Ultra Light J アプリケーションを開発するには、以下が必要です。

●Eclipse などの Java IDE

●Java SE 1.6 以降

ターゲットプラットフォーム

Ultra Light J 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。

●「Android スマートフォン」

●OS 4.2 以降を実行している 「BlackBerry スマートフォン」

●コンピューターまたはデバイス上で実行する Java SE 1.6 以降

Ultra Light のサポート対象プラットフォームの詳細については、http://www.sybase.com/detail?id=1002288 を参照してください。

UltraLiteJ API のアーキテクチャー

Ultra Light J API のアーキテクチャーは、**com.ianywhere.ultralitej12** または **com.ianywhere.ultralitejni12** パッケージに定義されています。次のリストに、よく使用されるオブジェクトの一部を示します。

- **DatabaseManager** CreateDatabase など、データベース接続を管理するメソッドを提供します。
- **Connection** Ultra Light データベースへの接続を表します。Connection オブジェクトは 1 つまたは複数作成できます。
- **SyncParms** Ultra Light データベースを Mobile Link サーバーと同期させます。
- **PreparedStatement, ResultSet** Dynamic SQL 文の作成、クエリの記述、INSERT、UPDATE、DELETE 文の実行、プログラムによるデータベースの結果セットの制御を実行します。

参照

- 「Ultra Light J API リファレンス」71 ページ

Ultra Light J アプリケーションの開発

この項では、Ultra Light J API を使用したアプリケーション開発の情報について説明します。

Ultra Light J API は、Java アプリケーションにデータベース機能と同期を提供します。特に Android および BlackBerry スマートフォンと連携して動作するように設計されていますが、Java ME 環境および Java SE プラットフォームとも互換性があります。この API には、Ultra Light Java Edition データベースまたは Android 用 Ultra Light データベースに接続し、スキーマ操作を実行し、SQL 文を使用してデータを管理するために必要なすべてのメソッドが含まれています。データの暗号化や同期などの高度な操作もサポートされています。

Ultra Light J アプリケーション開発のクイックスタートガイド

Ultra Light J アプリケーションを作成するときは、一般にアプリケーションコードで次のデータ管理タスクを実行します。

1. Ultra Lght J API パッケージをユーザーの Java ファイルにインポートします。

   Ultra Light J パッケージの名前とロケーションは、アプリケーション開発の対象デバイスによって異なります。

2. 新しい Configuration オブジェクトを作成して、データベースを作成するか、データベースに接続します。

   Configuration オブジェクトは、クライアントデータベースの保存場所または作成先を定義します。また、データベースへの接続に必要なユーザー名とパスワードを指定します。異なるデバイスや非永続データベーストア用にさまざまな Configuration オブジェクトがあります。

3. 新しい Connection オブジェクトを作成します。

   Connection オブジェクトは、Configuration オブジェクトで定義されている指定内容に従ってクライアントデータベースに接続します。

4. SQL 文を使用してデータベーススキーマを作成または変更し、PreparedStatement インターフェイスを使用してデータベースを問い合わせます。

   SQL 文を使用して、データベースのテーブル、インデックス、外部キー、パブリケーションを作成または更新できます。

   PreparedStatement オブジェクトは、Connection オブジェクトに関連付けられているデータベースを問い合わせます。引数には、サポートされている SQL 文を文字列として指定します。PreparedStatement オブジェクトを使用して、データベースの内容を更新できます。

5. ResultSet オブジェクトを生成します。

ResultSet オブジェクトは、SQL SELECT 文を含む PreparedStatement が Connection オブジェクトによって実行されたときに作成されます。ResultSet オブジェクトを使用して、データベースのテーブルの内容を確認するために、クエリ結果のローを取得できます。

参照

- 「Android と BlackBerry のセットアップの考慮事項」4 ページ
- Configuration インターフェイス [Ultra LightJ]103 ページ
- Connection インターフェイス [Ultra Light J]105 ページ
- 「Ultra Light SQL 文」『Ultra Light データベース管理とリファレンス』
- PreparedStatement インターフェイス [Ultra Light J]178 ページ
- ResultSet インターフェイス [Ultra Light J]194 ページ

Android と BlackBerry のセットアップの考慮事項

Android または BlackBerry スマートフォン用のアプリケーションを開発する前に、Ultra Light API の次の考慮事項に留意してください。

JAR リソースファイル

Ultra Light J API のアプリケーションを設定する場合、適切な *UltraLiteJ12.jar* または *UltraLiteJNI12.jar* ファイルが使用されるようにプロジェクトが正しく設定されていることを確認してください。

Android 用 Ultra Light J API は、SQL Anywhere インストール環境の *UltraLite¥UltraLiteJ¥Android¥UltraLiteJNI12.jar* ファイルに格納されています。Android 開発プロジェクトを設定し、クラスパスに *UltraLiteJNI12.jar* ファイルを含める必要があります。 詳細については、「チュートリアル：Android アプリケーションの構築」37 ページを参照してください。

Android 開発の場合は、次の文を使用して、Ultra Light J パッケージを Java ファイルにインポートします。

```
import com.ianywhere.ultralitejni12.*;
```

Ultra Light J API for BlackBerry、Java ME、および Java SE は、SQL Anywhere インストール環境の *UltraLite¥UltraLiteJ¥* ディレクトリにあります。また、各ターゲットプラットフォームのサブディレクトリと *UltraLiteJ12.jar* ファイルがあります。BlackBerry 開発プロジェクトを設定し、クラスパスに *UltraLiteJNI12.jar* ファイルを含める必要があります。 詳細については、「チュートリアル：BlackBerry アプリケーションの構築」43 ページを参照してください。

BlackBerry 開発の場合は、次の文を使用して、Ultra Light J パッケージを Java ファイルにインポートします。

```
import com.ianywhere.ultralitej12.*;
```

このマニュアル内のすべてのサンプルコードとチュートリアルでは、上記の文が指定されていること、開発者が Eclipse での Java アプリケーション開発に精通していることを前提としています。

Java SE アプリケーションの注意

Java SE アプリケーションは、*UltraLiteJNI.jar* ファイルでサポートされません。*UltraLite¥UltraLiteJ¥J2SE¥UltraLiteJ12.jar* ファイルを使用する必要があります。

Ultra Light データベースと Ultra Light Java Edition データベース

Android スマートフォンでは、Ultra Light J は、Windows Mobile、iPhone、Windows に提供されているものと同じ Ultra Light データベース管理システムへのインターフェイスを提供します。BlackBerry スマートフォンでは、Ultra Light J は、Ultra Light Java Edition データベース管理システムへのインターフェイスを提供します。Ultra Light Java Edition は、Ultra Light と似た機能を備えてはいますが、同一ではありません。Ultra Light Java Edition データベースは、Ultra Light データベースと互換性がありません。

Android スマートフォンがサポートするのは、Ultra Light データベースだけです。これらは、Sybase Central または Ultra Light のコマンドラインユーティリティを使用して作成できます。Ultra Light データベースの作成の詳細については、「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。

BlackBerry スマートフォンがサポートするのは、Ultra Light Java Edition データベースだけです。Ultra Light Java Edition データベースの作成または使用の詳細については、「Ultra Light または Ultra Light Java Edition データベースの作成または接続」5 ページ を参照してください。

参照

- 「Windows Mobile 用 Ultra Light API の選択」『Ultra Light データベース管理とリファレンス』
- 「Ultra Light、Ultra Light Java Edition、SQL Anywhere の機能比較」『Ultra Light データベース管理とリファレンス』
- 「Ultra Light および Ultra Light Java Edition データベースの制限事項」『Ultra Light データベース管理とリファレンス』

Ultra Light または Ultra Light Java Edition データベースの作成または接続

Java アプリケーションでデータを操作するには、先にデータベースに接続する必要があります。この項では、Ultra Light J API を使用し、指定のパスワードで Ultra Light データベースまたは Ultra Light Java Edition データベースを作成するか、またはこれらのデータベースに接続する方法について説明します。

注意
Ultra Light J API を使用しないで Ultra Light データベースを作成する場合は、Sybase Central または Ultra Light のコマンドラインユーティリティを使用できます。 詳細については、「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。

Ultra Light J API を使用しないで Ultra Light Java Edition データベースを作成するには、次のいずれかのタスクを実行します。

- ULjLoad ユーティリティを使用してデータベースを作成します。 「Ultra Light Java Edition データベースロードユーティリティ (ULjLoad)」『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。
- ulunload および ULjLoad ユーティリティを使用して、Ultra Light データベースを変換します。「Ultra Light データベースのアンロードユーティリティ (ulunload)」『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。
- Ultra Light Java データベースを SD カードにコピーするか、ファイル転送メカニズムを使用してデータベースを Mobile Link から転送することによって、BlackBerry スマートフォンに Java SE アプリケーションを配備します。 「Mobile Link ファイル転送」『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。

Ultra Light データベースと Ultra Light Java Edition データベースの相違の詳細については、「Ultra Light データベースと Ultra Light Java Edition データベース」 5 ページ を参照してください。

データベースストアを設定するには、Configuration オブジェクトを使用します。Configuration オブジェクトは、いくつかの方法で実装できます。Ultra Light J API でサポートされているデータベースストアのタイプごとに固有の実装があります。各実装には、データベースストアの設定に使用するメソッドのセットがあります。

次の表に、サポート対象データベースストアで利用できる Configuration オブジェクトの実装を示します。

ストアの型	Ultra Light J API のサポート
Android ファイルシステム	ConfigFileAndroid インターフェイス [Android] [Ultra Light J] を参照してください。
RIM オブジェクト (BlackBerry)	ConfigObjectStore インターフェイス [BlackBerry] [Ultra Light J] を参照してください。
レコード	ConfigRecordStore インターフェイス (J2ME のみ) [UltraLiteJ] を参照してください。
Java SE ファイルシステム	ConfigFile インターフェイス [Ultra Light J] を参照してください。

ストアの型	Ultra Light J API のサポート
非永続型 (メモリ内)	ConfigNonPersistent インターフェイス [BlackBerry] [Ultra Light J] を参照してください。
内部フラッシュと SD カード	ConfigFileME インターフェイス [BlackBerry] [Ultra Light J] を参照してください。

永続的なデータベースストアを作成する場合、Configuration オブジェクトを使用して、Java アプリケーション用の永続性の型を設定できます。デフォルトでは、シャドウページング型の永続性が有効です。永続的な Configuration オブジェクトの setShadowPaging メソッドを false に設定することによって、代替としてシンプルページング型の永続性を使用できます。シンプルページング型の永続性を有効にした場合は、永続的な Configuration オブジェクトの setWriteAtEnd メソッドを true に設定して、終了時書き込み型の永続性を設定できます。

注意
終了時書き込み型とシンプルページング型の永続性は、データ損失が許容される場合、データベースを簡単に再作成できる場合、またはデータベースが更新されない場合にのみアプリケーションで使用してください。

Configuration オブジェクトを使用した永続ストアと非永続ストアの設定の詳細については、以下を参照してください。

- ConfigNonPersistent インターフェイス [BlackBerry] [Ultra Light J]84 ページ
- ConfigPersistent.setShadowPaging メソッド [Ultra Light J]98 ページ
- ConfigPersistent.setWriteAtEnd メソッド [Ultra Light J]99 ページ

Configuration オブジェクトを作成して設定した後、Connection オブジェクトを使用してデータベースを作成するか、データベースに接続します。また、Connection オブジェクトを使用して、次の操作を実行できます。

- **トランザクション**　トランザクションは、コミットまたはロールバックの間の一連のオペレーションです。永続的なデータベースストアの場合、コミット操作は、最後のコミットまたはロールバックの実行後に行われたすべての変更を永続的にします。ロールバック操作は、最後にコミットが呼び出されたときの状態にデータベースを戻します。

  Ultra Light のトランザクションとローレベルの操作は、それぞれアトミックです。複数のカラムを対象とした挿入操作では、すべてのカラムにデータが挿入されるか、どのカラムにもデータが挿入されないかのいずれかです。

  トランザクションは、Connection オブジェクトの commit メソッドを使用してデータベースにコミットする必要があります。

- **SQL 準備文**　SQL 文を処理するメソッドが PreparedStatement インターフェイスに用意されています。PreparedStatement は、Connection オブジェクトの prepareStatement メソッドを使用して作成できます。

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

Java アプリケーションからのデータベースの作成または接続

前提条件

Ultra Light J API を実装する Android デバイスまたは BlackBerry スマートフォン用の既存の Java アプリケーション

内容と備考

次のとおりです。

♦ Java アプリケーションからのデータベースの作成または接続

1. データベース名を参照し、プラットフォームに適した新しい Configuration を作成します。

   次の例では、**config** が Configuration オブジェクトの名前で、*DBname* がデータベースの名前です。

   Android スマートフォンの場合：

   ```
   ConfigFileAndroid config =
     DatabaseManager.createConfigurationFileAndroid("DBname.udb");
   ```

   BlackBerry スマートフォンの場合：

   ```
   ConfigObjectStore config =
     DatabaseManager.createConfigurationObjectStore("DBname.ulj");

   ConfigFileME config =
   DatabaseManager.createConfigFileME( "file:///store/home/user/DBname.ulj" );

   ConfigFileME config =
   DatabaseManager.createConfigFileME( "file:///SDCard/DBname.ulj" );
   ```

   その他の Java ME デバイスの場合：

   ```
   ConfigRecordStore config =
     DatabaseManager.createConfigurationRecordStore("DBname.ulj");
   ```

   Java SE プラットフォームの場合：

   ```
   ConfigFile config =
     DatabaseManager.createConfigurationFile("DBname.ulj");
   ```

   任意のプラットフォームの場合、非永続的なデータベースの Configuration オブジェクトを作成できます。

   ```
   ConfigNonPersistent config =
     DatabaseManager.createConfigurationNonPersistent("DBname.ulj");
   ```

2. Configuration オブジェクトのメソッドを使用して、データベースプロパティを設定します。

   たとえば、setPassword メソッドを使用して新しいデータベースパスワードを設定できます。

```
config.setPassword("my_password");
```

BlackBerry スマートフォンの場合は、Configuration オブジェクトを使用してデータベースの永続性を設定することもできます。次の例は、データベースを作成する前に終了時書き込み型の永続性を設定する方法を示しています。

```
config.setShadowPaging(false);
config.setWriteAtEnd(true);
```

Android スマートフォンの場合は、setCreationString メソッドと setConnectionString メソッドを使用して、それぞれ追加の作成パラメーターと接続パラメーターを設定できます。

作成パラメーターと接続パラメーターの詳細については、以下を参照してください。

- 「Ultra Light 作成パラメーター」『Ultra Light データベース管理とリファレンス』
- ConfigPersistent.setCreationString メソッド [Android] [Ultra Light J]95 ページ
- 「Ultra Light 接続パラメーター」『Ultra Light データベース管理とリファレンス』
- ConfigPersistent.setConnectionString メソッド [Android] [Ultra Light J]94 ページ

3. データベースを作成する、またはデータベースに接続するために、Connection オブジェクトを作成します。

   新しいデータベースを作成するには、次のコードを使用して、Connection オブジェクトを作成します。

   ```
   Connection conn = DatabaseManager.createDatabase(config);
   ```

   DatabaseManager.createDatabase メソッドによりデータベースが作成され、そのデータベースへの接続が返されます。

   既存のデータベースに接続するには、次のコードを使用して、Connection オブジェクトを作成します。

   ```
   Connection conn = DatabaseManager.connect(config);
   ```

   connect メソッドはデータベース接続処理を完了します。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。

結果

Java アプリケーションから SQL 文を実行すると、データベースにテーブルとインデックスを作成できますが、データベース名、パスワード、ページサイズなどの特定のデータベース作成パラメーターを変更することはできません。

次の手順

なし。

参照

- 「Java SE のサンプル: データベースの作成」28 ページ
- Connection インターフェイス [Ultra Light J]105 ページ
- DatabaseManager クラス [Ultra Light J]130 ページ

SQL 文を使用したデータの作成と修正

一部の SQL 文は、Ultra Light ではサポートされますが、Ultra Light Java Edition ではサポートされません。

参照

- 「Ultra Light SQL 文」『Ultra Light データベース管理とリファレンス』

Database オブジェクト

Database オブジェクトを作成および更新するには、SQL 文とクエリを使用します。これらの SQL 文を使用して、データベースのテーブル、インデックス、外部キー、パブリケーションを作成、更新、または検索できます。スキーマ情報は、Connection.prepareStatement メソッドと PreparedStatement.executeQuery メソッドにより作成できます。

参照

- Connection.prepareStatement メソッド [Ultra Light J]117 ページ
- PreparedStatement.executeQuery メソッド [Ultra Light J]180 ページ

例

次の例は、SQL 文を使用してテーブルを作成する方法を示しています。

```
static String stmt_1 = "CREATE TABLE Department("
   + "id INT PRIMARY KEY, "
   + "name CHAR(50) NOT NULL)";

static String stmt_2 = "CREATE TABLE Employee("
   + "id INT PRIMARY KEY, "
   + "last_name CHAR(50) NOT NULL, "
   + "first_name CHAR(50) NOT NULL, "
   + "dept_id INT NOT NULL, "
   + "NOT NULL FOREIGN KEY(dept_id) "
   + "REFERENCES Department(id))";

static String stmt_3 = "CREATE INDEX ON Employee(last_name, first_name)";

void createDb(Connection connection) throws ULjException {
   PreparedStatement ps;
   ps = connection.prepareStatement(stmt_1);
   ps.execute();
   ps.close();
   ps = connection.prepareStatement(stmt_2);
   ps.execute();
   ps.close();
   ps = connection.prepareStatement(stmt_3);
   ps.execute();
   ps.close();
}
```

この例では、createDB メソッドで SQL 文の文字列を使用し、2 つのテーブルおよび **last_name** と **first_name** での追加インデックスを作成する文を準備し、実行しています。

INSERT、UPDATE、DELETE を使用したデータ修正

PreparedStatement の execute メソッドを使用して SQL データ修正を実行できます。PreparedStatement は、データベースに対してユーザー定義の SQL 文を実行します。

PreparedStatement に SQL 文を適用するときは、**?** 文字でクエリパラメーターを指定します。INSERT 文、UPDATE 文、DELETE 文では、文での順序位置に従ってそれぞれの **?** パラメーターが参照されます。たとえば、最初の **?** はパラメーター 1、2 番目の **?** はパラメーター 2、のようになります。

♦ テーブルへのローの挿入 (INSERT)

1. 新しい SQL 文を String として準備します。

   ```
   String sql_string =
      "INSERT INTO Department(dept_no, name) VALUES( ?, ? )";
   ```

2. String を PreparedStatement に渡します。

   ```
   PreparedStatement inserter =
      conn.prepareStatement(sql_string);
   ```

3. set メソッドを使用して、入力値を PreparedStatement に渡します。

   この例では、パラメーター 1 として参照している dept_no に 101 を設定し、パラメーター 2 として参照している name に "Electronics" を設定しています。

   ```
   inserter.set(1, 101);
   inserter.set(2, "Electronics");
   ```

4. 文を実行します。

   ```
   inserter.execute();
   ```

5. PreparedStatement を閉じてリソースを解放します。

   ```
   inserter.close();
   ```

6. データベースへのすべての変更をコミットします。

   ```
   conn.commit();
   ```

手順 (3) と (4) は、必要に応じて何回でも繰り返すことができます。

あるいは、次の例に示すように、単一エントリだけを挿入する必要がある場合は、1 つの INSERT 文を準備し、実行して、終了できます。

```
INSERT INTO Department(dept_no, name) VALUES(2, 'Electronics');
```

文は、Java String を連結させて作成することもできます。

♦ テーブル内のローの更新 (UPDATE)

1. 新しい SQL 文を String として準備します。

```
String sql_string =
   "UPDATE Department SET dept_no = ? WHERE dept_no = ?";
```

2. String を PreparedStatement に渡します。

```
PreparedStatement updater =
   conn.prepareStatement(sql_string);
```

3. set メソッドを使用して、入力値を PreparedStatement に渡します。

```
updater.set(1, 102);
updater.set(2, 101);
```

上記の例は、次の SQL 文の実行と同じです。

```
UPDATE Department SET dept_no = 102 WHERE dept_no = 101;
```

4. 文を実行します。

```
updater.execute();
```

5. PreparedStatement を閉じてリソースを解放します。

```
updater.close();
```

6. データベースへのすべての変更をコミットします。

```
conn.commit();
```

手順 (3) と (4) は、必要に応じて何回でも繰り返すことができます。

あるいは、単一の更新だけを実行する場合は、次の文を準備し、実行して、終了できます。

```
UPDATE Department SET dept_no = 102 WHERE dept_no = 101;
```

この文は、Java String を連結させて作成することもできます。

♦ テーブル内のローの削除 (DELETE)

1. 新しい SQL 文を String として準備します。

```
String sql_string =
   "DELETE FROM Department WHERE dept_no = ?";
```

2. String を PreparedStatement に渡します。

```
PreparedStatement deleter =
   conn.prepareStatement(sql_string);
```

3. set メソッドを使用して、入力値を PreparedStatement に渡します。

```
deleter.set(1, 102);
```

上記の例は、次の SQL 文の実行と同じです。

```
DELETE FROM Department WHERE dept_no = 102;
```

4. 文を実行します。


```
deleter.execute();
```

5. PreparedStatement を閉じてリソースを解放します。

   ```
   deleter.close();
   ```

6. データベースへのすべての変更をコミットします。

   ```
   conn.commit();
   ```

7. 手順 (3) と (4) は、必要に応じて何回でも繰り返すことができます。

   あるいは、前述したように、単一の削除だけを実行する場合は、次の文を準備し、実行して、終了できます。

   ```
   DELETE FROM Department WHERE dept_no = 102;
   ```

   この文は、Java String を連結させて作成することもできます。

例

準備された文は、何回も実行できます。次の例は、ホスト変数を使用して複数のローを挿入する方法を示しています。

```
String stmt = "INSERT INTO Department(dept_no, name) VALUES (?,?)";
PreparedStatement inserter = conn.prepareStatement(stmt);

inserter.set(1, 101);
inserter.set(2, "Electronics");
inserter.execute();

inserter.set(1, 105);
inserter.set(2, "Sales");
inserter.execute();

inserter.set(1, 109);
inserter.set(2, "Accounting");
inserter.execute();

inserter.close();
```

1 回しか実行する必要のない SQL 文を作成する場合は、ホスト変数を連結することをおすすめします。次の例は、連結を使用して SQL 文を実行するローの挿入メソッドを示しています。

```
void addRow(int id, String name, Connection conn) throws ULjException {
   PreparedStatement ps = conn.prepareStatement(
      "INSERT INTO Department(id, name) VALUES("
      + id
      + ", '"
      + name
      + "')";
   try {
      ps.execute();
   }
   finally {
      ps.close(); ¥¥ close the PreparedStatement even if an error occurs after execution.
   }
}
```

SELECT を使用したデータの検索

SELECT 文を使用すると、データベースから情報を取り出すことができます。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。

♦ データベースからのデータの検索 (SELECT)

1. 新しい SQL 文を String として準備します。

```
String sql_string =
   "SELECT * FROM Department ORDER BY dept_no";
```

2. String を PreparedStatement に渡します。

```
PreparedStatement select_statement =
   conn.prepareStatement(sql_string);
```

3. 文を実行し、クエリの結果を ResultSet に割り当てます。

```
ResultSet cursor =
   select_statement.executeQuery();
```

4. ResultSet をトラバースしてデータを取り出します。

```
// Get the next row stored in the ResultSet.
cursor.next();

// Store the data from the first column in the table.
int dept_no = cursor.getInt(1);

// Store the data from the second column in the table.
String dept_name = cursor.getString(2);
```

5. ResultSet を閉じてリソースを解放します。

```
cursor.close();
```

6. PreparedStatement を閉じてリソースを解放します。

```
select_statement.close();
```

参照

- Connection.prepareStatement メソッド [Ultra Light J]117 ページ
- PreparedStatement.executeQuery メソッド [Ultra Light J]180 ページ
- ResultSet インターフェイス [Ultra Light J]194 ページ

SQL 結果セットのナビゲーション

スキーマ情報は、Connection.prepareStatement メソッドと PreparedStatement.executeQuery メソッドにより取得できます。これらのメソッドでは、ユーザー定義の SQL 文を使用して Ultra Light データベースに問い合わせ、結果セットを ResultSet オブジェクトに格納します。

ResultSet オブジェクトは、ResultSet インターフェイスのナビゲーションメソッドを使用して検索することができます。このインターフェイスには、次のナビゲーションメソッドが含まれています。

- **afterLast**[1] カーソルを最後のローの直後に配置します。
- **beforeFirst**[1] 最初のローの直前に配置します。
- **first**[1] 最初のローに移動します。
- **last**[1] 最後のローに移動します。
- **next** 次のローに移動します。
- **previous** 前のローに移動します。
- **relative(offset)**[1] 現在のローを基準にして、符号付きオフセット値で指定された数だけローを移動します。 オフセット値を正の値で指定すると、現在の結果セットのポインター位置から前方に移動します。 負の値で指定すると後方に移動します。オフセット値が 0 の場合、カーソルは現在のロケーションから移動しませんが、ローバッファーが再配置されます。

[1] このメソッドは、Ultra Light Java Edition データベースではサポートされていません。

参照

- Connection.prepareStatement メソッド [Ultra Light J]117 ページ
- PreparedStatement.executeQuery メソッド [Ultra Light J]180 ページ
- ResultSet インターフェイス [Ultra Light J]194 ページ
- ResultSet.afterLast メソッド [Android] [Ultra Light J]196 ページ
- ResultSet.beforeFirst メソッド [Android] [Ultra Light J]196 ページ
- ResultSet.first メソッド [Android] [Ultra Light J]197 ページ
- ResultSet.last メソッド [Android] [Ultra Light J]211 ページ
- ResultSet.next メソッド [Ultra Light J]211 ページ
- ResultSet.previous メソッド [Ultra Light J]212 ページ
- ResultSet.relative メソッド [Android] [Ultra Light J]212 ページ

検索の例

次の例は、ユーザー定義の SQL 文を照会し、next メソッドを使用して結果セットからローを検索し、指定したカラムのデータにアクセスする方法を示しています。

```
// Define a new SQL SELECT statement.
String sql_string = "SELECT column1, column2 FROM SampleTable";

// Create a new PreparedStatement from an existing connection.
PreparedStatement ps = conn.prepareStatement(sql_string);

// Create a new ResultSet to contain the query results of the SQL statement.
ResultSet rs = ps.executeQuery();

// Check if the PreparedStatement contains a ResultSet.
if (ps.hasResultSet()) {
    // Retrieve the column1 value from the first row using getString.
    String row1_col1 = rs.getString(1);
```

```
        // Get the next row in the table.
        if (rs.next()) {
            // Retrieve the value of column1 from the second row.
            String row2_col1 = rs.getString(1);
        }
    }

    rs.close();
    ps.close();
```

ナビゲーションの例

次の例は、ResultSet.next メソッドを使用して、結果セットから整数リストを構築する方法を示しています。

```
PreparedStatement ps = conn.prepareStatement("SELECT id FROM t");
ResultSet rs = ps.executeQuery();
List<Integer> l = new ArrayList<Integer> ();

while(rs.next()) {
   l.add(rs.getInt(1));
}
rs.close();
ps.close();
```

トランザクション管理

Ultra Lite J API は、オートコミットモードをサポートしません。トランザクションは、Connection インターフェイスでサポートされているメソッドを使用して明示的にコミットまたはロールバックする必要があります。

トランザクションをコミットするには、commit メソッドを使用します。トランザクションをロールバックするには、rollback メソッドを使用します。

参照

- Connection.commit メソッド [Ultra Light J]109 ページ
- Connection.rollback メソッド [Ultra Light J]119 ページ

スキーマ情報へのアクセス

データベーススキーマの記述は、プログラムを使用して取得できます。これらの記述は、「スキーマ情報」と呼ばれ、システムテーブルおよび Ultra Light J API スキーマインターフェイスを使用してアクセスできます。

システムテーブルを使用したスキーマ情報へのアクセス

データベースのスキーマ情報は、Ultra Light または Ultra Light Java Edition のシステムテーブルに格納されます。この情報にアクセスするには、正規 SQL クエリを実行して適切なシステムテーブルから所定の情報を選択し、結果セットにアクセスします。

スキーマインターフェイスを使用したスキーマ情報へのアクセス

一部のスキーマ情報は、システムテーブルの代わりにスキーマインターフェイスを使用してアクセスできます。UltraLiteJ API には、次のスキーマインターフェイスが含まれています。

- **TableSchema**　カラムとインデックスの設定に関する情報を返します。
- **IndexSchema**　インデックス内のカラムに関する情報を返します。IndexSchema オブジェクトは、TableSchema オブジェクトから取得できます。
- **ColumnSchema**　テーブル内のカラムに関する情報を返します。ColumnSchema オブジェクトは、TableSchema オブジェクトから取得できます。

参照

- 「SQL 結果セットのナビゲーション」14 ページ
- 「Ultra Light のシステムテーブル」『Ultra Light データベース管理とリファレンス』
- 「Ultra Light Java Edition システムテーブル」『Ultra Light データベース管理とリファレンス』
- 「SELECT を使用したデータの検索」14 ページ
- TableSchema インターフェイス [Ultra Light J]264 ページ
- IndexSchema インターフェイス [Ultra Light J]175 ページ
- ColumnSchema インターフェイス [Ultra Light J]71 ページ

エラー処理

ULjException クラスおよび SQLCode クラスを使用して、エラーを処理できます。ほとんどの Ultra Light メソッドは、ULException エラーをスローします。ULjException.getErrorCode メソッドを使用して、エラーに割り当てられた SQLCode 値を取得できます。ULjException.toString メソッドを使用して、エラーの説明文を取得できます。SQLCode エラーは、エラー型を示す負数で、ULjException.SQLE_INDEX_NOT_FOUND などの定数を使用して参照できます。

例

次の例は、Ultra Light Java Edition データベースへの接続時に発生する可能性があるエラーを、ULjException クラスを使用して処理する Java クラスを示しています。

```
import com.ianywhere.ultralitej12.*;
import net.rim.device.api.ui.component.*;
import java.util.*;

class DataAccess {
    DataAccess() {
    }

    public static synchronized DataAccess getDataAccess(boolean reset)
        throws Exception
    {
        if (_da == null) {
            _da = new DataAccess();
            ConfigObjectStore config = DatabaseManager.createConfigurationObjectStore("HelloDB");
            if (reset) {
                _conn = DatabaseManager.createDatabase(config);
                // _da.createDatabaseSchema();
```

```
            }
            else {
                try {
                    _conn = DatabaseManager.connect(config);
                }
                catch (ULjException uex1) {
                    if (uex1.getErrorCode() != ULjException.SQLE_ULTRALITE_DATABASE_NOT_FOUND) {
                        Dialog.alert("Exception: " + uex1.toString() + ". Recreating database...");
                    }
                    _conn = DatabaseManager.createDatabase(config);
                    // _da.createDatabaseSchema();
                }
            }
        }
        return _da;
    }
    private static Connection _conn;
    private static DataAccess _da;
}
```

参照

- ULjException クラス [Ultra Light J]269 ページ
- ULjException.getErrorCode メソッド [Ultra Light J]270 ページ
- 「SQL Anywhere のエラーメッセージ (SQLCODE 順)」『エラーメッセージ』

Ultra Light Java Edition データベースでのデータの暗号化

暗号化ではデータを安全に表現できますが、難読化ではデータベースの内容を不用意に閲覧されないことを目的とした簡易的なセキュリティを実現します。Ultra Light Java Edition データベースに格納されるデータは、デフォルトでは暗号化されません。

Ultra Light Java Edition データベースでデータを暗号化するには、独自の暗号制御を提供する必要があります。暗号制御は ConfigPersistent.setEncryption メソッドを使用して設定します。このメソッドにより、ページを暗号化および複合化する EncryptionControl オブジェクトが受け入れられます。

注意
暗号化は、非永続データベースストアには使用できません。

BlackBerry アプリケーションでの、EncryptionControl インターフェイスを使用した Ultra Light Java Edition データベースの暗号化または難読化 API 技術の作成とカスタマイズ

前提条件

Ultra Light J API を実装する BlackBerry スマートフォン用の既存の Java アプリケーション

内容と備考

次のとおりです。

♦ Ultra Light Java Edition データベースでのデータの暗号化または難読化

1. EncryptionControl インターフェイスを実装するクラスを作成します。

   次の例では、暗号化インターフェイスを実装する Encryptor という名前の新しいクラスを作成しています。

   ```
   static class Encryptor
      implements EncryptionControl
   {
   ```

2. 新しいクラスで initialize、encrypt、decrypt の各メソッドを実装します。

   クラスは次のようになります。

   ```
   static class Encryptor
      implements EncryptionControl
   {
      /** Decrypt a page stored in the database.
       * @param page_no the number of the page being decrypted
       * @param src the encrypted source page which was read from the database
       * @param tgt the decrypted page (filled in by method)
       */
      public void decrypt( int page_no, byte[] src, byte[] tgt )
         throws ULjException
      {
         // Your decryption method goes here.
      }

      /** Encrypt a page stored in the database.
       * @param page_no the number of the page being encrypted
       * @param src the unencrypted source
       * @param tgt the encrypted target page which will be written to the database (filled in by method)
       */
      public void encrypt( int page_no, byte[] src, byte[] tgt )
         throws ULjException
      {
         // Your encryption method goes here.
      }

      /** Initialize the encryption control with a password.
       * @param password the password
       */
      public void initialize(String password)
         throws ULjException
      {
         // Your initialization method goes here.
      }
   }
   ```

3. データベースに接続する前に、新しい暗号化制御クラスが指定されるように、API コードを更新します。

   Configuration.setEncryption メソッドで暗号化制御を指定します。次の例は、データベースを参照する config という Configuration オブジェクトが作成されていることを前提として、暗号化制御を設定する方法を示しています。

   ```
   config.setEncryption(new Encryptor());
   ```

結果

データベースで追加または修正されるデータはすべて、初期化、暗号化、複合化の各メソッドの実装方法に応じて、暗号化または難読化されます。

次の手順

なし。

「『BlackBerry デバイス上の Ultra Light J セキュリティ』」ホワイトペーパーには、セキュリティオプションに関する情報や、BlackBerry デバイスでの Ultra Light J アプリケーションに関する考慮事項 (デバイスの組み込みセキュリティをアプリケーションで操作する場合の利点と短所など) が記載されています。詳細については、UltraLiteJ Security on BlackBerry Devices を参照してください。

参照

- 「Ultra Light データベースのセキュリティ」『Ultra Light データベース管理とリファレンス』
- EncryptionControl インターフェイス [Ultra Light J]153 ページ
- 「Ultra Light データベースのキャッシュサイズの調整」『Ultra Light データベース管理とリファレンス』
- 「Java SE のサンプル: データの難読化」32 ページ
- 「Java SE のサンプル: データの暗号化」32 ページ
- EncryptionControl インターフェイス [Ultra Light J]153 ページ

Mobile Link データ同期

Ultra Light Java Edition 管理システムには、データベースの変更が同期されるように変更トラッキングシステムが組み込まれています。

データの同期は、HTTP ネットワークプロトコルまたは HTTPS ネットワークプロトコルを使用して実行できます。HTTPS 同期を使用すると、Mobile Link サーバーに安全な暗号化が提供されます。

データを同期するには、アプリケーションで次の手順を実行する必要があります。

1. 統合データベースに関する情報 (サーバー名、ポート番号)、同期するベータベース名、同期するテーブルの定義を含む syncParms オブジェクトをインスタンス化します。

2. syncParms オブジェクトを渡した接続オブジェクトから synchronize メソッドを呼び出して、同期を実行します。

同期するデータはテーブルレベルで定義できます。テーブルの一部分を同期するように設定することはできません。

参照

- SyncParms クラス [Ultra Light J]241 ページ
- SyncResult クラス [Ultra Light J]258 ページ
- 「チュートリアル：Android アプリケーションの構築」37 ページ

例

次の例は、Ultra Light J アプリケーションを使用したデータの同期を示しています。この例は、*%SQLANYSAMP12%¥UltraLiteJ¥J2SE¥Sync.java* にあります。

```
package com.ianywhere.ultralitej.demo;
import com.ianywhere.ultralitej12.*;
/**
 * Sync: sample program to demonstrate Database synchronization.
 *
 * Requires starting the MobiLink Server Sample using start_ml.bat
 */
public class Sync
{
    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config = DatabaseManager.createConfigurationFile( "Demo1.ulj" );
            Connection conn = DatabaseManager.createDatabase( config );

            PreparedStatement ps = conn.prepareStatement(
                "CREATE TABLE ULCustomer" +
                "( cust_id int NOT NULL PRIMARY KEY" +
                ", cust_name VARCHAR(30) NOT NULL" +
                ")"
            );
            ps.execute();
            ps.close();

            //
            // Synchronization
            //

            SyncParms syncParms = conn.createSyncParms( SyncParms.HTTP_STREAM, "50", "custdb
12.0" );
            syncParms.getStreamParms().setPort( 9393 );
            conn.synchronize( syncParms );
            SyncResult result = syncParms.getSyncResult();
            Demo.display(
                  "*** Synchronized *** bytes sent=" + result.getSentByteCount()
                  + ", bytes received=" + result.getReceivedByteCount()
                  + ", rows received=" + result.getReceivedRowCount()
                );

            conn.release();

        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
```

```
    }
  }
```

CustDB を統合データベースとして Mobile Link サーバーを起動するには、*%SQLANYSAMP12%¥J2SE¥UltraLiteJ* ディレクトリから *start_ml.bat* を実行します。

Android スマートフォンでは、統合データベースとして CustDB を使用したチュートリアルを利用できます。

BlackBerry スマートフォンでのデータ同期

BlackBerry 環境では、データはデバイスと BlackBerry Enterprise Server (BES) 間で常に暗号化されます。HTTPS は、BES と Mobile Link サーバー間で暗号化が必要な場合、またはデバイスが BES で管理されていないときに役立ちます。

同時同期処理

通常、Ultra Light ランタイムでは、一度に 1 つのスレッドだけが許可されます。しかし、同期中はこの規則に例外が発生します。1 つの接続で同期操作を実行しているときに、他の接続で Ultra Light ランタイムにアクセスできます。ただし、同期操作の実行中は、同期中のデータベースに対して他のスレッドから同期メソッドを呼び出すことはできません。

Ultra Light は独立性レベル 0 で動作するので、接続により、実行中の同期が失敗した場合に消去される可能性のあるダウンロードされたローに、同期中 (ローがコミットされる前) にアクセスできます。同期によって後から変更されるローが、同期中に接続によって修正されると、同期は失敗します。同期によって変更されたローを、同期中に接続によって修正しようとすると、その試行は失敗します。

Ultra Light J 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 サーバーと同期するときは、アプリケーション内でネットワークプロトコルを設定する必要があります。各データベースはネットワークプロトコルを通じて同期されます。Ultra Light J では、HTTP と HTTPS の 2 つのネットワークプロトコルを使用できます。

設定したネットワークプロトコルでは、対応するプロトコルオプションのセットから選択することによって、Ultra Light J アプリケーションが Mobile Link サーバーを特定して通信できるようにします。ネットワークプロトコルのオプションは、アドレス情報 (ホストとポート) やプロトコル固有の情報などを提供します。

HTTP ネットワークプロトコルの設定

HTTP ネットワークプロトコルは、Ultra Light J API の StreamHTTPParms インターフェイスを使用して設定します。インターフェイスのメソッドを使用して、Mobile Link サーバーで定義されているネットワークプロトコルのオプションを指定します。

HTTPS ネットワークプロトコルの設定

HTTPS ネットワークプロトコルは、Ultra Light J API の StreamHTTPSParms インターフェイスを使用して設定します。インターフェイスのメソッドを使用して、Mobile Link サーバーで定義されているネットワークプロトコルのオプションを指定します。

参照

- 「Mobile Link クライアントネットワークプロトコルオプション」『Mobile Link クライアント管理』
- StreamHTTPParms インターフェイス [Ultra Light J]220 ページ
- StreamHTTPSParms インターフェイス [Ultra Light J]229 ページ

BlackBerry スマートフォンでの CustDB アプリケーションの同期

CustDB (顧客データベース) は、SQL Anywhere と同時にインストールされるサンプルデータベースであり Ultra Light J アプリケーションです。CustDB データベースは、販売注文用の簡単なデータベースです。

アプリケーションの検索と配備

Ultra Light J には、CustDB データベースに基づいたサンプルの BlackBerry アプリケーションが含まれています。このアプリケーションの名前は CustDB で、ソースコードと関連ファイルは、*%SQLANYSAMP12%¥UltraLiteJ¥BlackBerry¥CustDB¥* フォルダーにあります。CustDB ディレクトリには、BlackBerry JDE を使用して開くことのできるプロジェクトファイルが含まれています。CustDB アプリケーションに関するアプリケーション情報については、*readme.txt* を参照してください。

CustDB アプリケーションの構築後、*CustDB12.cod* と必要なファイルをシミュレーターまたはデバイスに配備します。

CustDB アプリケーションの関連ファイル

CustDB プロジェクトの次のファイルには、データベースアクセスコードが含まれています。

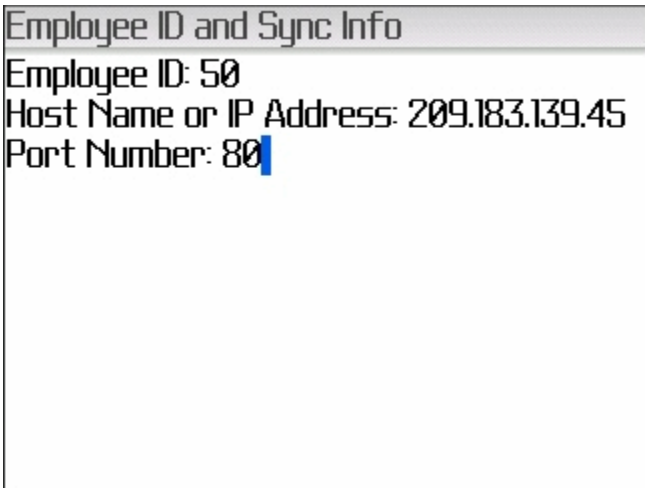
- ***CustDB.java*** このファイルには、基本的なデータベースアクセスメソッドがすべて含まれています。これらのメソッドには、データベースの作成、データベースへの接続、注文の挿入、削除、更新などのメソッドがあります。このファイルには、バックエンドサーバーと通信するデータベース呼び出しの多くが含まれています。

- ***SchemaCreator.java*** このファイルには、Ultra Light J を使用してデバイス上にテーブルを作成するコードが含まれています。

CustDB アプリケーションの使用

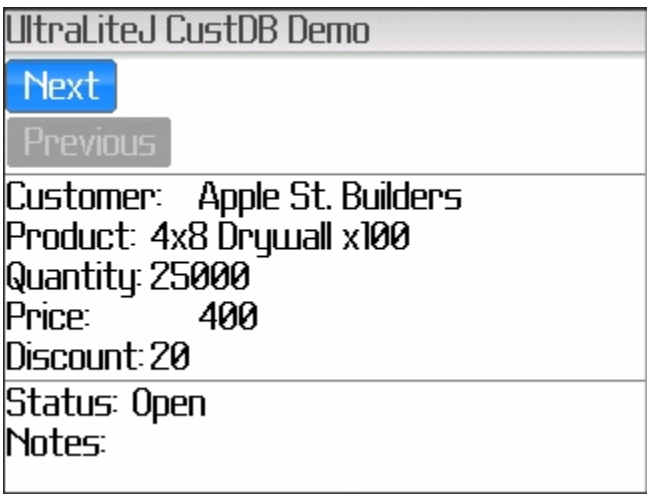
%SQLANYSAMP12%¥UltraLiteJ¥BlackBerry¥CustDB¥mobilink.bat を実行して、ローカルの Mobile Link サーバーを起動します。

CustDB プログラムは、最初の起動時に、CustDB データベースをホストするサーバーと対話するために必要な情報を収集します。ここで、クエリに使用する Employee ID (50 を推奨します)、データベースをホストするサーバーのホスト名または IP アドレス、サーバーとの接続に使用するポート番号を指定します。



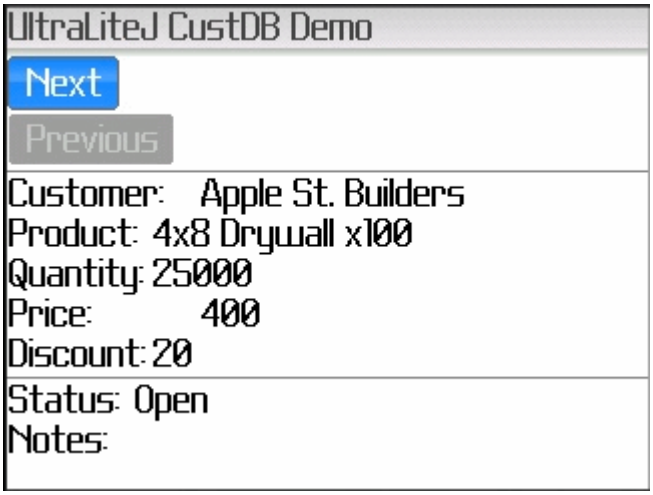
これらの値が指定されて設定が保存されると (**[Menu] » [Save]**)、アプリケーションは指定されたサーバーと同期します。アプリケーションは、指定した従業員番号 (50) に対応する Employee ID と一致する注文だけをサーバーからダウンロードします。オープン状態になっている注文のみが選択されます (注文のステータスは、Open、Approved、Denied のいずれかです)。

それぞれの注文が、顧客名、注文品、数量、価格、割引の情報とともに画面に表示されます。また、画面には注文の現在のステータス (Status) とその注文に関するメモ (Notes) も表示されます。



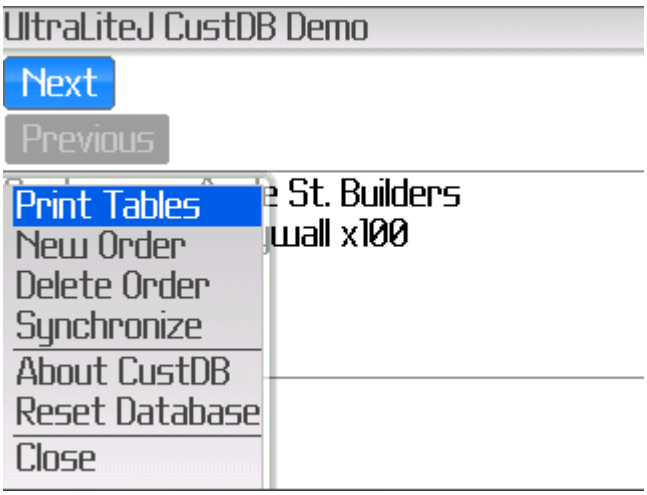
この画面では、注文にメモを書き加えたり、注文のステータスを Approved や Denied に変更したりすることができます。**[Next]** ボタンと **[Previous]** ボタンを使用して注文をナビゲーションできます。

CustDB プログラムでも、データベースに新しい注文を追加できます。新しい注文を追加するには、**[Menu]** » **[New Order]** をクリックします。



必要な注文数と割引情報を入力できます。

アプリケーションを終了する前に、メインメニューで **[Synchronize]** をクリックして、変更内容と新しい注文を統合データベースと同期します。



参照

- 「Ultra Light J アプリケーションの開発」 26 ページ

Ultra Light または Ultra Light Java Edition データベースの終了

すべての同時接続がリリースされると、Ultra Light または Ultra Light Java Edition データベースは終了します。接続はすべて、DatabaseManager.release メソッドを使用してリリースできます。

注意
現在の接続をリリースする場合は、Connection.release メソッドを使用できます。

参照

- DatabaseManager.release メソッド [Ultra Light J]138 ページ
- Connection.release メソッド [Ultra Light J]118 ページ

Ultra Light J アプリケーションの開発

Ultra Light J アプリケーションが正常に実行されるためには、配布ファイルとともに Ultra Light J API も配備する必要があります。次の表に、Ultra Light J の各種開発に必要なファイルを示します。ファイルのパスはすべて SQL Anywhere のインストールディレクトリ内の *UltraLite¥UltraLiteJ* ディレクトリからの相対パスです。

配備のタイプ	必要なファイル
Android スマートフォン	*Android¥UltraLiteJNI12.jar* *Android¥ARM¥libultralitej12.so*
BlackBerry スマートフォン	*BlackBerry4.2¥UltraLiteJ12.cod* *BlackBerry4.2¥UltraLiteJ12.jad*[1]
Java ME	*Java ME11¥UltraLiteJ12.jar*
Java SE	*Java SE¥UltraLiteJ12.jar*

[1] 無線配信 (OTA : over-the-air) での配備にのみ必要です。また、アプリケーションとともに Ultra Light J を配備する独自の *.jad* ファイルを作成することもできます。

参照

- 「Ultra Light アプリケーションのビルドと配備の仕様」『Ultra Light データベース管理とリファレンス』

サンプルコード

この項では、Ultra Light J API を使用する Java コードのサンプルを示します。これらのサンプルでは、デバッグのためにメッセージを表示し、**ULjException** オブジェクトを処理するデモクラスを使用しています。

すべてのサンプルコードは *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリにあります。ファイルの内容を変更する前に、元のソースコードのバックアップコピーを作成することをおすすめします。

これらのサンプルコードは、Windows デスクトップ環境用に作成されていますが、その概念は、特に指定がないかぎり、すべての Ultra Light J プラットフォームに適用されます。

これらのサンプルを実行するには、**JAVA_HOME** 環境変数をインストールされたバージョンの JDK 1.6 に設定します。その後、サンプルの名前を使用して *rundemo.cmd* を実行できます。

%SQLANYSAMP12%¥UltraLiteJ¥J2SE¥Demo.java にあるデモクラスは、この項に記載されているすべてのサンプルで使用されています。

たとえば、次のコマンドは CreateDb サンプルを実行します。

```
rundemo CreateDb
```

このコマンドは次の出力を生成します。

```
Executing:
CREATE TABLE department
( dept_no INT NOT NULL PRIMARY KEY
, name VARCHAR(50)
)

Executing:
CREATE TABLE Employee
( number INT NOT NULL PRIMARY KEY
, last_name VARCHAR(32)
, first_name VARCHAR(32)
, age INT
, dept_no INT
, FOREIGN KEY fk_emp_to_dept( dept_no ) REFERENCES department( dept_no ))

CreateDb completed successfully
```

出力は *demos.out* というファイルに保存されます。

BlackBerry サンプル

次のデモは、BlackBerry 開発者が利用できるものであり、*%SQLANYSAMP12%¥UltraLiteJ¥BlackBerry* ディレクトリにあります。

- BinaryStoreAsFile デモは、外部ファイルをデータベースの一部として関連付ける機能の使用方法を示しています。

- CustDB デモは、モバイル顧客の注文アプリケーションを示しています。

●BlackBerryEncryption デモは、非常に機密性の高い Ultra Light J データベースに関する高度な概念を示しています。

●HelloBlackBerry デモ。

HelloBlackBerry デモに基づいたチュートリアルを利用できます。

Android のサンプル

CustDB サンプルデータベースを使用するサンプル Eclipse プロジェクトが *%SQLANYSAMP12%¥UltraLiteJ¥Android¥CustDB* ディレクトリに用意されています。ソースコードは *%SQLANYSAMP12%¥UltraLiteJ¥Android¥CustDB¥src¥com¥sybase¥custdb* にあります。

例に基づいたチュートリアルを利用できます。

参照

- 「チュートリアル：BlackBerry アプリケーションの構築」43 ページ
- 「チュートリアル：Android アプリケーションの構築」37 ページ

Java SE のサンプル: データベースの作成

このサンプルは、Java SE Java 環境でファイルシステムのデータベースストアを作成する方法を示しています。データベースを作成するには、**Configuration** オブジェクトを使用します。作成すると、**Connection** オブジェクトが返されます。テーブルを作成するには、**schemaUpdateBegin** メソッドを呼び出して基本となるスキーマへの変更を開始し、**schemaUpdateComplete** メソッドでスキーマの変更を完了します。

♦ CreateDb.java サンプルの実行

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。

2. CreateDb サンプルを実行します。

```
rundemo CreateDb
```

説明

- Ultra Light Java Edition データベースには所有者のテーブルには所有者が存在せず、名前のみによって識別されます。
- **Domain** インターフェイスは、テーブルのカラムでサポートされる各種のデータ型を示す定数を定義します。
- プライマリインデックス (**createPrimaryIndex**) は、ユニークであることが保証されます。

Java SE のサンプル: ローの挿入

このサンプルは、Ultra Light Java Edition データベースにローを挿入する方法を示しています。

♦ **LoadDb.java サンプルの実行**

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。
2. CreateDb サンプルを実行します。

   ```
   rundemo CreateDb
   ```

3. 次のコマンドを実行します (コマンドでは大文字と小文字が区別されます)。

   ```
   rundemo LoadDb
   ```

説明

- 挿入されたデータは、**Connection** オブジェクトから commit メソッドが呼び出されたときに初めてデータベースで永続的になります。
- ローが挿入され、まだコミットされていないときは、他の接続からそのローを参照できます。このため、ある接続によって、実際にコミットされていないローデータが取り出される可能性があります。

参照

- 「Java SE のサンプル: データベースの作成」28 ページ

Java SE のサンプル: テーブルの読み込み

この例では、接続から **PreparedStatement** オブジェクトを取得し、**PreparedStatement** オブジェクトから **ResultSet** オブジェクトを取得しています。次のローが取得可能であると、**ResultSet** の next メソッドはそのたびに true を返します。現在のローのカラムの値は、**ResultSet** オブジェクトから取得できます。

♦ **ReadSeq.java サンプルの実行**

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。
2. CreateDb サンプルを実行します。

   ```
   rundemo CreateDb
   ```

3. LoadDb サンプルを実行します。

   ```
   rundemo LoadDb
   ```

4. 次のコマンドを実行します (コマンドでは大文字と小文字が区別されます)。

   ```
   rundemo ReadSeq
   ```

説明

- ResultSet が作成されると、結果セットの最初のローの前に配置されます。コードで next メソッドを呼び出して、テーブル内の最初のローに移動する必要があります。
- Ultra Light J のテーブル名およびカラム名は、大文字と小文字が区別されません。カラムは、カラム名のみ ("age") またはテーブル名を伴ったカラム名 ("Employee.age") を指定することによって参照できます。

参照

- 「Java SE のサンプル: データベースの作成」28 ページ
- 「Java SE のサンプル: ローの挿入」28 ページ

Java SE のサンプル: 内部ジョイン操作

このサンプルは、内部ジョイン操作を実行する方法を示しています。このシナリオでは、各従業員に対応する部署情報があります。ジョイン操作によって、従業員 (Employee) テーブルのデータを部署 (Department) テーブルのそれに対応するデータと関連付けます。関連付けは従業員テーブルの部署番号を基準にして行われ、部署テーブル内の関連情報を特定します。

♦ ReadInnerJoin.java サンプルの実行

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。
2. CreateDb サンプルを実行します。

   ```
   rundemo CreateDb
   ```

3. LoadDb サンプルを実行します。

   ```
   rundemo LoadDb
   ```

4. 次のコマンドを実行します (コマンドでは大文字と小文字が区別されます)。

   ```
   rundemo ReadInnerJoin
   ```

参照

- 「Java SE のサンプル: データベースの作成」28 ページ
- 「Java SE のサンプル: ローの挿入」28 ページ

Java SE のサンプル: 販売データベースの作成

このサンプルでは、販売指向型のデータベースを作成します。

♦ CreateSales.java サンプルの実行

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。
2. 次のコマンドを実行します (コマンドでは大文字と小文字が区別されます)。


```
rundemo CreateSales
```

Java SE のサンプル: 集約とグループ化

このサンプルは、Ultra Light J でサポートされている、結果の集約を示しています。

♦ SalesReport.java サンプルの実行

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。
2. CreateSales サンプルを実行します。

   ```
   rundemo CreateSales
   ```

3. 次のコマンドを実行します (コマンドでは大文字と小文字が区別されます)。

   ```
   rundemo SalesReport
   ```

参照

- 「Java SE のサンプル: 販売データベースの作成」30 ページ

Java SE のサンプル: 別の順序でのローの取り出し

このサンプルは、Ultra Light J でサポートされている、別の順序でのローの処理を示しています。

♦ SortTransactions.java サンプルの実行

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。
2. CreateSales サンプルを実行します。

   ```
   rundemo CreateSales
   ```

3. 次のコマンドを実行します (コマンドでは大文字と小文字が区別されます)。

   ```
   rundemo SortTransactions
   ```

参照

- 「Java SE のサンプル: 販売データベースの作成」30 ページ

Java SE のサンプル: テーブル定義の変更

このサンプルは、テーブル定義の変更方法を示しています。このシナリオでは、カラム長を 50 文字から 100 文字に拡張するように Invoice テーブルを変更しています。

♦ Reorg.java サンプルの実行

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。
2. CreateSales サンプルを実行します。

   ```
   rundemo CreateSales
   ```

3. 次のコマンドを実行します (コマンドでは大文字と小文字が区別されます)。

   ```
   rundemo Reorg
   ```

参照

- 「Java SE のサンプル: 販売データベースの作成」 30 ページ

Java SE のサンプル: データの難読化

このサンプルは、データベースのデータを難読化する方法を示しています。難読化の目的は、データが単に表示されたときに認識されないようにすることにあります。これは暗号化とは異なります。暗号化にはプロパティもあり、高度なツールを使用してもデータを復号化できない可能性が多分にあります。

♦ Obfuscate.java サンプルの実行

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。
2. 次のコマンドを実行します (コマンドでは大文字と小文字が区別されます)。

   ```
   rundemo Obfuscate
   ```

Java SE のサンプル: データの暗号化

このサンプルは、データベースのデータを暗号化する方法を示しています。このシナリオでは、データの復号化が原因でパフォーマンスの低下が生じます。

♦ Encrypted.java サンプルの実行

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。
2. 次のコマンドを実行します (コマンドでは大文字と小文字が区別されます)。

   ```
   rundemo Encrypted
   ```

このサンプルは Java SE 用です。BlackBerry 暗号化の完全なサンプルは、*%SQLANYSAMP12%¥UltraLiteJ¥BlackBerryEncryption* にあります。

Java SE のサンプル: データベースのスキーマ情報の表示

このサンプルは、Ultra Light J データベースのシステムテーブルをナビゲーションしてスキーマ情報を確認する方法を示しています。テーブルの各ローのデータも表示されます。

♦ DumpSchema.java サンプルの実行

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。
2. CreateSales サンプルを実行します。

   ```
   rundemo CreateSales
   ```

3. 次のコマンドを実行します (コマンドでは大文字と小文字が区別されます)。

   ```
   rundemo DumpSchema
   ```

次に、アプリケーションの出力の一部を示します。

```
Metadata options:

Option[ date_format ] = 'YYYY-MM-DD'
Option[ date_order ] = 'YMD'
Option[ global_database_id ] = '0'
Option[ nearest_century ] = '50'
Option[ precision ] = '30'
Option[ scale ] = '6'
Option[ time_format ] = 'HH:NN:SS.SSS'
Option[ timestamp_format ] = 'YYYY-MM-DD HH:NN:SS.SSS'
Option[ timestamp_increment ] = '1'

Metadata tables:

Table[0] name = "systable"  id = 0 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "table_name" flags = 0x0 domain = VARCHAR(128)
  column[2 ]: name = "table_flags" flags = 0x0 domain = UNSIGNED-SHORT
  column[3 ]: name = "table_data" flags = 0x0 domain = INTEGER
  column[4 ]: name = "table_autoinc" flags = 0x0 domain = BIG
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD

Table[1] name = "syscolumn"  id = 1 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "column_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[2 ]: name = "column_name" flags = 0x0 domain = VARCHAR(128)
  column[3 ]: name = "column_flags" flags = 0x0 domain = TINY
  column[4 ]: name = "column_domain" flags = 0x0 domain = TINY
  column[5 ]: name = "column_length" flags = 0x0 domain = INTEGER
  column[6 ]: name = "column_default" flags = 0x0 domain = TINY
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "column_id" flags = 0x1,FORWARD

Table[2] name = "sysindex"  id = 2 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "index_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
```

```
  column[2 ]: name = "index_name" flags = 0x0 domain = VARCHAR(128)
  column[3 ]: name = "index_flags" flags = 0x0 domain = TINY
  column[4 ]: name = "index_data" flags = 0x0 domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "index_id" flags = 0x1,FORWARD

Table[3] name = "sysindexcolumn"  id = 3 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "index_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[2 ]: name = "order" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[3 ]: name = "column_id" flags = 0x0 domain = INTEGER
  column[4 ]: name = "index_column_flags" flags = 0x0 domain = TINY
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "index_id" flags = 0x1,FORWARD
    key[2 ]: name = "order" flags = 0x1,FORWARD

Table[4] name = "sysinternal"  id = 4 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "name" flags = 0x1,IN-PRIMARY-INDEX domain = VARCHAR(128)
  column[1 ]: name = "value" flags = 0x0 domain = VARCHAR(128)
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "name" flags = 0x1,FORWARD

Table[5] name = "syspublications"  id = 5 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "publication_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "publication_name" flags = 0x0 domain = VARCHAR(128)
  column[2 ]: name = "download_timestamp" flags = 0x0 domain = TIMESTAMP
  column[3 ]: name = "last_sync_sent" flags = 0x0 domain = INTEGER
  column[4 ]: name = "last_sync_confirmed" flags = 0x0 domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "publication_id" flags = 0x1,FORWARD

Table[6] name = "sysarticles"  id = 6 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "publication_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "publication_id" flags = 0x1,FORWARD
    key[1 ]: name = "table_id" flags = 0x1,FORWARD

Table[7] name = "sysforeignkey"  id = 7 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "foreign_table_id" flags = 0x0 domain = INTEGER
  column[2 ]: name = "foreign_key_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[3 ]: name = "name" flags = 0x0 domain = VARCHAR(128)
  column[4 ]: name = "index_name" flags = 0x0 domain = VARCHAR(128)
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "foreign_key_id" flags = 0x1,FORWARD

Table[8] name = "sysfkcol"  id = 8 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "foreign_key_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[2 ]: name = "item_no" flags = 0x1,IN-PRIMARY-INDEX domain = SHORT
  column[3 ]: name = "column_id" flags = 0x0 domain = INTEGER
  column[4 ]: name = "foreign_column_id" flags = 0x0 domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
```



```
key[0 ]: name = "table_id" flags = 0x1,FORWARD
key[1 ]: name = "foreign_key_id" flags = 0x1,FORWARD
key[2 ]: name = "item_no" flags = 0x1,FORWARD
```

参照

- 「Java SE のサンプル: 販売データベースの作成」 30 ページ

Java SE のサンプル: データベースの同期

このサンプルは、統合データベース (この場合は SQL Anywhere) を使用してクライアントデータベースを同期するための Mobile Link サーバーへの接続を示しています。

♦ Sync.java サンプルの実行

1. *%SQLANYSAMP12%¥UltraLiteJ¥J2SE* ディレクトリに移動します。
2. CreateSales サンプルを実行します。

   ```
   rundemo CreateSales
   ```

3. 次のコマンドを実行して、Mobile Link サーバーを起動します。

   ```
   start_ml
   ```

4. 次のコマンドを実行します (コマンドでは大文字と小文字が区別されます)。

   ```
   rundemo Sync
   ```

5. サンプルが完了したら、Mobile Link サーバーを停止します。

参照

- 「Java SE のサンプル: 販売データベースの作成」 30 ページ
- 「Mobile Link サーバーの停止」『Mobile Link サーバー管理』

チュートリアル : Android アプリケーションの構築

このチュートリアルでは、Ultra Light J API および Eclipse 環境を使用して、Android スマートフォン用のアプリケーションを開発する方法について説明します。このチュートリアルでは、Windows Simulator 上でアプリケーションを実行します。

このチュートリアルで使用するアプリケーションは、*%SQLANYSAMP12%¥UltraLiteJ¥Android¥CustDB¥* ディレクトリにあります。*src¥com¥sybase¥custdb* ディレクトリにあるアプリケーションコードで、Ultra Light J API で行う次の操作を参照できます。

●Ultra Lite のリモートデータベースの作成。

●データベースでの SQL の操作。

●Mobile Link を使用した SQL Anywhere CustDB サンプルデータベースとのデータの同期。

res¥menu および *res¥layout* ディレクトリは、Android のメニュー項目とインターフェイスの作成方法を示します。新しい Android プロジェクトを作成するときに、Eclipse を介して、これらのファイルを表示できます。

必要なソフトウェア

- Eclipse 3.5.2 以降
- Android SDK Starter Package
- Android Development Tools (ADT) Plug-in for Eclipse 1.1 以降
- SQL Anywhere 12 サンプル
- Ultra Light J API

前提知識と経験

- Java の知識
- Eclipse の知識

参照

- http://developer.android.com/sdk/installing.html#Installing
- http://developer.android.com/sdk/eclipse-adt.html#installing
- http://developer.android.com/sdk/adding-components.html
- 「Ultra Light J API リファレンス」71 ページ

レッスン 1 : 新しい Android プロジェクトの設定

このレッスンでは、Eclipse 統合開発環境を介して、新しい Android プロジェクトを作成します。

♦ Eclipse での新しい Android プロジェクトの設定

1. Android ライブラリを、Android CustDB サンプルディレクトリにコピーします。

コマンドプロンプトを開いて、*%SQLANYSAMP12%¥UltraLiteJ¥Android¥CustDB¥* ディレクトリに移動し、次のコマンドを実行します。

```
setup.bat
```

UltraLiteJNI12.jar および *libultralitej12.so* ファイルは、*Android¥CustDB¥libs* ディレクトリおよび *Android¥CustDB¥libs¥armeabi* ディレクトリにコピーされます。

2. Eclipse を実行します。

   デフォルトのアプリケーションパスは *C:¥Eclipse¥eclipse.exe* です。

3. **[Workspace]** フィールドで、CustDB サンプルディレクトリ以外の作業ディレクトリを指定して、**[OK]** をクリックします。

4. Eclipse への CustDB プロジェクトのインポート
   a. **[File]** » **[Import]** をクリックします。
   b. **[General]** ディレクトリを展開し、**[Existing Projects into Workspace]** を選択します。**[Next]** をクリックします。
   c. **[Select Root Directory]** フィールドに、**%SQLANYSAMP12%¥UltraLiteJ¥Android¥CustDB** と入力します。**[Copy Projects Into Workspace]** を選択して、**[Finish]** をクリックします。

5. Eclipse で適切な Android SDK パスが指定されていることを確認します。

   注意
   パスを指定する前に Android SDK をインストールする必要があります。 チュートリアルの前提条件の詳細については、「チュートリアル：Android アプリケーションの構築」37 ページを参照してください。

   a. **[Window]** » **[Preferences]** をクリックします。
   b. 左ウィンドウ枠で、**[Android]** をクリックします。
   c. **[SDK Location]** フィールドで、Android SDK のロケーションを入力し、**[Apply]** をクリックします。
      利用可能なビルドターゲットのリストが表示されます。
   d. **[OK]** をクリックします。

6. Eclipse で Ultra LightJ ライブラリパスが指定されていることを確認します。
   a. **[File]** » **[Properties]** をクリックします。
   b. 左ウィンドウ枠で、**[Java Build Path]** » **[User libraries]** をクリックします。
   c. **[Libraries]** タブをクリックします。
   d. **[UltraLiteJNI12.jar]** をクリックし、**[Edit]** をクリックします。
   e. 作業ディレクトリから、*¥CustDB¥libs¥UltraLiteJNI12.jar* ファイルを開きます。
   f. **[OK]** をクリックします。

7. プロジェクトに UltraLite JNI Javadoc マニュアルへのパスを追加します。
   a. 左ウィンドウ枠で、**[Javadoc Location]** をクリックします。
   b. **[Browse]** をクリックして、*%SQLANY12%¥UltraLite¥UltraLiteJ¥Android¥html* を開きます。
   c. **[OK]** をクリックします。
   d. **[OK]** をクリックして、ウィンドウを閉じます。
8. プロジェクトの構築

   [Project] » **[Clean]** をクリックし、**[OK]** をクリックします。

   プロジェクトはエラーを発生させずに構築する必要がありますが、**[問題]** タブの下に警告が表示されることがあります。.

レッスン 2：Mobile Link サーバーの起動

このレッスンでは、Mobile Link サーバーを起動します。

♦ Mobile Link サーバーの起動とアプリケーションの同期

- Mobile Link を起動するには、*%SQLANYSAMP12%¥MobiLink¥CustDB¥* から次のコマンドを発行します。

  ```
  mlsrv12 -v+ -zu+ -c "DSN=SQL Anywhere 12 CustDB;UID=ml_server;PWD=sql" -x http(port=80) -ot ml.mls
  ```

  -c オプションは、Mobile Link を SQL Anywhere CustDB データベースに接続します。-v+ オプションは、高い冗長レベルを設定して、処理中の内容を Mobile Link のサーバーメッセージウィンドウで確認できるようにします。-x オプションは、通信に使用されているポート番号を指定します。-ot オプションは、ログファイル (*ml.mls*) が、Mobile Link サーバーを起動したディレクトリに作成されるように指定します。

参照

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

レッスン 3：Android アプリケーションの実行

このレッスンでは、Android Simulator 上でアプリケーションを実行します。

♦ Android Simulator 上でのアプリケーションの実行

1. Eclipse で Android の仮想デバイスを設定します。
   a. **[Window]** » **[AVD Manager]** をクリックします。
   b. **[New]** をクリックします。

      [Create New Android Virtual Device (AVD)] ウィンドウが表示されます。
   c. **[Name]** フィールドに **my_avd** と入力します。

d. **[Target]** フィールドで、**[Android 2.2 - API Level 8]** をクリックします。
e. **[Create AVD]** をクリックします。
f. **[AVD Manager]** ウィンドウを閉じます。

2. **[Package Explorer]** ウィンドウで、**CustDB** を選択します。
3. **[Run]** メニューで、**[Run As]** » **[Android Application]** を選択します。

   Android Simulator がロードされます。
4. **[Menu]** をクリックします。

   Android アプリケーションがロードされます。

レッスン 4：Android アプリケーションのテストと同期

このレッスンでは、Android アプリケーションを使用して、Ultra Light のリモートデータベースを更新し、CustDB 統合データベースと同期します。

♦ Android アプリケーションのテストと同期

1. **[Employee ID]** フィールドが **50**、**[Host]** フィールドが **10.0.2.2**、**[Port]** フィールドが **80** になっていることを確認し、**[Save]** をクリックします。

   アプリケーションは自動的に同期を実行し、一連の顧客、製品、注文情報が CustDB 統合データベースからアプリケーションにダウンロードされます。
2. Simulator で **[メニュー]** » **[新規]** をクリックします。
3. **[Customer]** フィールドで **[Ace Properties]** を選択します。
4. **[Product]** フィールドで **[4x8 Drywall x100]** を選択します。
5. **[Quantity]** フィールドに **[999]** と入力します。
6. **[Discount]** フィールドに **[25]** と入力します。
7. **[OK]** をクリックし、新規注文を追加します。
8. CustDB 統合データベースとアプリケーションを同期します。

   Simulator で **[Menu]** をクリックし、**[Sync]** を選択します。
9. Interactive SQL を CustDB 統合データベースに接続します。
   a. **[スタート]** » **[プログラム]** » **[SQL Anywhere 12]** » **[管理ツール]** » **[Interactive SQL]** をクリックするか、次のコマンドを実行します。

      ```
      dbisql
      ```
   b. **[ODBC データソース名]** をクリックし、**SQL Anywhere 12 CustDB** を選択します。

c. **[接続]** をクリックします。

10. 同期が成功したことを確認します。

    Interactive SQL で次の SQL 文を実行します。

    ```
    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
    AND c.cust_name = 'Ace Properties'
    AND p.prod_name = '4x8 Drywall x100';
    ```

    Interactive SQL に注文にエントリが表示されたら、同期が成功しています。

11. [Simulator] ウィンドウを閉じます。

クリーンアップ

チュートリアルをコンピューターから削除します。

♦ コンピューターからのチュートリアルの削除

1. Eclipse を終了します。

   [File] » [Exit] をクリックします。

2. タスクバー上で、Mobile Link、Interactive SQL、同期クライアントの各ウィンドウを右クリックし、**[終了]** または **[シャットダウン]** をクリックします。

3. CustDB データベースをリセットします。

   %SQLANYSAMP12%¥UltraLite¥CustDB ディレクトリから次のコマンドを実行します。

   ```
   makedbs
   ```

チュートリアル：BlackBerry アプリケーションの構築

このチュートリアルでは、Ultra Light J API および Eclipse 環境を使用して、BlackBerry スマートフォン用のアプリケーションを開発する方法について説明します。このチュートリアルでは、アプリケーションを Windows シミュレーターで実行してから BlackBerry スマートフォンに配備します。チュートリアル全体に渡ってコード例を示しています。また、チュートリアルの最後にすべてのコードを示します。

必要なソフトウェア

- Eclipse 3.5 以降
- BlackBerry Java Plug-in for Eclipse 1.1 以降
- BlackBerry Email および MDS Services Simulator Package v4.1.4
- BlackBerry JDE 6.0 以降
- Ultra Light J API

前提知識と経験

- Java の知識
- Eclipse の知識

参照

- http://us.blackberry.com/developers/javaappdev/
- http://us.blackberry.com/developers/javaappdev/javadevenv.jsp

第 1 部：新しい BlackBerry アプリケーションの作成

第 1 部では、Ultra Light Java Edition データベースで名前のリストを管理する BlackBerry アプリケーションを作成する方法について説明します。第 2 部では、アプリケーションを Mobile Link サーバーと同期する方法について説明します。

レッスン 1：新しい BlackBerry プロジェクトの設定

このレッスンでは、Eclipse 統合開発環境を介して、新しい BlackBerry プロジェクトを作成します。

♦ Eclipse での新しい BlackBerry プロジェクトの設定

1. Eclipse を実行します。

   デフォルトのアプリケーションパスは *C:¥Eclipse¥eclipse.exe* です。

2. **Workspace** フィールドで、作業ディレクトリを指定して、**[OK]** をクリックします。

このチュートリアルでは、作業を *C:¥HelloBlackBerry* ディレクトリで行っていることを前提としています。

3. 新規プロジェクトを作成します。

   [File] » [New] » [Project] をクリックします。

4. **[BlackBerry]** フォルダーを展開してから、**[BlackBerry Project]** を選択します。
5. **[Next]** をクリックします。
6. **[Project Name]** フィールドに、**HelloBlackBerry** と入力します。
7. **[Finish]** をクリックします。
8. プロジェクトに Ultra Light J JAR ファイルを追加します。
   a. Eclipse で **[Package Explorer]** ウィンドウが表示されていな場合は表示します。

      [Window] » [Show View] » [Package Explorer] をクリックします。

   b. プロジェクトのパッケージプロパティにアクセスします。

      [Package Explorer] ウィンドウの **HelloBlackBerry** をクリックし、**[File] » [Properties]** をクリックします。

   c. 左側のウィンドウ枠で **[Java Build Path]** をクリックし、**[Libraries]** タブをクリックします。

   d. **[Add External Jars]** をクリックして、SQL Anywhere インストールディレクトリから *¥UltraLite¥UltraLiteJ¥BlackBerry4.2¥UltraLiteJ12.jar* を開きます。

9. プロジェクトに Ultra Light J Javadoc マニュアルへのパスを追加します。
   a. **[JARs And Class Folders On The Build Path]** リストで、**UltraLiteJ12.jar** を展開し、**[JavaDoc Location]** をクリックします。

   b. **[Edit]** をクリックします。

      [Javadoc For UltraLiteJ12.Jar] ウィンドウが表示されます。

   c. **[Browse]** をクリックして、SQL Anywhere インストールディレクトリから *¥UltraLite¥UltraLiteJ¥BlackBerry4.2¥html* を開きます。

   d. **[OK]** をクリックして、**[Javadoc For UltraLiteJ12.Jar]** ウィンドウを閉じます。

10. **[OK]** をクリックして、ウィンドウを閉じます。

レッスン 2：BlackBerry アプリケーションの作成とテスト

このレッスンでは、**HomeScreen** クラスを開く **main** メソッドがあるクラスを作成します。このクラスにはタイトルとステータスメッセージが表示されます。

♦ Eclipse でのサンプルアプリケーションの作成とテスト

1. プロジェクトに **Application** クラスを追加します。

a. **[Package Explorer]** ウィンドウで、**HelloBlackBerry** を展開して **[src]** をクリックします。

b. **[File]** » **[New]** » **[Class]** をクリックします。

[New Java Class] ウィンドウが表示されます。

c. **[Name]** フィールドに **Application** と入力します。

d. **[Which method stubs would you like to create?]** オプションで、**[Public Static Void Main([String() Args])]** を選択します。

e. **[Finish]** をクリックします。

Application.java ファイルが **[Package Explorer]** ウィンドウで作成中のプロジェクトの下に表示されます。

2. **Application** クラスを変更します。

[Package Explorer] ウィンドウの *Application.java* をダブルクリックしてから、コンストラクターと **main** メソッドを追加します。

Application.java コードは、次のコードのようになります。

```
class Application extends net.rim.device.api.ui.UiApplication {
    public static void main( String[] args )
    {
        Application instance = new Application();
        instance.enterEventDispatcher();
    }
    Application() {
        pushScreen( new HomeScreen() );
    }
}
```

3. プロジェクトに **HomeScreen** クラスを追加します。

a. **[Package Explorer]** ウィンドウで、**HelloBlackBerry** を展開して **[src]** をクリックします。

b. **[File]** » **[New]** » **[Class]** をクリックします。

[New Java Class] ウィンドウが表示されます。

c. **[Name]** フィールドに **HomeScreen** と入力します。

d. **[Finish]** をクリックします。

HomeScreen.java ファイルが **[Package Explorer]** ウィンドウで作成中のプロジェクトの下に表示されます。

4. タイトルとステータスメッセージが表示されるように、**HomeScreen** クラスを変更します。

[Package Explorer] ウィンドウで *HomeScreen.java* をダブルクリックしてから、タイトルとステータスメッセージが表示されるようにコードを更新します。

HomeScreen.java コードは、次のコードのようになります。

```
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import java.util.*;
```

```
class HomeScreen extends MainScreen {

    HomeScreen() {

        // Set the window title
        LabelField applicationTitle = new LabelField("Hello BlackBerry");
        setTitle(applicationTitle);

        // Add a label to show application status
        _statusLabel = new LabelField( "Status: Started" );
        add( _statusLabel );
    }
    private LabelField _statusLabel;
}
```

_statusLabel は、アプリケーションの他の部分からアクセスできるように、クラス変数として定義します。

5. シミュレーターを実行します。

   [Package Explorer] ウィンドウで *Application.java* をクリックしてから、**[Run]** » **[Run As]** » **[BlackBerry Simulator]** をクリックします。

   注意
   ワークスペースで複数のプロジェクトを開いている場合は、**[Run]** » **[Run Configurations]** をクリックし、**HelloBlackBerry** を選択してから **[Run]** をクリックします。

   HelloBlackBerry プロジェクトはコンパイルされ、シミュレーターウィンドウが表示されます。

   Eclipse の **[Problems]** タブを選択して、プロジェクトがエラーなくコンパイルされたことを確認します。

6. シミュレーターメニューで、**[Simulate]** » **[Set IT Policy]** をクリックします。

   [Set IT Policy] ウィンドウが表示されます。

7. **[Policy]** フィールドで、**[Allow Third Party Apps To Use Persistent Store]** » **[>>]** をクリックします。

8. **[Set]** をクリックしてから、**[Close]** をクリックします。

9. アプリケーションを起動します。

   シミュレーターウィンドウで、**[Downloads]** に移動してから **HelloBlackBerry** アプリケーションを実行します。

   Hello BlackBerry タイトルバーと **Status: Started** テキストを示す画面が表示されます。

10. シミュレーションを停止します。

    シミュレーターウィンドウで、**[File]** » **[Exit]** をクリックします。

レッスン 3：Ultra Light Java Edition データベースの作成

このレッスンでは、Ultra Light Java Edition データベースを作成し、そのデータベースに接続するコードを記述します。新しいデータベースを作成するコードは、**DataAccess** というシングルトンクラスで定義され、**HomeScreen** コンストラクターから呼び出されます。シングルトンクラスを使用すると、データベースへの接続が、一度に 1 つしか開かれないよう制御できます。Ultra Light API は複数の接続をサポートしていますが、一般的な設計パターンでは 1 つの接続を使用します。

♦ データベースを作成するためのサンプルアプリケーションの更新

1. **HomeScreen** クラスを変更して **DataAccess** オブジェクトをインスタンス化します。

   次に示すのは、完全で更新された **HomeScreen** クラスコードリストです。

```
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import java.util.*;

class HomeScreen extends MainScreen {

    HomeScreen() {

        // Set the window title
        LabelField applicationTitle = new LabelField("Hello BlackBerry");
        setTitle(applicationTitle);

        // Add a label to show application status
        _statusLabel = new LabelField("Status: Started");
        add(_statusLabel);

        // Create database and connect
        try {
            _da = DataAccess.getDataAccess(true);
            _statusLabel.setText("Status: Connected");
        }
        catch(Exception ex)
        {
            _statusLabel.setText("Exception: " + ex.toString());
        }
    }
    private LabelField _statusLabel;
    private DataAccess _da;
}
```

   Eclipse が **DataAccess** が解決できないという警告を表示する可能性があります。**DataAccess** オブジェクトは、コードの他の部分からアクセスできるように、クラスレベル変数として格納されます。**DataAccess** クラスは、次の手順で作成します。

2. プロジェクトに **DataAccess** クラスを追加します。
   a. **[Package Explorer]** ウィンドウで、**HelloBlackBerry** を展開して **[src]** をクリックします。
   b. **[File]** » **[New]** » **[Class]** をクリックします。
      [New Java Class] ウィンドウが表示されます。
   c. **[Name]** フィールドに **DataAccess** と入力します。

d. **[Finish]** をクリックします。

DataAccess.java ファイルが **[Package Explorer]** ウィンドウで作成中のプロジェクトの下に表示されます。

3. **DataAccess** クラスを変更して、単一のデータベース接続を確実にする **getDataAccess** メソッドを含むようにします。

[Package Explorer] ウィンドウの *DataAccess.java* をダブルクリックしてから、次の抜粋部分とコードを置き換えます。

```
import com.ianywhere.ultralitej12.*;
import net.rim.device.api.ui.component.*;
import java.util.*;

class DataAccess {
    DataAccess() {
    }

    public static synchronized DataAccess getDataAccess(boolean reset)
        throws Exception
    {
        if (_da == null) {
            _da = new DataAccess();
            ConfigObjectStore config = DatabaseManager.createConfigurationObjectStore("HelloDB");
            if (reset) {
                _conn = DatabaseManager.createDatabase(config);
                // _da.createDatabaseSchema();
            }
            else {
                try {
                    _conn = DatabaseManager.connect(config);
                }
                catch (ULjException uex1) {
                    if (uex1.getErrorCode() !=
ULjException.SQLE_ULTRALITE_DATABASE_NOT_FOUND) {
                        Dialog.alert("Exception: " + uex1.toString() + ". Recreating database...");
                    }
                    _conn = DatabaseManager.createDatabase(config);
                    // _da.createDatabaseSchema();
                }
            }
        }
        return _da;
    }
    private static Connection _conn;
    private static DataAccess _da;
}
```

このクラスは、*UltraLiteJ12.jar* ファイルから **com.ianywhere.ultralitej12** パッケージをインポートします。次の手順は、Ultra Light Java Edition データベースを作成したり、そのデータベースに接続するために必要です。

a. 設定を定義します。このチュートリアルでは、設定オブジェクトは **ConfigObjectStore** インタフェースによって定義されます。このインタフェースによって、BlackBerry オブジェクトストア上に存在する永続的なデータベースを設定できます。

b. データベースへの接続を試行します。このチュートリアルでは、データベースは、接続が失敗したときに **createDatabase** メソッドを使って作成されます。このメソッドが、開いた接続を返します。

4. **[File] » [Save]** をクリックします。

5. シミュレーターを実行します。

   [Package Explorer] ウィンドウで *Application.java* をクリックしてから、**[Run] » [Run As] » [BlackBerry Simulator]** をクリックします。

   注意
   ワークスペースで複数のプロジェクトを開いている場合は、**[Run] » [Run Configurations]** をクリックし、**HelloBlackBerry** を選択してから **[Run]** をクリックします。

   HelloBlackBerry プロジェクトがコンパイルされ、シミュレーターウィンドウが表示されます。

   Eclipse の **[Problems]** タブを選択して、プロジェクトがエラーなくコンパイルされたことを確認します。

6. シミュレーターメニューで、**[File] » [Load Java Program]** をクリックします。

7. SQL Anywhere インストール環境の *¥UltraLite¥UltraLiteJ¥BlackBerry4.2¥* ディレクトリに移動して、*UltraLiteJ12.cod* ファイルを開きます。

   注意
   アプリケーションを実行するには、*UltraLiteJ12.cod* と DBG ファイルを作業中のシミュレーターディレクトリ (*C:¥Eclipse¥plugins¥net.rim.ejde.componentpack6.0.0_6.0.0.0.26¥components¥simulator¥* など) にコピーする必要があります。コピーが完了したら、シミュレーターメニューから Java プログラムをロードする必要はありません。

8. シミュレーターメニューで、**[Simulate] » [Set IT Policy]** をクリックします。

   [Set IT Policy] ウィンドウが表示されます。

9. **[Policy]** フィールドで、**[Allow Third Party Apps to Use Persistent Store]** をクリックし、**[>>]** をクリックします。

10. **[Set]** をクリックしてから、**[Close]** をクリックします。

11. アプリケーションを起動します。

    シミュレーターウィンドウで、**[Downloads]** に移動してから **HelloBlackBerry** アプリケーションを実行します。

    Hello BlackBerry タイトルバーと **Status: Connected** テキストを示す画面が表示されます。画面には、アプリケーションが Ultra Light Java Edition データベースに正常に接続されたことが表示されます。

12. シミュレーションを停止します。

    シミュレーターウィンドウで、**[File] » [Exit]** をクリックします。

参照

- ConfigObjectStore インターフェイス [BlackBerry] [Ultra Light J]84 ページ
- DatabaseManager.createDatabase メソッド [Ultra Light J]136 ページ

レッスン 4：データベースでのテーブルの作成

このレッスンでは、次のプロパティを持つ 2 つのカラムが含まれる **Names** という名前のテーブルを作成します。

カラム名	データ型	NULL 入力可	デフォルト	プライマリキー
ID	UUID	いいえ	なし	はい
Name	varchar(254)	いいえ	なし	いいえ

♦ データベースでテーブルを作成するためのサンプルアプリケーションの更新

1. **Names** テーブルを作成する **DataAccess** クラスに新しいメソッドを追加します。

   [Package Explorer] ウィンドウの *DataAccess.java* をダブルクリックしてから、**getDataAccess** メソッドの後ろに次のコードを挿入します。

   ```
   private void createDatabaseSchema() {
      try {
         String sql = "CREATE TABLE Names (ID UNIQUEIDENTIFIER DEFAULT NEWID(), Name
   VARCHAR(254), " +
            "PRIMARY KEY (ID))";
         PreparedStatement ps = _conn.prepareStatement(sql);
         ps.execute();
         ps.close();
      }
      catch (ULjException uex1) {
         Dialog.alert("ULjException: " + uex1.toString());
      }
      catch (Exception ex1) {
         Dialog.alert("Exception: " + ex1.toString());
      }
   }
   ```

   このメソッドは、データベースに **Names** テーブルがすでに存在する場合は、例外をスローします。

2. **getDataAccess** メソッドから **createDatabaseSchema** メソッドを呼び出します。

   createDatabaseSchema 呼び出しが次のコード抜粋に似たかたちになるように、**getDataAccess** メソッドからコードコメントを削除します。

   ```
   _da.createDatabaseSchema()
   ```

3. 作成した **DataAccess** コードと **DataAccess** クラスの完全なコードリストと比較して、同一であることを確認します。

4. **[File]** » **[Save]** をクリックします。

5. シミュレーターを実行して、アプリケーションがコンパイルされ、実行されることを確認します。

   [Package Explorer] ウィンドウで *Application.java* をクリックしてから、**[Run]** » **[Run As]** » **[BlackBerry Simulator]** をクリックします。

   注意
   ワークスペースで複数のプロジェクトを開いている場合は、**[Run]** » **[Run Configurations]** をクリックし、**HelloBlackBerry** を選択してから **[Run]** をクリックします。

   HelloBlackBerry プロジェクトがコンパイルされ、シミュレーターウィンドウが表示されます。

   Eclipse の **[Problems]** タブを選択して、プロジェクトがエラーなくコンパイルされたことを確認します。

6. シミュレーターメニューで、**[File]** » **[Load Java Program]** をクリックします。

7. SQL Anywhere インストール環境の *¥UltraLite¥UltraLiteJ¥BlackBerry4.2¥* ディレクトリに移動して、*UltraLiteJ12.cod* ファイルを開きます。

   注意
   アプリケーションを実行するには、*UltraLiteJ12.cod* と DBG ファイルを作業中のシミュレーターディレクトリ (*C:¥Eclipse¥plugins¥net.rim.ejde.componentpack6.0.0_6.0.0.0.26¥components¥simulator¥* など) にコピーする必要があります。コピーが完了したら、シミュレーターメニューから Java プログラムをロードする必要はありません。

8. シミュレーターメニューで、**[Simulate]** » **[Set IT Policy]** をクリックします。

   [Set IT Policy] ウィンドウが表示されます。

9. **[Policy]** » **[Allow Third Party Apps to Use Persistent Store]** をクリックし、**[>>]** をクリックします。

10. **[Set]** をクリックしてから、**[Close]** をクリックします。

11. アプリケーションを起動します。

    シミュレーターウィンドウで、**[Downloads]** に移動してから **HelloBlackBerry** アプリケーションを実行します。

    Hello BlackBerry タイトルバーと **Status: Connected** テキストを示す画面が表示されます。画面には、アプリケーションが Ultra Light Java Edition データベースに正常に接続されたことが表示されます。

12. シミュレーションを停止します。

    シミュレーターウィンドウで、**[File]** » **[Exit]** をクリックします。

参照

- 「DataAccess.java」 65 ページ

レッスン 5：テーブルへのデータの追加

このレッスンでは、アプリケーションに次のコントロールを追加します。

●名前を入力できるテキストフィールド。

●テキストフィールドの名前をデータベースに追加するためのメニュー項目。

●テーブル内の名前を表示するリストフィールド。

次に、テキストフィールドに名前を挿入し、リストを更新するためのコードを追加します。

♦ ユーザーにデータを要求するためのサンプルアプリケーションの更新

1. **HomeScreen** クラスを更新して制御を追加します。

   [Package Explorer] ウィンドウの *HomeScreen.java* をダブルクリックしてから、**getDataAccess** メソッドを呼び出す **try-catch** 文の前に次のコードを挿入します。

   ```
   // Add an edit field for entering new names
   _nameEditField = new EditField( "Name: ", "", 50, EditField.USE_ALL_WIDTH );
   add (_nameEditField );

   // Add an ObjectListField for displaying a list of names
   _nameListField = new ObjectListField();
   add( _nameListField );

   // Add a menu item
   addMenuItem(_addToListMenuItem);
   ```

2. **_nameEditField** と **_nameListField** のクラスレベルの宣言を追加してから、**run** メソッドで **MenuItem** を定義します (現在は空です)。これらの宣言は、**_statusLabel** と **_da** の宣言の次にあります。

   次のコードを **private DataAccess _da;** 文の下に挿入します。

   ```
   private EditField _nameEditField;
   private ObjectListField _nameListField;

   private MenuItem _addToListMenuItem = new MenuItem("Add", 1, 1){
      public void run() {
         // TODO
      }
   };
   ```

3. テーブルにローを挿入する **DataAccess** クラスに新しいメソッドを追加します。

   [Package Explorer] ウィンドウの *DataAccess.java* をダブルクリックしてから、**createDatabaseSchema** メソッドの後ろに次のコードを挿入します。

   ```
   public void insertName(String name){
      try {
   ```



```
            UUIDValue nameID = _conn.createUUIDValue();
            String sql = "INSERT INTO Names(ID, Name) VALUES(?, ?)";
            PreparedStatement ps = _conn.prepareStatement(sql);
            ps.set(1, nameID);
            ps.set(2, name);
            ps.execute();
            _conn.commit();
            ps.close();
        }
        catch(ULjException uex) {
            Dialog.alert("ULjException: " + uex.toString());
        }
        catch( Exception ex ){
            Dialog.alert("Exception: " + ex.toString());
        }
    }
```

4. プロジェクトに **NameRow** クラスを追加します。
   a. **[Package Explorer]** ウィンドウで、**HelloBlackBerry** を展開して **[src]** をクリックします。
   b. **[File]** » **[New]** » **[Class]** をクリックします。
      [New Java Class] ウィンドウが表示されます。
   c. **[Name]** フィールドに **NameRow** と入力します。
   d. **[Finish]** をクリックします。
      NameRow.java ファイルが **[Package Explorer]** ウィンドウで作成中のプロジェクトの下に表示されます。
5. **Names** テーブルにオブジェクトとしてローを格納できるように、**NameRow** クラスを更新します。

   [Package Explorer] ウィンドウの *NameRow.java* をダブルクリックしてから、次の抜粋部分とコードを置き換えます。

```
class NameRow {

    public NameRow( String nameID, String name ) {
        _nameID = nameID;
        _name = name;
    }

    public String getNameID(){
        return _nameID;
    }

    public String getName(){
        return _name;
    }

    public String toString(){
        return _name;
    }

    private String _nameID;
    private String _name;

}
```

   toString メソッドは、**ObjectListField** コントロールによって使用されます。

6. オブジェクトのベクトルにローを読み込む **DataAccess** クラスに新しいメソッドを追加します。

   [Package Explorer] ウィンドウの *DataAccess.java* をダブルクリックしてから、**insertName** メソッドの後ろに次のコードを挿入します。

```
public Vector getNameVector(){
    Vector nameVector = new Vector();
    try {
        String sql = "SELECT ID, Name FROM Names";
        PreparedStatement ps = _conn.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String nameID = rs.getString(1);
            String name = rs.getString(2);
            NameRow nr = new NameRow(nameID, name);
            nameVector.addElement(nr);
        }
    }
    catch(ULjException uex) {
        Dialog.alert("ULjException: " + uex.toString());
    }
    catch(Exception ex) {
        Dialog.alert("Exception: " + ex.toString());
    }
    return nameVector;
}
```

7. 画面に表示されているリストの内容を再表示する新しいメソッドを **HomeScreen** クラスに追加します。

   [Package Explorer] ウィンドウの *HomeScreen.java* をダブルクリックしてから、**_addToListMenuItem** メソッドの後ろに次のコードを挿入します。

```
public void refreshNameList() {
    //Clear the list
    _nameListField.setSize(0);

    //Refill from the list of names
    Vector nameVector = _da.getNameVector();
    for( Enumeration e = nameVector.elements(); e.hasMoreElements(); ){
        NameRow nr = ( NameRow )e.nextElement();
        _nameListField.insert(0, nr);
    }
}
```

8. **refreshNameList** メソッドを呼び出せるように **HomeScreen** クラスを更新して、アプリケーションが開始したときにリストに値が入っているようにします。

   HomeScreen コンストラクターの最後の前に次のコードを挿入します。

```
// Fill the ObjectListField
refreshNameList();
```

9. 画面のリストにローを追加する新しいメソッドを **HomeScreen** クラスに追加します。

   次のコードを **refreshNameList** メソッドの後ろに挿入します。

```
private void onAddToList(){
    String name = _nameEditField.getText();
```



```
        _da.insertName(name);
        this.refreshNameList();
        _nameEditField.setText("");
        _statusLabel.setText(name + " added to list");
    }
```

10. **onAddToList** メソッドを呼び出すように、**HomeScreen** クラスの **run** メソッドを更新します。

    // TODO を示すコードの行を次のコード抜粋と置き換えます。

    ```
            onAddToList();
    ```

11. **[File] » [Save All]** をクリックします。

12. シミュレーターを実行して、アプリケーションがコンパイルされ、実行されることを確認します。

    [Package Explorer] ウィンドウで *Application.java* をクリックしてから、**[Run] » [Run As] » [BlackBerry Simulator]** をクリックします。

    注意
    ワークスペースで複数のプロジェクトを開いている場合は、**[Run] » [Run Configurations]** をクリックし、**HelloBlackBerry** を選択してから **[Run]** をクリックします。

    HelloBlackBerry プロジェクトがコンパイルされ、シミュレーターウィンドウが表示されます。

    Eclipse の **[Problems]** タブを選択して、プロジェクトがエラーなくコンパイルされたことを確認します。

13. シミュレーターメニューで、**[File] » [Load Java Program]** をクリックします。

14. SQL Anywhere インストール環境の *¥UltraLite¥UltraLiteJ¥BlackBerry4.2¥* ディレクトリに移動して、*UltraLiteJ12.cod* ファイルを開きます。

    注意
    アプリケーションを実行するには、*UltraLiteJ12.cod* と DBG ファイルを作業中のシミュレーターディレクトリ (*C:¥Eclipse¥plugins¥net.rim.ejde.componentpack6.0.0_6.0.0.0.26¥components¥simulator¥* など) にコピーする必要があります。コピーが完了したら、シミュレーターメニューから Java プログラムをロードする必要はありません。

15. シミュレーターメニューで、**[Simulate] » [Set IT Policy]** をクリックします。

    [Set IT Policy] ウィンドウが表示されます。

16. **[Policy]** フィールドで、**[Allow Third Party Apps To Use Persistent Store] » [>>]** をクリックします。

17. **[Set]** をクリックしてから、**[Close]** をクリックします。

18. アプリケーションを起動します。

シミュレーターウィンドウで、**[Downloads]** に移動してから **HelloBlackBerry** アプリケーションを実行します。

Hello BlackBerry タイトルバー、**Status: Connected** テキスト、**Name** フィールドを示す画面が表示されます。

19. [Name] フィールドに **John Smith** と入力します。

20. ***EMPTY*** をクリックし、**Add** を選択します。

    リストに **John Smith** が表示され、これは、データベースの **Names** テーブルに追加された名前のエントリを示しています。

    名前は、追加した時点でデータベースに格納されます。アプリケーションを閉じたり再度開いたりするときに、名前がデータベースから検索され、リストに追加されます。

21. シミュレーションを停止します。

    シミュレーターウィンドウで、**[File] » [Exit]** をクリックします。

レッスン 6 : BlackBerry スマートフォンへのアプリケーションの配備

BlackBerry スマートフォンにアプリケーションを配備するには、いくつかの方法があります。このレッスンでは、BlackBerry Desktop Manager ソフトウェアを使用してアプリケーションを配備する方法について説明します。

BlackBerry 上で実行するアプリケーションは、BlackBerry Signature Tool を使用して署名する必要があります。このツールは、BlackBerry JDE Component Package の一部として Research in Motion (RIM) から入手できます。*UltraLiteJ12.cod* ファイルはすでに署名されていますが、*HelloBlackBerry.cod* ファイルは署名する必要があります。

注意
BlackBerry Signature Tool を使用してアプリケーションに署名できるように、RIM からキーを取得してください。キーの取得については、BlackBerry Developer Program の次の Web サイトを参照してください。http://na.blackberry.com/eng/developers/.

♦ アプリケーションの署名と配備

1. BlackBerry Signature Tool を起動します。

2. BlackBerry JDE インストール環境 (例：*C:¥Program Files¥Research in Motion¥BlackBerry JDE 6.0.0¥bin*) の *¥bin* ディレクトリから次のコマンドを実行します。

   ```
   start javaw -jar SignatureTool.jar
   ```

3. *C:¥HelloBlackBerry* まで移動して、コンパイル済みアプリケーションである *HelloBlackBerry.cod* ファイルを選択します。

4. **[Request To Sign The File]** をクリックします。
5. **[Close]** をクリックして Signature Tool を閉じます。
6. USB ケーブルを使用して BlackBerry をコンピューターに接続し、BlackBerry Desktop Manager からデバイスを認識できることを確認します。
7. **[Application Loader]** をクリックして、ウィザードの指示に従います。
8. *HelloBlackBerry.alx* ファイルを参照し、デバイスに追加します。
9. *BlackBerry4.2¥UltraLiteJ.alx* ファイルを参照し、デバイスに追加します。

   これで、BlackBerry スマートフォンでアプリケーションを使用できるようになりました。

第 2 部：BlackBerry アプリケーションを同期するための Mobile Link の使用

チュートリアルの第 2 部では、BlackBerry アプリケーションを拡張して Mobile Link 同期をサポートできるようにします。次のタスクを完了します。

●Ultra Light Java Edition データベースで同期できる SQL Anywhere データベースを作成します。

●同期を扱うように、Mobile Link サーバーを開始します。

●Mobile Link 同期をサポートするために BlackBerry アプリケーションを更新します。

●BlackBerry アプリケーションを統合データベースと同期します。

レッスン 1：Mobile Link 統合データベースの設定

データの同期には、Ultra Light データベースが同期する Mobile Link 統合データベースが必要です。このレッスンでは、SQL Anywhere データベースを作成します。

♦ 統合データベースの設定

1. SQL Anywhere データベースに格納する作業ディレクトリを作成します。

   このチュートリアルでは、作業を *c:¥HelloBlackBerry¥database* ディレクトリで行っていることを前提としています。

2. 次のコマンドを実行して、空の SQL Anywhere データベースを作成します。

   ```
   dbinit HelloBlackBerry.db
   ```

3. ODBC データソースを作成してデータベースに接続します。

a. **[スタート]** » **[プログラム]** » **[SQL Anywhere 12]** » **[管理ツール]** » **[ODBC データソースアドミニストレーター]** をクリックします。
b. **[ユーザー DSN]** タブをクリックしてから、**[追加]** をクリックします。
c. **[データ ソースの新規作成]** ウィンドウで、**[SQL Anywhere 12]** をクリックし、**[完了]** をクリックします。
d. **[ODBC]** タブをクリックします。
e. **[データソース名]** フィールドに **HelloBlackBerry** と入力します。
f. **[ログイン]** タブをクリックします。
g. **[ユーザー ID]** フィールドに **DBA** と入力します。
h. **[パスワード]** フィールドに **sql** と入力します。
i. **[アクション]** リストで、**[このコンピューターのデータベースを起動して接続]** をクリックします。
j. **[データベースファイル]** フィールドに *c:¥tutorial¥database¥HelloBlackBerry.db* を入力します。
k. **[サーバー名]** フィールドに **HelloBlackBerry** と入力します。
l. **[最終切断後にデータベースを停止]** オプションを無効にします。
m. **[OK]** をクリックします。
n. **[ODBC データソースアドミニストレーター]** ウィンドウで **[OK]** をクリックします。

4. 次のコマンドを実行して、Interactive SQL を起動し、SQL Anywhere データベースに接続します。

```
dbisql -c dsn=HelloBlackBerry
```

5. Interactive SQL で次の SQL 文を実行し、統合データベースに **Names** テーブルを作成します。

```
CREATE TABLE Names (
   ID UNIQUEIDENTIFIER NOT NULL DEFAULT newID(),
   Name varchar(254),
   PRIMARY KEY (ID)
);
```

6. Interactive SQL を閉じます。

   [ファイル] » **[終了]** をクリックします。

レッスン 2：Mobile Link サーバーの設定と同期モデルの配備

このレッスンでは、Sybase Central を使用して、同期する統合データベースを準備します。

♦ Mobile Link サーバーの設定と同期モデルの配備

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

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

   プロジェクト作成ウィザードが表示されます。

3. **[名前]** フィールドに **rim_project** と入力します。

4. **[保存場所]** フィールドに *C:¥HelloBlackBerry¥database* と入力し、**[次へ]** をクリックします。

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

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

7. **[編集]** をクリックします。

8. **[汎用 ODBC データベースに接続]** ページで次のタスクを実行します。
   a. **[ユーザー ID]** フィールドに **DBA** と入力します。
   b. **[パスワード]** フィールドに **sql** と入力します。
   c. **[ODBC データソース名]** フィールドで、**[参照]** をクリックして **HelloBlackBerry** を選択します。
   d. **[OK]** をクリックし、**[保存]** をクリックします。

9. **[パスワードを記憶]** オプションを選択し、**[次へ]** をクリックします。

10. **[新しいモデルを作成する]** を選択し、**[完了]** をクリックします。

11. **[OK]** をクリックします。

    同期モデル作成ウィザードが表示されます。

12. **[新しい同期モデルの名前を指定してください。]** フィールドに **HelloBlackBerrySyncModel** と入力し、**[次へ]** をクリックします。

13. リストから **HelloBlackBerry** 統合データベースを選択し、**[次へ]** をクリックします。

14. **[いいえ、新しいリモートデータベーススキーマを作成します]** をクリックし、**[次へ]** をクリックします。

15. **[新しいリモートデータベーススキーマ]** ページで、**[リモートデータベースに含める統合データベースのテーブルとカラムを指定してください。]** リストから **Names** テーブルだけが選択されていることを確認して、**[次へ]** をクリックします。

16. **[タイムスタンプベースのダウンロード]** をクリックし、**[完了]** をクリックします。

    タイムスタンプベースのダウンロードでは、前回のダウンロード以降に更新されたデータのみが転送されるため、データ量を最小限に抑えることができます。

17. **[Mobile Link 12]** の下にある Sybase Central の左ウィンドウ枠で、**rim_project**、**[同期モデル]**、**HelloBlackBerrySyncModel** を展開します。

18. **[ファイル]** » **[展開]** をクリックします。

19. **[次の 1 つまたは複数の項目の展開の詳細を指定する]** オプションの下で、**[統合データベース]** オプションが選択されていることを確認します。**[次へ]** をクリックします。

20. **[統合データベースの展開先]** ページで、次のタスクを実行します。

    a. **[次の SQL ファイルに変更を保存する]** を選択し、ファイルのデフォルトロケーションをそのまま使用します。

       Mobile Link により、同期を設定するために統合データベースを変更する *.sql* ファイルが生成されます。以降で *.sql* ファイルを確認し、独自の変更を加えることもできます。この場合は、*.sql* ファイルを手動で実行する必要があります。

    b. 統合データベースに変更をすぐに適用します。

       [統合データベースに接続して変更を直接適用する] を選択します。

    c. リストから **HelloBlackBerry** 統合データベースを選択します。

    d. **[次へ]** をクリックします。

    consolidated ディレクトリを作成するかどうかを確認するプロンプトが表示されます。**[はい]** をクリックします。

21. **[Mobile Link ユーザーおよび同期プロファイル]** ページで、ユーザー名に **mluser**、パスワードに **mlpassword** を入力し、その後、**[完了]** をクリックします。

    これで、統合データベースへの同期モデルを配備できます。

レッスン 3 : BlackBerry アプリケーションへの Mobile Link サポートの追加

このレッスンでは、アプリケーションに同期機能を追加します。

♦ BlackBerry アプリケーションへの Mobile Link 同期機能の追加

1. **HomeScreen** クラスを更新して **Sync** メニュー項目を追加します。

   [Package Explorer] ウィンドウの *HomeScreen.java* をダブルクリックしてから、**getDataAccess** メソッドを呼び出す **try-catch** 文の前に次のコードを挿入します。

   ```
   // Add sync menu item
   addMenuItem(_syncMenuItem);
   ```

2. **HomeScreen** クラスを更新して、クラス変数宣言にメニュー項目を定義する新しいメソッドを追加します。

   次のコードを **_addToListMenuItem** メソッドの下に挿入します。

   ```
   private MenuItem _syncMenuItem = new MenuItem("Sync", 2, 1) {
      public void run() {
         onSync();
      }
   };
   ```

3. **HomeScreen** クラスを更新して、前の手順で呼び出された **onSync** メソッドを追加します。

次のコードを **onAddToList** メソッドの下に挿入します。

```
private void onSync() {
    try {
        if(_da.sync()) {
            _statusLabel.setText("Synchronization succeeded");
        } else {
            _statusLabel.setText("Synchronization failed");
        }
        refreshNameList();
    } catch (Exception ex) {
        Dialog.alert(ex.toString());
    }
}
```

4. **DataAccess** クラスを更新して **_syncParms** 変数を定義します。

   [Package Explorer] ウィンドウの *DataAccess.java* をダブルクリックしてから、**private static DataAccess _da;** 呼び出しの下に次のコードを挿入します。

   ```
   private static SyncParms _syncParms;
   ```

5. **DataAccess** クラスを更新して **sync** メソッドを追加します。

   次のコードを **getNameVector** メソッドの下に挿入します。

   > **注意**
   > *your-host-name* を使用しているコンピューター名と置き換える必要があります。この用語はアプリケーションには使用できません。

   ```
   public boolean sync() {
       try {
           if(_syncParms == null){
               _syncParms = _conn.createSyncParms(SyncParms.HTTP_STREAM,
                           "mluser",
                           "HelloBlackBerrySyncModel");
               _syncParms.setPassword("mlpassword");
               _syncParms.getStreamParms().setHost("your-host-name"); // USE YOUR OWN
               _syncParms.getStreamParms().setPort(8081); // USE YOUR OWN
           }
           _conn.synchronize(_syncParms);
           return true;
       }
       catch(ULjException uex) {
           Dialog.alert("Exception: " + uex.toString());
           return false;
       }
   }
   ```

   同期パラメーターオブジェクト SyncParms には、同期モデルの展開時に指定したユーザー名とパスワードが含まれています。また、作成した同期モデルの名前も含まれています。Mobile Link では、この名前は統合データベースに展開された同期バージョン (同期論理セット) を参照できます。

   ストリームパラメーターオブジェクト **StreamHTTPParms** は、Mobile Link サーバーのホスト名とポート番号を示します。次のレッスンで Mobile Link サーバーを起動するときに、シミュレーターのテスト用に自分のコンピューター名を使用し、使用可能なポートを選択します。

> **注意**
> デバイスを使用する場合は、外部で表示可能なコンピューターを使用するか、自分のデバイスとペアになっている BlackBerry Enterprise Server (Sybase をホストとする Relay Server など) からアクセスできるコンピューターを使用してください。 Relay Server の詳細については、「Relay Server の概要」『Relay Server』を参照してください。

6. **[ファイル]** » **[すべて保存]** をクリックします。

レッスン 4：Mobile Link サーバーの起動とアプリケーションの同期

BlackBerry アプリケーションを実行して同期する前に、Mobile Link サーバーが実行中である必要があります。Device Simulator と Mobile Link 間の通信チャネルを提供するためには、MDS Simulator も実行中である必要があります。

♦ Mobile Link サーバーの起動とアプリケーションの同期

1. *c:¥HelloBlackBerry¥database¥* から次のコマンドを実行して Mobile Link を起動します。

   ```
   mlsrv12 -c "DSN=HelloBlackBerry" -v+ -x http(port=8081) -ot ml.mls
   ```

   -c オプションは、Mobile Link を SQL Anywhere データベースに接続します。-v+ オプションは、高い冗長レベルを設定して、処理中の内容を Mobile Link のサーバーメッセージウィンドウで確認できるようにします。-x オプションは、通信に使用されているポート番号を示します。-ot オプションは、ログファイル (*ml.txt*) が、Mobile Link サーバーを起動したディレクトリに作成されるように指定します。

2. BlackBerry シミュレーターがネットワーク経由で通信できるように、MDS シミュレーターを実行します。

   [スタート] » **[プログラム]** » **[Research In Motion]** » **[BlackBerry Email And MDS Services Simulator 4.1.4]** » **[MDS]** をクリックします。

3. 同期時にアプリケーションによって Ultra Light Java Edition データベースが更新されるように、Mobile Link 統合データベースに名前を追加します。

   a. 次のコマンドを実行して、Interactive SQL を起動し、SQL Anywhere データベースに接続します。

      ```
      dbisql -c dsn=HelloBlackBerry
      ```

   b. Interactive SQL で次の SQL 文を実行し、**Names** テーブルに名前を追加します。

      ```
      INSERT Names (Name) VALUES ('Jane Smith');
      INSERT Names (Name) VALUES ('David Smith');
      COMMIT;
      ```

   c. Interactive SQL を閉じます。

      [ファイル] » **[終了]** をクリックします。

4. Eclipse からシミュレーターを実行します。

[Package Explorer] ウィンドウで *Application.java* をクリックしてから、**[Run]** » **[Run As]** » **[BlackBerry Simulator]** をクリックします。

注意
ワークスペースで複数のプロジェクトを開いている場合は、**[Run]** » **[Run Configurations]** をクリックし、**HelloBlackBerry** を選択してから **[Run]** をクリックします。

HelloBlackBerry プロジェクトがコンパイルされ、シミュレーターウィンドウが表示されます。

Eclipse の **[Problems]** タブを選択して、プロジェクトがエラーなくコンパイルされたことを確認します。

5. シミュレーターメニューで、**[File]** » **[Load Java Program]** をクリックします。
6. SQL Anywhere インストール環境の *¥UltraLite¥UltraLiteJ¥BlackBerry4.2¥* ディレクトリに移動して、*UltraLiteJ12.cod* ファイルを開きます。

注意
アプリケーションを実行するには、*UltraLiteJ12.cod* と DBG ファイルを作業中のシミュレーターディレクトリ (*C:¥Eclipse¥plugins¥net.rim.ejde.componentpack6.0.0_6.0.0.0.26¥components¥simulator¥* など) にコピーする必要があります。コピーが完了したら、シミュレーターメニューから Java プログラムをロードする必要はありません。

7. シミュレーターメニューで、**[Simulate]** » **[Set IT Policy]** をクリックします。

   [Set IT Policy] ウィンドウが表示されます。
8. **[Policy]** フィールドで、**[Allow Third Party Apps To Use Persistent Store]** » **[>>]** をクリックします。
9. **[Set]** をクリックしてから、**[Close]** をクリックします。
10. アプリケーションを起動します。

    シミュレーターウィンドウで、**[Downloads]** に移動してから **HelloBlackBerry** アプリケーションを実行します。

    Hello BlackBerry タイトルバー、**Status: Connected** テキスト、**Name** フィールドを示す画面が表示されます。
11. アプリケーションを Mobile Link サーバーと同期させます。

    EMPTY をクリックし、**Sync** を選択します。

    Jane Smith と **David Smith** がリストに表示され、アプリケーションが Mobile Link 統合データベースと同期できたことを示します。Interactive SQL から Names テーブル内の名前を問い合わせると、これまでにシミュレータに入力した名前がすべてサーバーに到達していることが確認できます。

12. シミュレーションを停止します。

    シミュレーターウィンドウで、**[File]** » **[Exit]** をクリックします。

クリーンアップ

チュートリアルをコンピューターから削除します。

♦ コンピューターからのチュートリアルの削除

1. Eclipse を終了します。

   [File] » **[Exit]** をクリックします。

2. MDS シミュレーターを実行しているコマンドウィンドウを閉じます。

3. Sybase Central および Interactive SQL のそれぞれのタスクバーを右クリックして閉じ、**[終了]** を選択します。

4. 次のように **HelloBlackBerry** データソースを削除します。
   a. ODBC データソースアドミニストレーターを起動します。

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

   b. **[ユーザー データ ソース]** リストから **HelloBlackBerry** を選択し、**[削除]** をクリックします。

   c. ODBC データソースアドミニストレーターを閉じます。

5. *C:¥HelloBlackBerry* ディレクトリを削除します。

チュートリアルのコードリスト

この項では、このチュートリアルの完全なコードを示します。チュートリアルで使用した Java クラスは 4 つあります。これらのクラスは、*%SQLANYSAMP12%¥UltraLiteJ¥BlackBerry ¥HelloBlackBerry¥myapp* にあります。

参照

- 「第 1 部：新しい BlackBerry アプリケーションの作成」43 ページ
- 「第 2 部：BlackBerry アプリケーションを同期するための Mobile Link の使用」57 ページ

Application.java

```
// ***************************************************
// Copyright (c) 2006-2011 iAnywhere Solutions, Inc.
// Portions copyright (c) 2006-2011 Sybase, Inc.
```

```
// All rights reserved. All unpublished rights reserved.
// ***************************************************
// This sample code is provided AS IS, without warranty or liability
// of any kind.
//
// You may use, reproduce, modify and distribute this sample code
// without limitation, on the condition that you retain the foregoing
// copyright notice and disclaimer as to the original iAnywhere code.
//
// *********************************************************************
package myapp;


class Application extends net.rim.device.api.ui.UiApplication {
    public static void main( String[] args )
    {
        Application instance = new Application();
        instance.enterEventDispatcher();
    }
    Application() {
        pushScreen( new HomeScreen() );
    }
}
```

DataAccess.java

```
// ***************************************************
// Copyright (c) 2006-2011 iAnywhere Solutions, Inc.
// Portions copyright (c) 2006-2011 Sybase, Inc.
// All rights reserved. All unpublished rights reserved.
// ***************************************************
// This sample code is provided AS IS, without warranty or liability
// of any kind.
//
// You may use, reproduce, modify and distribute this sample code
// without limitation, on the condition that you retain the foregoing
// copyright notice and disclaimer as to the original iAnywhere code.
//
// *********************************************************************
package myapp;

import com.ianywhere.ultralitej12.*;

import net.rim.device.api.ui.component.*;
import java.util.*;

class DataAccess {
    DataAccess() {
    }

    public static synchronized DataAccess getDataAccess(boolean reset)
        throws Exception
    {
        if (_da == null) {
            _da = new DataAccess();
            ConfigObjectStore config = DatabaseManager.createConfigurationObjectStore("HelloDB");
            if (reset) {
                _conn = DatabaseManager.createDatabase(config);
                _da.createDatabaseSchema();
            }
            else {
```

```
                try {
                    _conn = DatabaseManager.connect(config);
                }
                catch (ULjException uex1) {
                    if (uex1.getErrorCode() != ULjException.SQLE_ULTRALITE_DATABASE_NOT_FOUND) {
                        Dialog.alert("Exception: " + uex1.toString() + ". Recreating database...");
                    }
                    _conn = DatabaseManager.createDatabase(config);
                    _da.createDatabaseSchema();
                }
            }
        }
        return _da;
    }

    private void createDatabaseSchema() {
        try {
            String sql = "CREATE TABLE Names (ID UNIQUEIDENTIFIER DEFAULT NEWID(), Name
VARCHAR(254), " +
                "PRIMARY KEY (ID))";
            PreparedStatement ps = _conn.prepareStatement(sql);
            ps.execute();
            ps.close();
        }
        catch (ULjException uex1) {
            Dialog.alert("ULjException: " + uex1.toString());
        }
        catch (Exception ex1) {
            Dialog.alert("Exception: " + ex1.toString());
        }
    }
    public void insertName(String name){
        try {
            UUIDValue nameID = _conn.createUUIDValue();
            String sql = "INSERT INTO Names(ID, Name) VALUES(?, ?)";
            PreparedStatement ps = _conn.prepareStatement(sql);
            ps.set(1, nameID);
            ps.set(2, name);
            ps.execute();
            _conn.commit();
            ps.close();
        }
        catch(ULjException uex) {
            Dialog.alert("ULjException: " + uex.toString());
        }
        catch( Exception ex ){
            Dialog.alert("Exception: " + ex.toString());
        }
    }
    public Vector getNameVector(){
        Vector nameVector = new Vector();
        try {
            String sql = "SELECT ID, Name FROM Names";
            PreparedStatement ps = _conn.prepareStatement(sql);
            ResultSet rs = ps.executeQuery();
            while ( rs.next() ){
                String nameID = rs.getString(1);
                String name = rs.getString(2);
                NameRow nr = new NameRow( nameID, name);
                nameVector.addElement(nr);
            }
        }
        catch( ULjException uex ){
            Dialog.alert("ULjException: " + uex.toString());
```



```
            }
            catch( Exception ex ){
                Dialog.alert("Exception: " + ex.toString());
            }
            return nameVector;
        }

        public boolean sync() {
            try {
                if(_syncParms == null){
                    _syncParms = _conn.createSyncParms(SyncParms.HTTP_STREAM,
                                "mluser",
                                "HelloBlackBerrySyncModel");
                    _syncParms.setPassword("mlpassword");
                    _syncParms.getStreamParms().setHost("your-host-name"); // USE YOUR OWN
                    _syncParms.getStreamParms().setPort(8081); // USE YOUR OWN
                }
                _conn.synchronize(_syncParms);
                return true;
            }
            catch(ULjException uex) {
                Dialog.alert("Exception: " + uex.toString());
                return false;
            }
        }

        private static Connection _conn;
        private static DataAccess _da;
        private static SyncParms _syncParms;
    }
```

HomeScreen.java

```
// ************************************************
// Copyright (c) 2006-2011 iAnywhere Solutions, Inc.
// Portions copyright (c) 2006-2011 Sybase, Inc.
// All rights reserved. All unpublished rights reserved.
// ************************************************
// This sample code is provided AS IS, without warranty or liability
// of any kind.
//
// You may use, reproduce, modify and distribute this sample code
// without limitation, on the condition that you retain the foregoing
// copyright notice and disclaimer as to the original iAnywhere code.
//
// ***********************************************************************
package myapp;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import java.util.*;

class HomeScreen extends MainScreen {

    HomeScreen() {

        // Set the window title
        LabelField applicationTitle = new LabelField("Hello BlackBerry");
        setTitle(applicationTitle);
```

```
        // Add a label to show application status
        _statusLabel = new LabelField("Status: Started");
        add(_statusLabel);

        // Add an edit field for entering new names
        _nameEditField = new EditField( "Name: ", "", 50, EditField.USE_ALL_WIDTH );
        add (_nameEditField );

        // Add an ObjectListField for displaying a list of names
        _nameListField = new ObjectListField();
        add( _nameListField );

        // Add a menu item
        addMenuItem(_addToListMenuItem);

        // Add sync menu item
        addMenuItem(_syncMenuItem);

        // Create database and connect
        try{
            _da = DataAccess.getDataAccess(true);
            _statusLabel.setText("Status: Connected");
        }
        catch(Exception ex)
        {
            _statusLabel.setText("Exception: " + ex.toString());
        }
        // Fill the ObjectListField
        refreshNameList();
    }

    private LabelField _statusLabel;
    private DataAccess _da;
    private EditField _nameEditField;
    private ObjectListField _nameListField;

    private MenuItem _addToListMenuItem = new MenuItem("Add", 1, 1){
        public void run() {
            onAddToList();
        }
    };
    private MenuItem _syncMenuItem = new MenuItem("Sync", 2, 1){
        public void run() {
            onSync();
        }
    };

    public void refreshNameList() {
        //Clear the list
        _nameListField.setSize(0);

        //Refill from the list of names
        Vector nameVector = _da.getNameVector();
        for( Enumeration e = nameVector.elements(); e.hasMoreElements(); ){
            NameRow nr = ( NameRow )e.nextElement();
            _nameListField.insert(0, nr);
        }
    }

    private void onAddToList(){
        String name = _nameEditField.getText();
        _da.insertName(name);
        this.refreshNameList();
        _nameEditField.setText("");
```



```
        _statusLabel.setText(name + " added to list");
    }

    private void onSync() {
        try {
            if(_da.sync()) {
                _statusLabel.setText("Synchronization succeeded");
            } else {
                _statusLabel.setText("Synchronization failed");
            }
            refreshNameList();
        } catch (Exception ex) {
            Dialog.alert(ex.toString());
        }
    }
}
```

NameRow.java

```
// ****************************************************
// Copyright (c) 2006-2011 iAnywhere Solutions, Inc.
// Portions copyright (c) 2006-2011 Sybase, Inc.
// All rights reserved. All unpublished rights reserved.
// ****************************************************
// This sample code is provided AS IS, without warranty or liability
// of any kind.
//
// You may use, reproduce, modify and distribute this sample code
// without limitation, on the condition that you retain the foregoing
// copyright notice and disclaimer as to the original iAnywhere code.
//
// ********************************************************************
package myapp;

class NameRow {

    public NameRow( String nameID, String name ) {
        _nameID = nameID;
        _name = name;
    }

    public String getNameID(){
        return _nameID;
    }

    public String getName(){
        return _name;
    }

    public String toString(){
        return _name;
    }

    private String _nameID;
    private String _name;

}
```

Ultra Light J API リファレンス

パッケージ [Android]

```
com.ianywhere.ultralitejni12
```

パッケージ [BlackBerry]

```
com.ianywhere.ultralitej12
```

参照

- 「Android と BlackBerry のセットアップの考慮事項」4 ページ

ColumnSchema インターフェイス

カラムのスキーマを指定します。

構文

```
public interface ColumnSchema
```

メンバー

継承されたメンバーを含む ColumnSchema インターフェイスのすべてのメンバー。

名前	説明
COLUMN_DEFAULT_AUTOFILENAME 変数	カラムのデフォルト属性が autofilename である事を示します。
COLUMN_DEFAULT_AUTOINC 変数	カラムのデフォルト属性が autoincrement である事を示します。
COLUMN_DEFAULT_CONSTANT 変数	カラムのデフォルト属性が constant である事を示します。
COLUMN_DEFAULT_CURRENT_DATE 変数	カラムのデフォルト属性が current date (年、月、日) である事を示します。
COLUMN_DEFAULT_CURRENT_TIME 変数	カラムのデフォルト属性が current time である事を示します。
COLUMN_DEFAULT_CURRENT_TIMESTAMP 変数	カラムのデフォルト属性が current timestamp である事を示します。
COLUMN_DEFAULT_CURRENT_UTC_TIMESTAMP 変数	カラムのデフォルト属性が current utc timestamp である事を示します。

名前	説明
COLUMN_DEFAULT_GLOBAL_AUTOINC 変数	カラムのデフォルト属性が global autoincrement である事を示します。
COLUMN_DEFAULT_NONE 変数	カラムにデフォルト属性が存在しない事を示します。
COLUMN_DEFAULT_UNIQUE_ID 変数	カラムのデフォルト属性が新しいユニーク ID である事を示します。

備考

バージョン 12 の時点で、このインターフェイスには、システムテーブル syscolumn の column_default カラムに格納される異なるカラムのデフォルト値に関する制約のみが含まれるようになりました。

COLUMN_DEFAULT_AUTOFILENAME 変数

カラムのデフォルト属性が autofilename である事を示します。

構文

```
final byte ColumnSchema.COLUMN_DEFAULT_AUTOFILENAME
```

備考

VARCHAR カラムにこのデフォルト値がある場合、カラムは外部 blob 定義のファイル名のカラムになります。

カラムにこのタイプのデフォルトがある場合、システムテーブル TableSchema.SYS_COLUMN の column_default_value カラムには、外部 blob 定義にあるプレフィクスと拡張子の文字列が 'prefix|extension' の形式で含まれます。

既存のテーブルのデフォルト値は、システムテーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- TableSchema.SYS_COLUMNS 変数 [Ultra Light J]266 ページ

COLUMN_DEFAULT_AUTOINC 変数

カラムのデフォルト属性が autoincrement である事を示します。

構文

```
final byte ColumnSchema.COLUMN_DEFAULT_AUTOINC
```


備考

AUTOINCREMENT 属性を使用する場合、カラムは整数データ型の 1 つ、または真数値型にします。INSERT 時に AUTOINCREMENT のカラムに値を指定しないと、カラム内のほかの値より大きいユニーク値が生成されます。INSERT で、カラムの現在の最大値より大きい値を指定した場合、この値が後続の挿入処理の開始ポイントとして使用されます。

Ultra Light J では、テーブルが作成された時点でのオートインクリメントの初期値は 0 ではありません。カラムに符号付きデータ型が指定されている場合は、AUTOINCREMENT 属性によって負の値が生成されます。このため、オートインクリメントを適用するカラムを符号なし整数として宣言し、負の値が生成されないようにしてください。

既存のテーブルのデフォルト値は、システムテーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- TableSchema.SYS_COLUMNS 変数 [Ultra Light J]266 ページ

COLUMN_DEFAULT_CONSTANT 変数

カラムのデフォルト属性が constant である事を示します。

構文

```
final byte ColumnSchema.COLUMN_DEFAULT_CONSTANT
```

備考

既存のテーブルのデフォルト値は、システムテーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- TableSchema.SYS_COLUMNS 変数 [Ultra Light J]266 ページ

COLUMN_DEFAULT_CURRENT_DATE 変数

カラムのデフォルト属性が current date (年、月、日) である事を示します。

構文

```
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT_DATE
```

備考

SQL Anywhere のマニュアルセットで、「Ultra Light の特別値」の下の「CURRENT DATE 特別値」を参照してください。

既存のテーブルのデフォルト値は、システムテーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- TableSchema.SYS_COLUMNS 変数 [Ultra Light J]266 ページ

COLUMN_DEFAULT_CURRENT_TIME 変数

カラムのデフォルト属性が current time である事を示します。

構文

```
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT_TIME
```

備考

SQL Anywhere のマニュアルセットで、「Ultra Light の特別値」の下の「CURRENT TIME 特別値」を参照してください。

既存のテーブルのデフォルト値は、システムテーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- TableSchema.SYS_COLUMNS 変数 [Ultra Light J]266 ページ

COLUMN_DEFAULT_CURRENT_TIMESTAMP 変数

カラムのデフォルト属性が current timestamp である事を示します。

構文

```
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT_TIMESTAMP
```

備考

この定数は、CURRENT DATE と CURRENT TIME の値を結合して、TIMESTAMP 値を形成します。この値は、年、月、日、時、分、秒、秒の小数位で構成されます。秒の精度は小数点以下第 3 位に設定されます。この定数の精度はシステムクロックの精度によって制限されます。

SQL Anywhere のマニュアルセットで、「Ultra Light の特別値」の下の「CURRENT TIMESTAMP 特別値」を参照してください。

既存のテーブルのデフォルト値は、システムテーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- TableSchema.SYS_COLUMNS 変数 [Ultra Light J]266 ページ

COLUMN_DEFAULT_CURRENT_UTC_TIMESTAMP 変数

カラムのデフォルト属性が current utc timestamp である事を示します。

構文

```
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT_UTC_TIMESTAMP
```

備考

この定数は、CURRENT DATE と CURRENT TIME の値を結合して、UTC TIMESTAMP 値を形成します。この値は、GMT での年、月、日、時、分、秒、秒の小数位で構成されます。秒の精度は小数点以下第 3 位に設定されます。この定数の精度はシステムクロックの精度によって制限されます。

SQL Anywhere のマニュアルセットで、「Ultra Light の特別値」の下の「CURRENT UTC TIMESTAMP 特別値」を参照してください。

既存のテーブルのデフォルト値は、システムテーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- TableSchema.SYS_COLUMNS 変数 [Ultra Light J]266 ページ

COLUMN_DEFAULT_GLOBAL_AUTOINC 変数

カラムのデフォルト属性が global autoincrement である事を示します。

構文

```
final byte ColumnSchema.COLUMN_DEFAULT_GLOBAL_AUTOINC
```

備考

この定数は AUTOINCREMENT 属性と同じですが、ドメインはパーティションに分割されます。各分割には同じ数の値が含まれます。データベースの各コピーにユニークなグローバルデータベース ID 番号を割り当てる必要があります。Ultra Light J では、データベースのデフォルト値は、そのデータベース番号でユニークに識別された分割から設定されます。

既存のテーブルのデフォルト値は、システムテーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- TableSchema.SYS_COLUMNS 変数 [Ultra Light J]266 ページ
- Connection.setDatabaseId メソッド [Ultra Light J]119 ページ

COLUMN_DEFAULT_NONE 変数

カラムにデフォルト属性が存在しない事を示します。

構文

```
final byte ColumnSchema.COLUMN_DEFAULT_NONE
```

備考

デフォルト属性を指定しない場合、次のデフォルト値が適用されます。

●NULL 入力可のカラムのデフォルト値は NULL

●NULL 入力不可の数値カラムのデフォルト値は 0

●NULL 入力不可の可変長カラムのデフォルト値は長さ 0 の値

既存のテーブルのデフォルト値は、システムテーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- TableSchema.SYS_COLUMNS 変数 [Ultra Light J]266 ページ

COLUMN_DEFAULT_UNIQUE_ID 変数

カラムのデフォルト属性が新しいユニーク ID である事を示します。

構文

```
final byte ColumnSchema.COLUMN_DEFAULT_UNIQUE_ID
```

備考

UUID を使用して、テーブルのローをユニークに識別できます。生成される値は、すべてのコンピューターまたはデバイスでユニークになります。つまり、同期やレプリケーション環境でキーとして使用できます。

既存のテーブルのデフォルト値は、システムテーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- TableSchema.SYS_COLUMNS 変数 [Ultra Light J]266 ページ

ConfigFile インターフェイス

ファイルに保存される永続的なデータベース用の Configuration オブジェクトを確立します。

構文

```
public interface ConfigFile
```

基本クラス

- ConfigPersistent インターフェイス [Ultra Light J]87 ページ

派生クラス

- ConfigFileAndroid インターフェイス [Android] [Ultra Light J]79 ページ

メンバー

継承されたメンバーを含む ConfigFile インターフェイスのすべてのメンバー。

名前	説明
enableAesDBEncryption メソッド [Android]	データベースの AES 暗号化を有効にします。
enableObfuscation メソッド [Android]	データベースの難読化を有効にします。
getAutoCheckpoint メソッド (廃止予定)	自動チェックポイントがオンになっているかどうかを確認します。
getCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を返します。
getConnectionString メソッド [Android]	SetConnectionString メソッドで登録された接続文字列を取得します。
getCreationString メソッド [Android]	SetCreationString メソッドで登録された作成文字列を取得します。
getDatabaseKey メソッド [Android]	SetDatabaseKey メソッドで登録されたデータベース暗号化キーを取得します。
getDatabaseName メソッド	データベース名を返します。
getLazyLoadIndexes メソッド	インデックスの遅延ロードがオンになっているかどうかを確認します。
getPageSize メソッド	データベースのページサイズ (バイト単位) を返します。
getRowScoreFlushSize メソッド [BlackBerry]	現在のロースコアのフラッシュサイズを返します。
getRowScoreMaximum メソッド [BlackBerry]	現在のロースコアの最大サイズを返します。
getUserName メソッド [Android]	setUserName メソッドで設定されたユーザーの名前を取得します。

名前	説明
hasPersistentIndexes メソッド	インデックスが永続的かどうかを確認します。
hasShadowPaging メソッド	シャドウページングがオンになっているかどうかを確認します。
setAutocheckpoint メソッド (廃止予定)	自動チェックポイントをオンに設定します。
setCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を設定します。
setConnectionString メソッド [Android]	データベースの作成または接続に使用される接続文字列を設定します。
setCreationString メソッド [Android]	データベースを作成するために使用される作成文字列を設定します。
setDatabaseKey メソッド [Android]	暗号化キーを設定します。
setDatabaseName メソッド	データベース名を設定します。
setEncryption メソッド [BlackBerry]	暗号化制御を設定します。
setIndexPersistence メソッド [BlackBerry]	永続的なインデックスをオンに設定します。
setLazyLoadIndexes メソッド	必要なときにインデックスをロードするか、起動時にすべてのインデックスを一度にロードするかを設定します。
setPageSize メソッド	データベースのページサイズを設定します。
setPassword メソッド	データベースのパスワードを設定します。
setRowScoreFlushSize メソッド	古いローの削除に使用されるスコアを指定することによって、ロー制限を有効にします。
setRowScoreMaximum メソッド	メモリに保持する最大ロースコアのスレッショルドを設定します。
setShadowPaging メソッド	シャドウページングをオンまたはオフに設定します。
setUserName メソッド [Android]	ユーザーの名前を設定します。
setWriteAtEnd メソッド	停止時にのみ発生するデータベースの永続性を設定します。

名前	説明
writeAtEnd メソッド	データベースで終了時書き込み型の永続性を使用するかどうかを確認します。

備考

ConfigFile インターフェイスを実装するオブジェクトは、DatabaseManager.createConfigurationFile メソッドを使用して作成されます。

ConfigFileAndroid インターフェイス [Android]

Android デバイス上のファイルに保存される永続的なデータベース用の Configuration オブジェクトを確立します。

構文

```
public interface ConfigFileAndroid
```

基本クラス

- ConfigFile インターフェイス [Ultra Light J]76 ページ

メンバー

継承されたメンバーを含む ConfigFileAndroid インターフェイスのすべてのメンバー。

名前	説明
enableAesDBEncryption メソッド [Android]	データベースの AES 暗号化を有効にします。
enableObfuscation メソッド [Android]	データベースの難読化を有効にします。
getAutoCheckpoint メソッド (廃止予定)	自動チェックポイントがオンになっているかどうかを確認します。
getCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を返します。
getConnectionString メソッド [Android]	SetConnectionString メソッドで登録された接続文字列を取得します。
getCreationString メソッド [Android]	SetCreationString メソッドで登録された作成文字列を取得します。
getDatabaseKey メソッド [Android]	SetDatabaseKey メソッドで登録されたデータベース暗号化キーを取得します。
getDatabaseName メソッド	データベース名を返します。

名前	説明
getLazyLoadIndexes メソッド	インデックスの遅延ロードがオンになっているかどうかを確認します。
getPageSize メソッド	データベースのページサイズ (バイト単位) を返します。
getRowScoreFlushSize メソッド [BlackBerry]	現在のロースコアのフラッシュサイズを返します。
getRowScoreMaximum メソッド [BlackBerry]	現在のロースコアの最大サイズを返します。
getUserName メソッド [Android]	setUserName メソッドで設定されたユーザーの名前を取得します。
hasPersistentIndexes メソッド	インデックスが永続的かどうかを確認します。
hasShadowPaging メソッド	シャドウページングがオンになっているかどうかを確認します。
setAutocheckpoint メソッド (廃止予定)	自動チェックポイントをオンに設定します。
setCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を設定します。
setConnectionString メソッド [Android]	データベースの作成または接続に使用される接続文字列を設定します。
setCreationString メソッド [Android]	データベースを作成するために使用される作成文字列を設定します。
setDatabaseKey メソッド [Android]	暗号化キーを設定します。
setDatabaseName メソッド	データベース名を設定します。
setEncryption メソッド [BlackBerry]	暗号化制御を設定します。
setIndexPersistence メソッド [BlackBerry]	永続的なインデックスをオンに設定します。
setLazyLoadIndexes メソッド	必要なときにインデックスをロードするか、起動時にすべてのインデックスを一度にロードするかを設定します。
setPageSize メソッド	データベースのページサイズを設定します。
setPassword メソッド	データベースのパスワードを設定します。

名前	説明
setRowScoreFlushSize メソッド	古いローの削除に使用されるスコアを指定することによって、ロー制限を有効にします。
setRowScoreMaximum メソッド	メモリに保持する最大ロースコアのスレッショルドを設定します。
setShadowPaging メソッド	シャドウページングをオンまたはオフに設定します。
setUserName メソッド [Android]	ユーザーの名前を設定します。
setWriteAtEnd メソッド	停止時にのみ発生するデータベースの永続性を設定します。
writeAtEnd メソッド	データベースで終了時書き込み型の永続性を使用するかどうかを確認します。

備考

ConfigFileAndroid インターフェイスを実装するオブジェクトは、DatabaseManager.createConfigurationFileAndroid メソッドを使用して作成されます。

参照

- DatabaseManager クラス [Ultra Light J]130 ページ
- DatabaseManager.createConfigurationFileAndroid メソッド [Ultra Light J]133 ページ

ConfigFileME インターフェイス [BlackBerry]

J2ME デバイスファイルシステム上のファイルに保存される永続的なデータベース用の Configuration オブジェクトを確立します。

構文

```
public interface ConfigFileME
```

基本クラス

- ConfigPersistent インターフェイス [Ultra Light J]87 ページ

メンバー

継承されたメンバーを含む ConfigFileME インターフェイスのすべてのメンバー。

名前	説明
enableAesDBEncryption メソッド [Android]	データベースの AES 暗号化を有効にします。

名前	説明
enableObfuscation メソッド [Android]	データベースの難読化を有効にします。
getAutoCheckpoint メソッド (廃止予定)	自動チェックポイントがオンになっているかどうかを確認します。
getCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を返します。
getConnectionString メソッド [Android]	SetConnectionString メソッドで登録された接続文字列を取得します。
getCreationString メソッド [Android]	SetCreationString メソッドで登録された作成文字列を取得します。
getDatabaseKey メソッド [Android]	SetDatabaseKey メソッドで登録されたデータベース暗号化キーを取得します。
getDatabaseName メソッド	データベース名を返します。
getLazyLoadIndexes メソッド	インデックスの遅延ロードがオンになっているかどうかを確認します。
getPageSize メソッド	データベースのページサイズ (バイト単位) を返します。
getRowScoreFlushSize メソッド [BlackBerry]	現在のロースコアのフラッシュサイズを返します。
getRowScoreMaximum メソッド [BlackBerry]	現在のロースコアの最大サイズを返します。
getUserName メソッド [Android]	setUserName メソッドで設定されたユーザーの名前を取得します。
hasPersistentIndexes メソッド	インデックスが永続的かどうかを確認します。
hasShadowPaging メソッド	シャドウページングがオンになっているかどうかを確認します。
setAutocheckpoint メソッド (廃止予定)	自動チェックポイントをオンに設定します。
setCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を設定します。
setConnectionString メソッド [Android]	データベースの作成または接続に使用される接続文字列を設定します。

名前	説明
setCreationString メソッド [Android]	データベースを作成するために使用される作成文字列を設定します。
setDatabaseKey メソッド [Android]	暗号化キーを設定します。
setDatabaseName メソッド	データベース名を設定します。
setEncryption メソッド [BlackBerry]	暗号化制御を設定します。
setIndexPersistence メソッド [BlackBerry]	永続的なインデックスをオンに設定します。
setLazyLoadIndexes メソッド	必要なときにインデックスをロードするか、起動時にすべてのインデックスを一度にロードするかを設定します。
setPageSize メソッド	データベースのページサイズを設定します。
setPassword メソッド	データベースのパスワードを設定します。
setRowScoreFlushSize メソッド	古いローの削除に使用されるスコアを指定することによって、ロー制限を有効にします。
setRowScoreMaximum メソッド	メモリに保持する最大ロースコアのスレッショルドを設定します。
setShadowPaging メソッド	シャドウページングをオンまたはオフに設定します。
setUserName メソッド [Android]	ユーザーの名前を設定します。
setWriteAtEnd メソッド	停止時にのみ発生するデータベースの永続性を設定します。
writeAtEnd メソッド	データベースで終了時書き込み型の永続性を使用するかどうかを確認します。

備考

ConfigFileME インターフェイスを実装するオブジェクトは、DatabaseManager.createConfigurationFileME メソッドを使用して作成されます。

参照

- DatabaseManager クラス [Ultra Light J]130 ページ
- DatabaseManager.createConfigurationFileME メソッド [BlackBerry] [Ultra Light J]134 ページ

ConfigNonPersistent インターフェイス [BlackBerry]

非永続的な (メモリ内) データベース用の Configuration オブジェクトを確立します。

構文

```
public interface ConfigNonPersistent
```

基本クラス

- Configuration インターフェイス [Ultra LightJ]103 ページ

メンバー

継承されたメンバーを含む ConfigNonPersistent インターフェイスのすべてのメンバー。

名前	説明
getDatabaseName メソッド	データベース名を返します。
getPageSize メソッド	データベースのページサイズ (バイト単位) を返します。
setDatabaseName メソッド	データベース名を設定します。
setPageSize メソッド	データベースのページサイズを設定します。
setPassword メソッド	データベースのパスワードを設定します。

備考

ConfigNonPersistent インターフェイスを実装するオブジェクトは、DatabaseManager.createConfigurationNonPersistent メソッドを使用して作成されます。

NonPersistent オブジェクトを作成すると、メモリ内だけに存在するデータベースストアが構成されます。データベースは起動時に作成され、アプリケーションの実行中に使用され、アプリケーションの終了時に破棄されます。アプリケーションの終了時には、非永続ストアに含まれるデータがすべて削除されます。

参照

- DatabaseManager クラス [Ultra Light J]130 ページ

ConfigObjectStore インターフェイス [BlackBerry]

オブジェクトストアに保存される永続的なデータベース用の Configuration オブジェクトを確立します。

構文

```
public interface ConfigObjectStore
```

基本クラス

- ConfigPersistent インターフェイス [Ultra Light J]87 ページ

メンバー

継承されたメンバーを含む ConfigObjectStore インターフェイスのすべてのメンバー。

名前	説明
enableAesDBEncryption メソッド [Android]	データベースの AES 暗号化を有効にします。
enableObfuscation メソッド [Android]	データベースの難読化を有効にします。
getAutoCheckpoint メソッド (廃止予定)	自動チェックポイントがオンになっているかどうかを確認します。
getCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を返します。
getConnectionString メソッド [Android]	SetConnectionString メソッドで登録された接続文字列を取得します。
getCreationString メソッド [Android]	SetCreationString メソッドで登録された作成文字列を取得します。
getDatabaseKey メソッド [Android]	SetDatabaseKey メソッドで登録されたデータベース暗号化キーを取得します。
getDatabaseName メソッド	データベース名を返します。
getLazyLoadIndexes メソッド	インデックスの遅延ロードがオンになっているかどうかを確認します。
getPageSize メソッド	データベースのページサイズ (バイト単位) を返します。
getRowScoreFlushSize メソッド [BlackBerry]	現在のロースコアのフラッシュサイズを返します。
getRowScoreMaximum メソッド [BlackBerry]	現在のロースコアの最大サイズを返します。
getUserName メソッド [Android]	setUserName メソッドで設定されたユーザーの名前を取得します。
hasPersistentIndexes メソッド	インデックスが永続的かどうかを確認します。

名前	説明
hasShadowPaging メソッド	シャドウページングがオンになっているかどうかを確認します。
setAutocheckpoint メソッド (廃止予定)	自動チェックポイントをオンに設定します。
setCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を設定します。
setConnectionString メソッド [Android]	データベースの作成または接続に使用される接続文字列を設定します。
setCreationString メソッド [Android]	データベースを作成するために使用される作成文字列を設定します。
setDatabaseKey メソッド [Android]	暗号化キーを設定します。
setDatabaseName メソッド	データベース名を設定します。
setEncryption メソッド [BlackBerry]	暗号化制御を設定します。
setIndexPersistence メソッド [BlackBerry]	永続的なインデックスをオンに設定します。
setLazyLoadIndexes メソッド	必要なときにインデックスをロードするか、起動時にすべてのインデックスを一度にロードするかを設定します。
setPageSize メソッド	データベースのページサイズを設定します。
setPassword メソッド	データベースのパスワードを設定します。
setRowScoreFlushSize メソッド	古いローの削除に使用されるスコアを指定することによって、ロー制限を有効にします。
setRowScoreMaximum メソッド	メモリに保持する最大ロースコアのスレッショルドを設定します。
setShadowPaging メソッド	シャドウページングをオンまたはオフに設定します。
setUserName メソッド [Android]	ユーザーの名前を設定します。
setWriteAtEnd メソッド	停止時にのみ発生するデータベースの永続性を設定します。
writeAtEnd メソッド	データベースで終了時書き込み型の永続性を使用するかどうかを確認します。

備考

ConfigObjectStore インターフェイスを実装するオブジェクトは、DatabaseManager.createConfigurationObjectStore メソッドを使用して作成されます。

参照

- DatabaseManager クラス [Ultra Light J]130 ページ
- DatabaseManager.createConfigurationObjectStore メソッド [BlackBerry] [Ultra Light J]135 ページ

ConfigPersistent インターフェイス

永続的なデータベース用の Configuration オブジェクトを確立します。

構文

```
public interface ConfigPersistent
```

基本クラス

- Configuration インターフェイス [Ultra LightJ]103 ページ

派生クラス

- ConfigFile インターフェイス [Ultra Light J]76 ページ
- ConfigFileME インターフェイス [BlackBerry] [Ultra Light J]81 ページ
- ConfigObjectStore インターフェイス [BlackBerry] [Ultra Light J]84 ページ
- ConfigRecordStore インターフェイス (J2ME のみ) [UltraLiteJ]100 ページ

メンバー

継承されたメンバーを含む ConfigPersistent インターフェイスのすべてのメンバー。

名前	説明
enableAesDBEncryption メソッド [Android]	データベースの AES 暗号化を有効にします。
enableObfuscation メソッド [Android]	データベースの難読化を有効にします。
getAutoCheckpoint メソッド (廃止予定)	自動チェックポイントがオンになっているかどうかを確認します。
getCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を返します。
getConnectionString メソッド [Android]	SetConnectionString メソッドで登録された接続文字列を取得します。
getCreationString メソッド [Android]	SetCreationString メソッドで登録された作成文字列を取得します。

名前	説明
getDatabaseKey メソッド [Android]	SetDatabaseKey メソッドで登録されたデータベース暗号化キーを取得します。
getDatabaseName メソッド	データベース名を返します。
getLazyLoadIndexes メソッド	インデックスの遅延ロードがオンになっているかどうかを確認します。
getPageSize メソッド	データベースのページサイズ (バイト単位) を返します。
getRowScoreFlushSize メソッド [BlackBerry]	現在のロースコアのフラッシュサイズを返します。
getRowScoreMaximum メソッド [BlackBerry]	現在のロースコアの最大サイズを返します。
getUserName メソッド [Android]	setUserName メソッドで設定されたユーザーの名前を取得します。
hasPersistentIndexes メソッド	インデックスが永続的かどうかを確認します。
hasShadowPaging メソッド	シャドウページングがオンになっているかどうかを確認します。
setAutocheckpoint メソッド (廃止予定)	自動チェックポイントをオンに設定します。
setCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を設定します。
setConnectionString メソッド [Android]	データベースの作成または接続に使用される接続文字列を設定します。
setCreationString メソッド [Android]	データベースを作成するために使用される作成文字列を設定します。
setDatabaseKey メソッド [Android]	暗号化キーを設定します。
setDatabaseName メソッド	データベース名を設定します。
setEncryption メソッド [BlackBerry]	暗号化制御を設定します。
setIndexPersistence メソッド [BlackBerry]	永続的なインデックスをオンに設定します。
setLazyLoadIndexes メソッド	必要なときにインデックスをロードするか、起動時にすべてのインデックスを一度にロードするかを設定します。

名前	説明
setPageSize メソッド	データベースのページサイズを設定します。
setPassword メソッド	データベースのパスワードを設定します。
setRowScoreFlushSize メソッド	古いローの削除に使用されるスコアを指定することによって、ロー制限を有効にします。
setRowScoreMaximum メソッド	メモリに保持する最大ロースコアのスレッショルドを設定します。
setShadowPaging メソッド	シャドウページングをオンまたはオフに設定します。
setUserName メソッド [Android]	ユーザーの名前を設定します。
setWriteAtEnd メソッド	停止時にのみ発生するデータベースの永続性を設定します。
writeAtEnd メソッド	データベースで終了時書き込み型の永続性を使用するかどうかを確認します。

備考

データベースのタイプは次のように決定されます。シャドウページングがオンになっている場合、データベースは、シャドウページングの永続的なデータベースとなります。これ以外の場合は、終了時書き込みがオンになっていると、停止時にのみ永続的なメモリ内データベースとなります。シャドウページングと終了時書き込みがどちらもオフの場合は、非シャドウページングの永続的なデータベースとなります。

遅延ロード、ロースコアのフラッシュサイズ、ロースコアの最大サイズなどのオプションは、シャドウの永続的なデータベースと非シャドウの永続的なデータベースにのみ適用されます。

enableAesDBEncryption メソッド [Android]

データベースの AES 暗号化を有効にします。

構文

```
void ConfigPersistent.enableAesDBEncryption()
```

enableObfuscation メソッド [Android]

データベースの難読化を有効にします。

構文

```
void ConfigPersistent.enableObfuscation()
```

getAutoCheckpoint メソッド (廃止予定)

自動チェックポイントがオンになっているかどうかを確認します。

構文

```
boolean ConfigPersistent.getAutoCheckpoint()
```

戻り値

データベースで自動チェックポイントがオンになっている場合は true、それ以外の場合は false。自動チェックポイントがオンになっているかどうかを判別します。データベースで自動チェックポイントがオンになっている場合は true、それ以外の場合は false です。

getCacheSize メソッド

データベースのキャッシュサイズ (バイト単位) を返します。

構文

```
int ConfigPersistent.getCacheSize()
```

戻り値

キャッシュサイズ。

参照

- ConfigPersistent.setCacheSize メソッド [Ultra Light J]94 ページ

getConnectionString メソッド [Android]

SetConnectionString メソッドで登録された接続文字列を取得します。

構文

```
String ConfigPersistent.getConnectionString()
```

戻り値

SetConnectionString メソッドで登録された接続文字列。

参照

- ConfigPersistent.setConnectionString メソッド [Android] [Ultra Light J]94 ページ

getCreationString メソッド [Android]

SetCreationString メソッドで登録された作成文字列を取得します。

構文

```
String ConfigPersistent.getCreationString()
```

戻り値

SetCreationString メソッドで登録された作成文字列。

参照

- ConfigPersistent.setCreationString メソッド [Android] [Ultra Light J]95 ページ

getDatabaseKey メソッド [Android]

SetDatabaseKey メソッドで登録されたデータベース暗号化キーを取得します。

構文

```
String ConfigPersistent.getDatabaseKey()
```

戻り値

SetDatabaseKey メソッドで登録されたデータベース暗号化キー。

参照

- ConfigPersistent.setDatabaseKey メソッド [Android] [Ultra Light J]95 ページ

getLazyLoadIndexes メソッド

インデックスの遅延ロードがオンになっているかどうかを確認します。

構文

```
boolean ConfigPersistent.getLazyLoadIndexes()
```

戻り値

遅延ロードがオンの場合は true、それ以外の場合は false。

参照

- ConfigPersistent.setLazyLoadIndexes メソッド [Ultra Light J]96 ページ

getRowScoreFlushSize メソッド [BlackBerry]

現在のロースコアのフラッシュサイズを返します。

構文

```
int ConfigPersistent.getRowScoreFlushSize()
```

戻り値

現在のロースコアのフラッシュサイズ。

参照

- ConfigPersistent.setRowScoreFlushSize メソッド [Ultra Light J]97 ページ

getRowScoreMaximum メソッド [BlackBerry]

現在のロースコアの最大サイズを返します。

構文

```
int ConfigPersistent.getRowScoreMaximum()
```

戻り値

現在のロースコアの最大サイズ。

参照

- ConfigPersistent.setRowScoreMaximum メソッド [Ultra Light J]98 ページ

getUserName メソッド [Android]

setUserName メソッドで設定されたユーザーの名前を取得します。

構文

```
String ConfigPersistent.getUserName()
```

戻り値

setUserName メソッドで設定されたユーザーの名前。

参照

- ConfigPersistent.setUserName メソッド [Android] [Ultra Light J]99 ページ

hasPersistentIndexes メソッド

インデックスが永続的かどうかを確認します。

構文

```
boolean ConfigPersistent.hasPersistentIndexes()
```

戻り値

インデックスが永続的な場合は true、それ以外の場合は false。

参照

- ConfigPersistent.setIndexPersistence メソッド [BlackBerry] [Ultra Light J]96 ページ

hasShadowPaging メソッド

シャドウページングがオンになっているかどうかを確認します。

構文

```
boolean ConfigPersistent.hasShadowPaging()
```

戻り値

シャドウページングがオンの場合は true、それ以外の場合は false。

参照

- ConfigPersistent.setShadowPaging メソッド [Ultra Light J]98 ページ

setAutocheckpoint メソッド (廃止予定)

自動チェックポイントをオンに設定します。

構文

```
ConfigPersistent ConfigPersistent.setAutocheckpoint(
    boolean auto_checkpoint
) throws ULjException
```

パラメーター

- **auto_checkpoint**　自動チェックポイントをオンに設定する場合は true。

パラメーター

- **auto_checkpoint**　自動チェックポイントをオンに設定する場合は true。

戻り値

自動チェックポイントが指定された ConfigPersistent。自動チェックポイントはオンに設定されます。

戻り値

自動チェックポイントが指定された ConfigPersistent。

setCacheSize メソッド

データベースのキャッシュサイズ (バイト単位) を設定します。

構文

```
ConfigPersistent ConfigPersistent.setCacheSize(
    int cache_size
) throws ULjException
```

パラメーター

- **cache_size** キャッシュサイズ。すべてのプラットフォームで、デフォルトのキャッシュサイズは 20480 (20 KB) です。

戻り値

キャッシュサイズが指定された ConfigPersistent オブジェクト。

備考

キャッシュサイズによって、ページキャッシュに常駐するデータベースのページ数が決まります。サイズを拡大すると、データベースページの読み込みと書き込みの回数が減りますが、キャッシュ内でページを検索する時間が長くなります。

参照

- ConfigPersistent.getCacheSize メソッド [Ultra Light J]90 ページ

setConnectionString メソッド [Android]

データベースの作成または接続に使用される接続文字列を設定します。

構文

```
void ConfigPersistent.setConnectionString(String connection_string)
```

パラメーター

- **connection_string** データベース接続および作成で使用される接続文字列。

備考

この設定に設定されている他の項目もすべて、データベースの作成または接続のために渡されます。

setCreationString メソッド [Android]

データベースを作成するために使用される作成文字列を設定します。

構文

```
void ConfigPersistent.setCreationString(String creation_string)
```

パラメーター

- **creation_string**　データベースの作成に使用される作成文字列。

備考

この設定に設定されている他の項目もすべて、データベースの作成のために渡されます。

setDatabaseKey メソッド [Android]

暗号化キーを設定します。

構文

```
void ConfigPersistent.setDatabaseKey(String encryption_key)
```

パラメーター

- **encryption_key**　暗号化に使用する文字列。

setEncryption メソッド [BlackBerry]

暗号化制御を設定します。

構文

```
ConfigPersistent ConfigPersistent.setEncryption(
    EncryptionControl control
) throws ULjException
```

パラメーター

- **control**　データベースの暗号化に使用する EncryptionControl オブジェクト。

戻り値

暗号化が指定された ConfigPersistent オブジェクト。

備考

このメソッドでは、EncryptionControl オブジェクトを使用して、ページの暗号化と復号化を設定します。

> **注意**
> 独自の暗号化メソッドを指定する必要があります。

setIndexPersistence メソッド [BlackBerry]

永続的なインデックスをオンに設定します。

構文

```
ConfigPersistent ConfigPersistent.setIndexPersistence(
    boolean store
) throws ULjException
```

パラメーター

- **store** インデックスを格納するには true に設定します。インデックスを初めて使用する前に構築するには false に設定します。

戻り値

インデックスの永続性が指定された ConfigPersistent オブジェクト。

備考

この設定は、データベースの作成時にのみ使用されます。データベースに使用するインデックスの永続性の方法を決定します。

既存のデータベースを開くときは、作成時の設定が使用され、その値を反映して設定が更新されます。

setLazyLoadIndexes メソッド

必要なときにインデックスをロードするか、起動時にすべてのインデックスを一度にロードするかを設定します。

構文

```
ConfigPersistent ConfigPersistent.setLazyLoadIndexes(
    boolean lazy_load
) throws ULjException
```

パラメーター

- **lazy_load** 必要なときにインデックスをロードする場合は true、起動時にすべてのインデックスを一度にロードする場合は false に設定します。

戻り値

インデックスのロードスキーマが指定された ConfigPersistent オブジェクト。

備考

このオプションを有効にすると、データベースの起動時間が短くなりますが、その後の操作が低速になる可能性があります。

参照

- ConfigPersistent.getLazyLoadIndexes メソッド [Ultra Light J]91 ページ
- ConfigPersistent.setRowScoreFlushSize メソッド [Ultra Light J]97 ページ

setRowScoreFlushSize メソッド

古いローの削除に使用されるスコアを指定することによって、ロー制限を有効にします。

構文

```
ConfigPersistent ConfigPersistent.setRowScoreFlushSize(
    int flushSize
) throws ULjException
```

パラメーター

- **flushSize**　削除するロー数の決定に使用されるロースコアの値。デフォルトは 0 (ロー数の制限なし) です。

戻り値

フラッシュサイズの値が指定された ConfigPersistent オブジェクト。

備考

ロースコアは、最近使用されたローをメモリ内に保持するために使用される、参照を計るものです。メモリ内の各ローには、ローのカラム数とそのタイプに基づいてスコアが割り当てられ、カラムで使用できる最大参照数が概算されます。

ほとんどのカラムのスコアは 1 ですが、varchar binary、long binary、UUID のスコアは 2、long varchar のスコアは 4 になります。

最大ロースコアのスレッショルドに達すると、フラッシュサイズを使用して、削除する古いロー数が決定されます。

大幅な中断を回避するため、フラッシュサイズ (ロースコアとして測定) を適切な値 (1000 未満) に保持することをおすすめします。

ロー制限が有効な状態でアクセスされるデータベースでは、常にインデックスが遅延ロードされます。

参照

- ConfigPersistent.setRowScoreMaximum メソッド [Ultra Light J]98 ページ
- ConfigPersistent.setLazyLoadIndexes メソッド [Ultra Light J]96 ページ

setRowScoreMaximum メソッド

メモリに保持する最大ロースコアのスレッショルドを設定します。

構文

```
ConfigPersistent ConfigPersistent.setRowScoreMaximum(
    int threshold
) throws ULjException
```

パラメーター

- **threshold**　スレッショルドの最大値。最大値は 200,000、デフォルト値は 50,000 です。

戻り値

最大スレッショルドの値が指定された ConfigPersistent オブジェクト。

備考

ロー制限が有効な状態でアクセスされるデータベースでは、常にインデックスが遅延ロードされます。

参照

- ConfigPersistent.setRowScoreFlushSize メソッド [Ultra Light J]97 ページ
- ConfigPersistent.setLazyLoadIndexes メソッド [Ultra Light J]96 ページ

setShadowPaging メソッド

シャドウページングをオンまたはオフに設定します。

構文

```
ConfigPersistent ConfigPersistent.setShadowPaging(
    boolean shadow
) throws ULjException
```

パラメーター

- **shadow**　シャドウページングをオンにする場合は true、それ以外の場合は false に設定します。

戻り値

ShadowPaging が指定された ConfigPersistent オブジェクト。

備考

デフォルトでは、永続的なデータベースストアのシャドウページングはオンに設定されます。

シャドウページング型は、永続性が最も強力であり、ほとんどのアプリケーションで使用されています。データベースの作成時に有効にすると、アプリケーションが予期せず終了した場合でも、最後のコミットまたはロールバック時の状態にデータベースを回復できます。

シャドウページングとは、永続ストアへの書き込みがすべて未使用のデータベースページに対して行われ、コミット操作が完了するまで永続的に格納されないことです。コミットされた変更は、アプリケーションが異常終了しても、永続的に保存されることが保証されます。

シャドウページングが false に設定されている場合、変更操作が行われてもコミットの完了前であればデータベースが破損する可能性があります。

シャドウページングで永続的にしなかった場合、データベース操作がより高速になり、返されるデータベースの結果が小さくなります。

データベース内のデータが重要ではないか、同期によってリカバリできる場合にのみ、シャドウページングなしでデータベースを処理するようにしてください。

参照

- ConfigPersistent.hasShadowPaging メソッド [Ultra Light J]93 ページ

setUserName メソッド [Android]

ユーザーの名前を設定します。

構文

```
void ConfigPersistent.setUserName(String user_name)
```

パラメーター

- **user_name**　ユーザーの名前。

備考

この名前は、接続文字列に UID= 句を使用してネイティブデータベースに接続、またはネイティブデータベースを作成する場合に使用されます。

setWriteAtEnd メソッド

停止時にのみ発生するデータベースの永続性を設定します。

構文

```
ConfigPersistent ConfigPersistent.setWriteAtEnd(
    boolean write_at_end
) throws ULjException
```

パラメーター

- **write_at_end**　データベースを停止するまでメモリに保持する場合、true に設定します。

戻り値

終了時書き込み型の永続性が指定どおりに設定された ConfigPersistent オブジェクト。

備考

このオプションを有効にすると、データベース操作が高速になりますが、アプリケーションが異常終了した場合はデータベースの変更内容がすべて失われます。データベースは基本的に、停止時に 1 回の大規模な書き込みを行い、以降に開くときに 1 回の大規模な読み込みを行うメモリ内データベースであるため、データベースのサイズはメモリ内に十分に収まるように小さくする必要があります。

データベース内のデータが重要ではないか、同期によってリカバリできる場合にのみ、終了時書き込み型の永続性を有効にしてデータベースを処理するようにしてください。

シャドウページングが有効になっている場合は、このオプションは無視されます。

参照

- ConfigPersistent.writeAtEnd メソッド [Ultra Light J]100 ページ
- ConfigPersistent.setShadowPaging メソッド [Ultra Light J]98 ページ

writeAtEnd メソッド

データベースで終了時書き込み型の永続性を使用するかどうかを確認します。

構文

```
boolean ConfigPersistent.writeAtEnd()
```

戻り値

データベースが停止するまでメモリに保持される場合は true、それ以外の場合は false。

参照

- ConfigPersistent.setWriteAtEnd メソッド [Ultra Light J]99 ページ

ConfigRecordStore インターフェイス (J2ME のみ)

J2ME レコードストアに保存される永続的なデータベース用の設定を確立します。

構文

```
public interface ConfigRecordStore
```

基本クラス

- ConfigPersistent インターフェイス [Ultra Light J]87 ページ

メンバー

継承されたメンバーを含む ConfigRecordStore インターフェイスのすべてのメンバー。

名前	説明
enableAesDBEncryption メソッド [Android]	データベースの AES 暗号化を有効にします。
enableObfuscation メソッド [Android]	データベースの難読化を有効にします。
getAutoCheckpoint メソッド (廃止予定)	自動チェックポイントがオンになっているかどうかを確認します。
getCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を返します。
getConnectionString メソッド [Android]	SetConnectionString メソッドで登録された接続文字列を取得します。
getCreationString メソッド [Android]	SetCreationString メソッドで登録された作成文字列を取得します。
getDatabaseKey メソッド [Android]	SetDatabaseKey メソッドで登録されたデータベース暗号化キーを取得します。
getDatabaseName メソッド	データベース名を返します。
getLazyLoadIndexes メソッド	インデックスの遅延ロードがオンになっているかどうかを確認します。
getPageSize メソッド	データベースのページサイズ (バイト単位) を返します。
getRowScoreFlushSize メソッド [BlackBerry]	現在のロースコアのフラッシュサイズを返します。
getRowScoreMaximum メソッド [BlackBerry]	現在のロースコアの最大サイズを返します。
getUserName メソッド [Android]	setUserName メソッドで設定されたユーザーの名前を取得します。
hasPersistentIndexes メソッド	インデックスが永続的かどうかを確認します。
hasShadowPaging メソッド	シャドウページングがオンになっているかどうかを確認します。

名前	説明
setAutocheckpoint メソッド (廃止予定)	自動チェックポイントをオンに設定します。
setCacheSize メソッド	データベースのキャッシュサイズ (バイト単位) を設定します。
setConnectionString メソッド [Android]	データベースの作成または接続に使用される接続文字列を設定します。
setCreationString メソッド [Android]	データベースを作成するために使用される作成文字列を設定します。
setDatabaseKey メソッド [Android]	暗号化キーを設定します。
setDatabaseName メソッド	データベース名を設定します。
setEncryption メソッド [BlackBerry]	暗号化制御を設定します。
setIndexPersistence メソッド [BlackBerry]	永続的なインデックスをオンに設定します。
setLazyLoadIndexes メソッド	必要なときにインデックスをロードするか、起動時にすべてのインデックスを一度にロードするかを設定します。
setPageSize メソッド	データベースのページサイズを設定します。
setPassword メソッド	データベースのパスワードを設定します。
setRowScoreFlushSize メソッド	古いローの削除に使用されるスコアを指定することによって、ロー制限を有効にします。
setRowScoreMaximum メソッド	メモリに保持する最大ロースコアのスレッショルドを設定します。
setShadowPaging メソッド	シャドウページングをオンまたはオフに設定します。
setUserName メソッド [Android]	ユーザーの名前を設定します。
setWriteAtEnd メソッド	停止時にのみ発生するデータベースの永続性を設定します。
writeAtEnd メソッド	データベースで終了時書き込み型の永続性を使用するかどうかを確認します。

Configuration インターフェイス

データベース用の Configuration オブジェクトを確立します。

構文

```
public interface Configuration
```

派生クラス

- ConfigNonPersistent インターフェイス [BlackBerry] [Ultra Light J]84 ページ
- ConfigPersistent インターフェイス [Ultra Light J]87 ページ

メンバー

継承されたメンバーを含む Configuration インターフェイスのすべてのメンバー。

名前	説明
getDatabaseName メソッド	データベース名を返します。
getPageSize メソッド	データベースのページサイズ (バイト単位) を返します。
setDatabaseName メソッド	データベース名を設定します。
setPageSize メソッド	データベースのページサイズを設定します。
setPassword メソッド	データベースのパスワードを設定します。

備考

一部の属性は、データベースの作成時にのみ使用されます。その他の属性は、データベースへの最初の接続に適用されます。データベースの作成後、またはデータベースへの接続後に設定された属性は無視されます。

getDatabaseName メソッド

データベース名を返します。

構文

```
String Configuration.getDatabaseName()
```

戻り値

データベースの名前。

getPageSize メソッド

データベースのページサイズ (バイト単位) を返します。

構文

```
int Configuration.getPageSize()
```

戻り値

ページサイズ。

setDatabaseName メソッド

データベース名を設定します。

構文

```
Configuration Configuration.setDatabaseName(
    String db_name
) throws ULjException
```

パラメーター

- **db_name** データベースの名前。

戻り値

データベース名が指定された Configuration オブジェクト。

setPageSize メソッド

データベースのページサイズを設定します。

構文

```
Configuration Configuration.setPageSize(
    int page_size
) throws ULjException
```

パラメーター

- **page_size** ページサイズ (バイト単位)

戻り値

ページサイズが指定された Configuration オブジェクト。

備考

ページサイズの設定を使用して、永続的なデータベースに格納されるローの最大サイズが決定されます。また、このページサイズによって、インデックスページのサイズが確立され、各ページの子の数が決まります。

既存のデータベースを使用する場合は、データベースの作成時のページサイズにすでに設定されています。このメソッドを使用して、既存のデータベースのページサイズをリセットすることはできません。

Android スマートフォンの場合、ページサイズは 1024、2048、4096、8192、または 16384 バイトに設定できます。デフォルトは 4096 バイトです。

BlackBerry スマートフォンの場合、ページサイズは 256 ～ 16384 バイトの範囲内で設定できます。デフォルトは 1024 バイトです。ページサイズは、常に 32 の倍数に調整されます。

setPassword メソッド

データベースのパスワードを設定します。

構文

```
Configuration Configuration.setPassword(
    String password
) throws ULjException
```

パラメーター

- **password**　新しいデータベースのパスワード、または既存のデータベースにアクセスするためのパスワード。

戻り値

データベースパスワードが設定された Configuration オブジェクト。

備考

このパスワードを使用してデータベースへのアクセスが許可されます。このパスワードは、データベースの作成時に指定されたパスワードと一致する必要があります。デフォルトは "dba" です。

Connection インターフェイス

データベース接続を表します。データベース操作を開始するには接続が必要です。

構文

```
public interface Connection
```

メンバー

継承されたメンバーを含む Connection インターフェイスのすべてのメンバー。

名前	説明
checkpoint メソッド	データベースの変更内容にチェックポイントを設定します。
commit メソッド	データベースの変更内容をコミットします。
createDecimalNumber メソッド	新しい DecimalNumber オブジェクトを作成します。
createSyncParms メソッド	同期パラメーターセットを作成します。
createUUIDValue メソッド	UUID 値を作成します。
dropDatabase メソッド	データベースを削除します。
emergencyShutdown メソッド [BlackBerry]	接続しているデータベースを緊急停止します。
getDatabaseId メソッド [BlackBerry]	データベース ID の値を返します。
getDatabaseInfo メソッド	データベースプロパティに関する情報を含む DatabaseInfo オブジェクトを返します。
getDatabaseProperty メソッド	データベースプロパティを返します。
getLastDownloadTime メソッド	指定されたパブリケーションの最後のダウンロードの時刻を返します。
getLastIdentity メソッド	DEFAULT AUTOINCREMENT カラムまたは DEFAULT GLOBAL AUTOINCREMENT カラムに挿入された最新の値が取得されます。最新の INSERT トランザクションが、このようなカラムがないテーブルに対して行われた場合は 0 になります。
getOption メソッド [BlackBerry]	データベースオプションを返します。
getState メソッド [BlackBerry]	接続のステータスを返します。
getSyncObserver メソッド	この Connection オブジェクトに対して現在登録されている SyncObserver オブジェクトを返します。
getSyncResult メソッド	前回の SYNCHRONIZE SQL 文の結果を返します。

名前	説明
isSynchronizationDeleteDisabled メソッド [BlackBerry]	削除の同期が無効になっているかどうかを確認します。
prepareStatement メソッド	実行する文を準備します。
release メソッド	この接続を解放します。
resetLastDownloadTime メソッド	指定されたパブリケーションのダウンロード時刻をリセットします。
rollback メソッド	データベースへの変更を取り消すロールバックをコミットします。
setDatabaseId メソッド	グローバルオートインクリメントのデータベース ID を設定します。
setOption メソッド	データベースオプションを設定します。
setSyncObserver メソッド	この接続で同期の進行状況をモニターする SyncObserver オブジェクトを設定します。
synchronize メソッド	データベースを Mobile Link サーバーと同期させます。
CONNECTED 変数	接続されている状態を示します。
NOT_CONNECTED 変数	接続されていない状態を示します。
OPTION_BLOB_FILE_BASE_DIR 変数 [BlackBerry]	データベースオプション：blob ファイルのベースディレクトリ。
OPTION_DATABASE_ID 変数 [BlackBerry]	データベースオプション：データベース ID。
OPTION_DATE_FORMAT 変数 [BlackBerry]	データベースオプション：日付形式。
OPTION_DATE_ORDER 変数 [BlackBerry]	データベースオプション：日付順。
OPTION_ML_REMOTE_ID 変数 [BlackBerry]	データベースオプション：ML リモート ID。
OPTION_NEAREST_CENTURY 変数 [BlackBerry]	データベースオプション：基準年。
OPTION_PRECISION 変数 [BlackBerry]	データベースオプション：精度。
OPTION_SCALE 変数 [BlackBerry]	データベースオプション：位取り。
OPTION_TIME_FORMAT 変数 [BlackBerry]	データベースオプション：時間形式。

名前	説明
OPTION_TIMESTAMP_FORMAT 変数 [BlackBerry]	データベースオプション：タイムスタンプ形式。
OPTION_TIMESTAMP_INCREMENT 変数 [BlackBerry]	データベースオプション：タイムスタンプインクリメント。
OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT 変数 [BlackBerry]	データベースオプション：タイムゾーン形式のタイムスタンプ。
PROPERTY_DATABASE_NAME 変数 [BlackBerry]	データベースプロパティ：データベース名。
PROPERTY_PAGE_SIZE 変数 [BlackBerry]	データベースプロパティ：ページサイズ。
SYNC_ALL 変数	データベース内の全テーブルの同期を要求するために使用するパブリケーションのリストです。どのパブリケーションにも含まれないテーブルも含まれます。
SYNC_ALL_DB_PUB_NAME 変数	SYNC_ALL_DB パブリケーションの予約名です。
SYNC_ALL_PUBS 変数	データベース内の全パブリケーションの同期を要求するために使用するパブリケーションのリストです。

備考

接続は、DatabaseManager クラスの connect メソッドまたは createDatabase メソッドを使用して取得します。接続が不要になったら release メソッドを使用します。データベースのすべての接続を解放したら、データベースは終了します。

Connection オブジェクトには、次の機能があります。

●新しいスキーマの作成 (テーブル、インデックス、パブリケーション)

●新しい値とドメインオブジェクトの作成

●データベースへの変更の永続的なコミット

●実行する SQL 文の準備

●コミットされていないデータベースへの変更のロールバック

次の例は、単純なデータベースのために作成された Connection オブジェクト conn を使用して、このデータベースのスキーマを作成する方法を示しています。データベースにはテーブル T1 と T2 があります。T1 には num という整数のプライマリキーカラムが 1 つあります。T2 には num という整数のプライマリキーカラムと quantity という整数カラムがあります。T2 の quantity には追加インデックスがあります。T1 は PubA というパブリケーションに含まれます。


```
// Assumes a valid connection object, conn, for the current database.

PreparedStatement ps;

ps = conn.prepareStatement( "CREATE TABLE T1 ( num INT NOT NULL PRIMARY KEY )" );
ps.execute();
ps.close();

ps = conn.prepareStatement( "CREATE TABLE T2 ( num INT NOT NULL PRIMARY KEY, quantity
INT)" );
ps.execute();
ps.close();

ps = conn.prepareStatement( "CREATE INDEX index1 ON T2( quantity )" );
ps.execute();
ps.close();

ps = conn.prepareStatement( "CREATE Publication PubA ( Table T1 )" );
ps.execute();
ps.close();
```

参照

- DatabaseManager クラス [Ultra Light J]130 ページ
- DatabaseManager.createDatabase メソッド [Ultra Light J]136 ページ
- DatabaseManager.connect メソッド [Ultra Light J]132 ページ
- Connection.release メソッド [Ultra Light J]118 ページ

checkpoint メソッド

データベースの変更内容にチェックポイントを設定します。

構文

```
void Connection.checkpoint() throws ULjException
```

備考

この関数を呼び出すと、コミットされたすべてのトランザクションが、永続的なデータベースに適用されます。データベースの変更に対するチェックポイントを実行します。

この関数を呼び出すと、コミットされたすべてのトランザクションが、永続的なデータベースに適用されます。

commit メソッド

データベースの変更内容をコミットします。

構文

```
void Connection.commit() throws ULjException
```

備考

このメソッドを呼び出すと、最後のコミット後またはロールバック後に行われたテーブルデータへの変更がすべて永続的になります。

createDecimalNumber メソッド

新しい DecimalNumber オブジェクトを作成します。

オーバーロードリスト

名前	説明
createDecimalNumber(int, int) メソッド	DecimalNumber オブジェクトを作成します。
createDecimalNumber(int, int, String) メソッド	DecimalNumber オブジェクトを作成します。

createDecimalNumber(int, int) メソッド

DecimalNumber オブジェクトを作成します。

構文

```
DecimalNumber Connection.createDecimalNumber(
    int precision,
    int scale
) throws ULjException
```

パラメーター

- **precision**　数値の桁数。
- **scale**　数値の小数点以下の桁数。

戻り値

指定された型の DecimalNumber オブジェクト。

参照

- DecimalNumber インターフェイス [Ultra Light J]139 ページ

createDecimalNumber(int, int, String) メソッド

DecimalNumber オブジェクトを作成します。

構文

```
DecimalNumber Connection.createDecimalNumber(
    int precision,
    int scale,
```



```
    String value
) throws ULjException
```

パラメーター

- **precision**　数値の桁数。
- **scale**　数値の小数点以下の桁数。
- **value**　設定する値。

戻り値

指定された型の DecimalNumber オブジェクト。

参照

- DecimalNumber インターフェイス [Ultra Light J]139 ページ

createSyncParms メソッド

同期パラメーターセットを作成します。

オーバーロードリスト

名前	説明
createSyncParms(int, String, String) メソッド	HTTP 同期用の同期パラメーターセットを作成します。
createSyncParms(String, String) メソッド	HTTP 同期用の同期パラメーターセットを作成します。

createSyncParms(int, String, String) メソッド

HTTP 同期用の同期パラメーターセットを作成します。

構文

```
SyncParms Connection.createSyncParms(
    int streamType,
    String userName,
    String version
) throws ULjException
```

パラメーター

- **streamType**　同期ストリームのタイプの指定に使用する、SyncParms クラス内で定義されている定数の 1 つ。
- **userName**　Mobile Link ユーザー名。

- **version** Mobile Link スクリプトのバージョン。

戻り値

SyncParms オブジェクト。

参照

- Connection.createSyncParms メソッド [Ultra Light J]111 ページ
- SyncParms.HTTP_STREAM 変数 [Ultra Light J]257 ページ
- SyncParms.HTTPS_STREAM 変数 [Ultra Light J]257 ページ

createSyncParms(String, String) メソッド

HTTP 同期用の同期パラメーターセットを作成します。

構文

```
SyncParms Connection.createSyncParms(
    String userName,
    String version
) throws ULjException
```

パラメーター

- **userName** このクライアントデータベース用のユニークな Mobile Link ユーザー名。
- **version** Mobile Link スクリプトのバージョン。

戻り値

SyncParms オブジェクト。

参照

- Connection.createSyncParms メソッド [Ultra Light J]111 ページ
- SyncParms.setUserName メソッド [Ultra Light J]256 ページ

createUUIDValue メソッド

UUID 値を作成します。

構文

```
UUIDValue Connection.createUUIDValue() throws ULjException
```

戻り値

ドメインの UUIDValue インスタンス。

dropDatabase メソッド

データベースを削除します。

構文

```
void Connection.dropDatabase() throws ULjException
```

備考

接続によって参照されているデータベースを消去し、接続を解放します。削除するデータベースへの有効な接続が、この接続だけである必要があります。

emergencyShutdown メソッド [BlackBerry]

接続しているデータベースを緊急停止します。

構文

```
void Connection.emergencyShutdown() throws ULjException
```

備考

このメソッドは、重大なエラーが発生したときにのみ呼び出してください。物理的なハードウェアまたはデータが壊れた場合にのみ使用してください。

このメソッドは、開いている接続をすべて閉じ、接続しているデータベースを停止します。

getDatabaseId メソッド [BlackBerry]

データベース ID の値を返します。

構文

```
int Connection.getDatabaseId() throws ULjException
```

戻り値

データベース ID。

例外

- **ULjException クラス**　データベース ID が設定されていない場合。

参照

- Connection.getLastIdentity メソッド [Ultra Light J]115 ページ

getDatabaseInfo メソッド

データベースプロパティに関する情報を含む DatabaseInfo オブジェクトを返します。

構文

```
DatabaseInfo Connection.getDatabaseInfo() throws ULjException
```

戻り値

DatabaseInfo オブジェクト。

getDatabaseProperty メソッド

データベースプロパティを返します。

構文

```
String Connection.getDatabaseProperty(String name) throws ULjException
```

パラメーター

- **name** データベースプロパティの名前。Android デバイスの場合、このパラメーターにサポート対象の任意の Ultra Light データベースプロパティ名を設定できます。BlackBerry デバイスの場合、このパラメーターに、接続インターフェイス内の **PROPERTY_** プレフィクスが付いている任意の定数を設定できます。

戻り値

指定された名前に対応するプロパティの値。

参照

- Connection.PROPERTY_DATABASE_NAME 変数 [BlackBerry] [Ultra Light J]125 ページ
- Connection.PROPERTY_PAGE_SIZE 変数 [BlackBerry] [Ultra Light J]125 ページ

getLastDownloadTime メソッド

指定されたパブリケーションの最後のダウンロードの時刻を返します。

構文

```
Date Connection.getLastDownloadTime(String pub_name) throws ULjException
```

パラメーター

- **pub_name** チェックするパブリケーションの名前。このパラメーターは、1 つのパブリケーションを参照するか、データベース全体を最後にダウンロードしたときの特殊な Connection.SYNC_ALL_DB_PUB_NAME パブリケーションである必要があります。

戻り値

最後のダウンロードのタイムスタンプ。

参照

- Connection.SYNC_ALL_DB_PUB_NAME 変数 [Ultra Light J]126 ページ
- Connection.resetLastDownloadTime メソッド [Ultra Light J]118 ページ

getLastIdentity メソッド

DEFAULT AUTOINCREMENT カラムまたは DEFAULT GLOBAL AUTOINCREMENT カラムに挿入された最新の値が取得されます。最新の INSERT トランザクションが、このようなカラムがないテーブルに対して行われた場合は 0 になります。

構文

```
long Connection.getLastIdentity()
```

戻り値

直前に使用した identity の値。

備考

テーブルに複数の (GLOBAL) AUTOINCREMENT 型のカラムが含まれている場合、この値が属しているカラムは特定されません。

getOption メソッド [BlackBerry]

データベースオプションを返します。

構文

```
String Connection.getOption(String option_name) throws ULjException
```

パラメーター

- **option_name**　取得するオプションの名前。
- **option_name**　取得するオプションの名前。このパラメーターには、接続インターフェイス内の **OPTION_** プレフィクスが付いている任意の定数を設定できます。

戻り値

データベースオプションの値。

備考

データベースオプションはデータベース内に格納され、オプションの設定後にデータベースに接続したときにも取得できます。

データベースの作成時に、一連の必須オプションが作成されます。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

getState メソッド [BlackBerry]

接続のステータスを返します。

構文

```
byte Connection.getState() throws ULjException
```

戻り値

接続のステータスを示すバイト数。

備考

次の例は、接続状態を確認し、接続を解放する方法を示しています。

```
if( _conn.getState() == Connection.CONNECTED ){
   _conn.release();
}
```

参照

- Connection インターフェイス [Ultra Light J]105 ページ

getSyncObserver メソッド

この Connection オブジェクトに対して現在登録されている SyncObserver オブジェクトを返します。

構文

```
SyncObserver Connection.getSyncObserver()
```

戻り値

SyncObserver、または observer が存在しない場合は NULL。

参照

- Connection.setSyncObserver メソッド [Ultra Light J]120 ページ

getSyncResult メソッド

前回の SYNCHRONIZE SQL 文の結果を返します。

構文

```
SyncResult Connection.getSyncResult()
```

戻り値

前回の SYNCHRONIZE SQL 文の結果を示す SyncResult オブジェクト。

備考

次に、前回の SYNCHRONIZE SQL 文の結果を取得する例を示します。

```
PreparedStatement ps = conn.prepareStatement("SYNCHRONIZE PROFILE myprofile");
ps.execute();
ps.close();
SyncResult result = conn.getSyncResult();
display(
   "*** Synchronized *** sent=" + result.getSentRowCount()
   + ", received=" + result.getReceivedRowCount()
);
```

注意
このメソッドでは、前回の Connection.synchronize メソッド呼び出しの結果は返されません。前回の Connection.synchronize(SyncParms) メソッド呼び出しの SyncResult オブジェクトを取得するには、渡された SyncParms オブジェクトで getSyncResult メソッドを使用します。

参照

- SyncResult クラス [Ultra Light J]258 ページ
- SyncParms.getSyncResult メソッド [Ultra Light J]247 ページ

isSynchronizationDeleteDisabled メソッド [BlackBerry]

削除の同期が無効になっているかどうかを確認します。

構文

```
boolean Connection.isSynchronizationDeleteDisabled()
```

戻り値

削除の同期が無効になっている場合にのみ、True。

prepareStatement メソッド

実行する文を準備します。

構文

```
PreparedStatement Connection.prepareStatement(
    String sql
) throws ULjException
```

パラメーター

- **sql**　準備する SQL 文。

戻り値

PreparedStatement オブジェクト。

参照

- PreparedStatement インターフェイス [Ultra Light J]178 ページ

release メソッド

この接続を解放します。

構文

```
void Connection.release() throws ULjException
```

備考

一度解放した接続は、データベースへのアクセスに使用できなくなります。

コミットされていないトランザクションがある接続を解放しようとするとエラーが発生します。

resetLastDownloadTime メソッド

指定されたパブリケーションのダウンロード時刻をリセットします。

構文

```
void Connection.resetLastDownloadTime(
    String pub_name
) throws ULjException
```

パラメーター

- **pub_name**　チェックするパブリケーションの名前。

備考

データベース全体が同期されたダウンロード時刻をリセットするには、特殊なパブリケーション Connection.SYNC_ALL_DB_PUB_NAME を使用します。

このメソッドを使用するには、現在の接続に、コミットされていないトランザクションがないことが必要です。

rollback メソッド

データベースへの変更を取り消すロールバックをコミットします。

構文

```
void Connection.rollback() throws ULjException
```

備考

このメソッドを呼び出すと、この Connection オブジェクトで、最後のコミット後またはロールバック後に行われたテーブルデータへの変更がすべて取り消されます。

setDatabaseId メソッド

グローバルオートインクリメントのデータベース ID を設定します。

構文

```
void Connection.setDatabaseId(int id) throws ULjException
```

パラメーター

- **id** データベース ID。

備考

データベース ID にはデフォルト値はありません。

データベース ID を明示的に設定しないかぎり、INSERT 時に GLOBAL AUTOINCREMENT カラムに NULL 値が挿入されます。

参照

- Connection.getDatabaseId メソッド [BlackBerry] [Ultra Light J]113 ページ

setOption メソッド

データベースオプションを設定します。

構文

```
void Connection.setOption(
    String option_name,
    String option_value
) throws ULjException
```

パラメーター

- **option_name** 設定するオプションの名前。Android デバイスの場合、このパラメーターにサポート対象の任意の Ultra Light データベースオプション名を設定できます。BlackBerry デ

バイスの場合、このパラメーターに、接続インターフェイス内の **OPTION_** プレフィクスが付いている任意の定数を設定できます。

- **option_value**　オプションの新しい値。

備考

オプションが現在データベースに格納されていない場合は作成されます。

この接続にコミットされていないトランザクションがあるときに、このメソッドを呼び出すことはできません。

参照

- Connection.getOption メソッド [BlackBerry] [Ultra Light J]115 ページ
- Connection.OPTION_BLOB_FILE_BASE_DIR 変数 [BlackBerry] [Ultra Light J]121 ページ
- Connection.OPTION_DATABASE_ID 変数 [BlackBerry] [Ultra Light J]122 ページ
- Connection.OPTION_DATE_FORMAT 変数 [BlackBerry] [Ultra Light J]122 ページ
- Connection.OPTION_DATE_ORDER 変数 [BlackBerry] [Ultra Light J]122 ページ
- Connection.OPTION_ML_REMOTE_ID 変数 [BlackBerry] [Ultra Light J]123 ページ
- Connection.OPTION_NEAREST_CENTURY 変数 [BlackBerry] [Ultra Light J]123 ページ
- Connection.OPTION_PRECISION 変数 [BlackBerry] [Ultra Light J]123 ページ
- Connection.OPTION_SCALE 数 [BlackBerry] [Ultra Light J]124 ページ
- Connection.OPTION_TIME_FORMAT 変数 [BlackBerry] [Ultra Light J]124 ページ
- Connection.OPTION_TIMESTAMP_FORMAT 変数 [BlackBerry] [Ultra Light J]124 ページ
- Connection.OPTION_TIMESTAMP_INCREMENT 変数 [BlackBerry] [Ultra Light J]125 ページ
- Connection.OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT 変数 [BlackBerry] [Ultra Light J]125 ページ

setSyncObserver メソッド

この接続で同期の進行状況をモニターする SyncObserver オブジェクトを設定します。

構文

```
void Connection.setSyncObserver(SyncObserver so)
```

パラメーター

- **so**　SyncObserver オブジェクト、または現在登録されている SyncObserver オブジェクトを削除する場合はヌル。

備考

この SyncObserver オブジェクトは、以降の SYNCHRONIZE SQL 文に使用されます。

デフォルトは NULL で、これは observer なしを示します。

参照

- SyncObserver インターフェイス [Ultra Light J]235 ページ

synchronize メソッド

データベースを Mobile Link サーバーと同期させます。

構文

```
void Connection.synchronize(SyncParms config) throws ULjException
```

パラメーター

- **config** 同期に使用するパラメーターが含まれている SyncParms オブジェクト。

備考

データベースにダウンロードが適用されるときにチェックポイントが設定されます。

参照

- SyncParms クラス [Ultra Light J]241 ページ

CONNECTED 変数

接続されている状態を示します。

構文

```
final byte Connection.CONNECTED
```

NOT_CONNECTED 変数

接続されていない状態を示します。

構文

```
final byte Connection.NOT_CONNECTED
```

OPTION_BLOB_FILE_BASE_DIR 変数 [BlackBerry]

データベースオプション：blob ファイルのベースディレクトリ。

構文

```
final String Connection.OPTION_BLOB_FILE_BASE_DIR
```

備考

BlackBerry デバイスの場合、デフォルト値は "file:///SDCard/"、BlackBerry デバイス以外の場合は "" です。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

OPTION_DATABASE_ID 変数 [BlackBerry]

データベースオプション：データベース ID。

構文

```
final String Connection.OPTION_DATABASE_ID
```

備考

デフォルト値は指定されません。明示的に割り当てる必要があります。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

OPTION_DATE_FORMAT 変数 [BlackBerry]

データベースオプション：日付形式。

構文

```
final String Connection.OPTION_DATE_FORMAT
```

備考

デフォルト値は "YYYY-MM-DD" です。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

OPTION_DATE_ORDER 変数 [BlackBerry]

データベースオプション：日付順。

構文

```
final String Connection.OPTION_DATE_ORDER
```

備考

デフォルト値は "YMD" です。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

OPTION_ML_REMOTE_ID 変数 [BlackBerry]

データベースオプション：ML リモート ID。

構文

```
final String Connection.OPTION_ML_REMOTE_ID
```

備考

デフォルト値は指定されません。最初の Mobile Link 同期の後で値が設定されます。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

OPTION_NEAREST_CENTURY 変数 [BlackBerry]

データベースオプション：基準年。

構文

```
final String Connection.OPTION_NEAREST_CENTURY
```

備考

デフォルト値は "50" です。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

OPTION_PRECISION 変数 [BlackBerry]

データベースオプション：精度。

構文

```
final String Connection.OPTION_PRECISION
```

備考

デフォルト値は "30" です。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

OPTION_SCALE 変数 [BlackBerry]

データベースオプション：位取り。

構文

```
final String Connection.OPTION_SCALE
```

備考

デフォルト値は "6" です。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

OPTION_TIME_FORMAT 変数 [BlackBerry]

データベースオプション：時間形式。

構文

```
final String Connection.OPTION_TIME_FORMAT
```

備考

デフォルト値は "HH:NN:SS.SSS" です。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

OPTION_TIMESTAMP_FORMAT 変数 [BlackBerry]

データベースオプション：タイムスタンプ形式。

構文

```
final String Connection.OPTION_TIMESTAMP_FORMAT
```

備考

デフォルト値は "YYYY-MM-DD HH:NN:SS.SSS" です。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

OPTION_TIMESTAMP_INCREMENT 変数 [BlackBerry]

データベースオプション：タイムスタンプインクリメント。

構文

```
final String Connection.OPTION_TIMESTAMP_INCREMENT
```

備考

デフォルト値は "1" です。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT 変数 [BlackBerry]

データベースオプション：タイムゾーン形式のタイムスタンプ。

構文

```
final String Connection.OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT
```

備考

デフォルト値は "YYYY-MM-DD HH:NN:SS.SSS+HH:NN" です。

参照

- Connection.setOption メソッド [Ultra Light J]119 ページ

PROPERTY_DATABASE_NAME 変数 [BlackBerry]

データベースプロパティ：データベース名。

構文

```
final String Connection.PROPERTY_DATABASE_NAME
```

参照

- Connection.getDatabaseProperty メソッド [Ultra Light J]114 ページ

PROPERTY_PAGE_SIZE 変数 [BlackBerry]

データベースプロパティ：ページサイズ。

構文

```
final String Connection.PROPERTY_PAGE_SIZE
```

参照

- Connection.getDatabaseProperty メソッド [Ultra Light J]114 ページ

SYNC_ALL 変数

データベース内の全テーブルの同期を要求するために使用するパブリケーションのリストです。どのパブリケーションにも含まれないテーブルも含まれます。

構文

```
final String Connection.SYNC_ALL
```

備考

NoSync と指定されているテーブルは同期されません。

この定数は、NULL 参照または空の文字列と同じです。

SYNC_ALL_DB_PUB_NAME 変数

SYNC_ALL_DB パブリケーションの予約名です。

構文

```
final String Connection.SYNC_ALL_DB_PUB_NAME
```

参照

- Connection.getLastDownloadTime メソッド [Ultra Light J]114 ページ
- Connection.resetLastDownloadTime メソッド [Ultra Light J]118 ページ

SYNC_ALL_PUBS 変数

データベース内の全パブリケーションの同期を要求するために使用するパブリケーションのリストです。

構文

```
final String Connection.SYNC_ALL_PUBS
```

備考

NoSync と指定されているテーブルは同期されません。

DatabaseInfo インターフェイス

Connection オブジェクトに関連付けられ、データベース情報を公開するメソッドを提供します。

構文

```
public interface DatabaseInfo
```

メンバー

継承されたメンバーを含む DatabaseInfo インターフェイスのすべてのメンバー。

名前	説明
getCommitCount メソッド [BlackBerry]	データベースに対して実行されたコミット操作の合計数を返します。
getDbFormat メソッド [BlackBerry]	データベースのバージョン番号を返します。
getDbSize メソッド [BlackBerry]	データベースサイズを返します。
getLogSize メソッド [BlackBerry]	トランザクションログの合計サイズ (バイト単位) を返します。
getNumberRowsToUpload メソッド	アップロードを待機しているローの数を返します。
getPageReads メソッド	ページの読み込み数を返します。
getPageSize メソッド	データベースのページサイズ (バイト単位) を返します。
getPageWrites メソッド	ページの書き込み数を返します。
getRelease メソッド	ソフトウェアのリリース番号を返します。

備考

このインターフェイスは、Connection オブジェクトの getDatabaseInfo メソッドを使用して呼び出します。

参照

- Connection インターフェイス [Ultra Light J]105 ページ
- Connection.getDatabaseInfo メソッド [Ultra Light J]114 ページ

getCommitCount メソッド [BlackBerry]

データベースに対して実行されたコミット操作の合計数を返します。

構文

```
int DatabaseInfo.getCommitCount()
```

戻り値

コミット操作の合計数。

getDbFormat メソッド [BlackBerry]

データベースのバージョン番号を返します。

構文

```
int DatabaseInfo.getDbFormat()
```

戻り値

バージョン番号。

getDbSize メソッド [BlackBerry]

データベースサイズを返します。

構文

```
int DatabaseInfo.getDbSize()
```

戻り値

データベースが永続的ではなく終了時書き込みが設定されている場合は (-1) が返され、それ例外の場合は永続ストアの現在のサイズが返されます。

getLogSize メソッド [BlackBerry]

トランザクションログの合計サイズ (バイト単位) を返します。

構文

```
int DatabaseInfo.getLogSize()
```

戻り値

トランザクションログのサイズ。

getNumberRowsToUpload メソッド

アップロードを待機しているローの数を返します。

構文

```
int DatabaseInfo.getNumberRowsToUpload()
```

戻り値

ローの数。

getPageReads メソッド

ページの読み込み数を返します。

構文

```
int DatabaseInfo.getPageReads()
```

戻り値

ページ読み込み数。

getPageSize メソッド

データベースのページサイズ (バイト単位) を返します。

構文

```
int DatabaseInfo.getPageSize()
```

戻り値

ページサイズ。

getPageWrites メソッド

ページの書き込み数を返します。

構文

```
int DatabaseInfo.getPageWrites()
```

戻り値

ページ書き込み数。

getRelease メソッド

ソフトウェアのリリース番号を返します。

構文

```
String DatabaseInfo.getRelease()
```

戻り値

リリース番号。

備考

たとえば、ソフトウェアリリースの値 "12.0.0.1234" は、リリースが 12.0.0、ビルド番号が 1234 であることを示します。

DatabaseManager クラス

基本設定を取得したり、新しいデータベースを作成したり、既存のデータベースに接続したりするための静的メソッドを提供します。

構文

```
public class DatabaseManager
```

メンバー

継承されたメンバーを含む DatabaseManager クラスのすべてのメンバー。

名前	説明
connect メソッド	設定に基づいて既存のデータベースに接続します。
createConfigurationFileME メソッド [BlackBerry]	J2ME デバイスのファイルシステム上のファイルから物理データベースストア用の Configuration オブジェクトを作成し、ConfigFileME オブジェクトを返します。
createConfigurationNonPersistent メソッド [BlackBerry]	非永続的なデータベースストア用の Configuration オブジェクトを作成し、ConfigNonPersist オブジェクトを返します。
createConfigurationObjectStore メソッド [BlackBerry]	RIM オブジェクトストア用の Configuration オブジェクトを作成し、ConfigObjectStore オブジェクトを返します。
createConfigurationRecordStore メソッド [J2ME]	物理データベースストアとするレコードストア用の Configuration オブジェクトを作成し、ConfigRecordStore オブジェクトを返します。
createDatabase メソッド	設定セットに基づいて新しいデータベースを作成し、データベースに接続します。

名前	説明
createFileTransfer メソッド	Mobile Link との間でファイルを転送するための FileTransfer オブジェクトを作成します。
createObjectStoreTransfer メソッド [BlackBerry]	Mobile Link との間で Ultra Light Java Edition データベースを転送し、RIM オブジェクトストアに格納するための FileTransfer オブジェクトを作成します。
createSISHTTPListener メソッド [BlackBerry]	サーバー起動同期用の SISListener オブジェクトを作成します。
release メソッド	DatabaseManager オブジェクトを閉じてすべての接続を解放し、すべてのデータベースを停止します。
setErrorLanguage メソッド	エラーメッセージに使用する言語を設定します。

備考

次の例は、J2SE プラットフォーム上で既存のデータベースを開く方法、または存在しない場合に新規のデータベースを作成する方法を示しています。

```
Connection conn;
ConfigFile config = DatabaseManager.createConfigurationFile(
        "test.ulj"
    );
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```

次の例は、BlackBerry デバイス上で既存のデータベースを開く方法、または存在しない場合に新規のデータベースを作成する方法を示しています。

```
Connection conn;
ConfigObjectStore config = DatabaseManager.createConfigurationObjectStore(
        "test.ulj"
    );
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```

次の例は、BlackBerry メディアカード上で既存のデータベースを開く方法、または存在しない場合に新規のデータベースを作成する方法を示しています。

```
Connection conn;
ConfigFileME config = DatabaseManager.createConfigurationFileME(
```

```
        "file:///SDCard/ulj/test.ulj"
    );
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```

次の例は、Android デバイス上で既存のデータベースを開く方法、または存在しない場合に新規のデータベースを作成する方法を示しています。

```
// Android Sample
Connection conn;
ConfigFileAndroid config = DatabaseManager.createConfigurationFileAndroid(
        "test.udb",
        getApplicationContext());
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```

参照

- Connection インターフェイス [Ultra Light J]105 ページ
- Configuration インターフェイス [Ultra LightJ]103 ページ

connect メソッド

設定に基づいて既存のデータベースに接続します。

構文

```
Connection DatabaseManager.connect(
    Configuration config
) throws ULjException
```

パラメーター

- **config**　既存のデータベースの仕様が含まれている Configuration オブジェクト。

戻り値

データベースへの接続を確立する Connection オブジェクト。

参照

- Configuration インターフェイス [Ultra LightJ]103 ページ
- Connection インターフェイス [Ultra Light J]105 ページ

createConfigurationFile メソッド

ファイルから物理データベーストア用の Configuration オブジェクトを作成し、ConfigFile オブジェクトを返します。

構文

```
ConfigFile DatabaseManager.createConfigurationFile(
    String file_name
) throws ULjException
```

パラメーター

- **file_name**　使用または作成するファイルの名前。

戻り値

データベースの設定に使用された ConfigFile オブジェクト。

参照

- ConfigFile インターフェイス [Ultra Light J]76 ページ

createConfigurationFileAndroid メソッド

Android デバイス上のファイルから物理データベーストア用の Configuration オブジェクトを作成し、ConfigFileAndroid オブジェクトを返します。

構文

```
ConfigFileAndroid DatabaseManager.createConfigurationFileAndroid(
    String file_name,
    android.content.Context context
) throws ULjException
```

パラメーター

- **file_name**　使用または作成するファイルの名前。
- **context**　Android アプリケーションからの Context オブジェクト。

戻り値

データベースの設定に使用された ConfigFileAndroid オブジェクト。

備考

次のいずれかの状態では、このメソッドを使用して、非ヌルの Context オブジェクトを設定する必要があります。

●file_name パラメーターがデータベースファイルへの絶対パスではない場合。

●アプリケーションがデータ同期に HTTPS プロトコルを使用する場合。

参照

- ConfigFileAndroid インターフェイス [Android] [Ultra Light J]79 ページ

createConfigurationFileME メソッド [BlackBerry]

J2ME デバイスのファイルシステム上のファイルから物理データベースストア用の Configuration オブジェクトを作成し、ConfigFileME オブジェクトを返します。

構文

```
ConfigFileME DatabaseManager.createConfigurationFileME(
    String db_name
) throws ULjException
```

パラメーター

- **db_name**　デバイス上のデータベースファイルの完全な URI 例 : *file:///SDCard/ulj/uljtest.ulj*.

戻り値

データベースの設定に使用された ConfigFileME オブジェクト。

備考

BlackBerry デバイスの場合、ストアは通常デバイスの SD メディアカードのことですが、他のファイルシステムも使用できます。

BlackBerry デバイスの場合、内部フラッシュのアクセスには file:///store/ で始まる URI (大文字と小文字が区別されます) を使用しますが、SD メディアカードのアクセスには file:///SDCard/ で始まる URI (大文字と小文字が区別されます) を使用します。

ファイルパスには、次の制限があります。

●ホスト部分は、大文字と小文字が区別されます。

●パーセント文字には特殊な意味があります。

●"." および ".." ディレクトリのような相対パスは使用できません。

ファイル名の制限の詳細については、『BlackBerry JDE API Reference』の **javax.microedition.io.file** オプションパッケージを参照してください。

参照

- ConfigFileME インターフェイス [BlackBerry] [Ultra Light J]81 ページ

createConfigurationNonPersistent メソッド [BlackBerry]

非永続的なデータベースストア用の Configuration オブジェクトを作成し、ConfigNonPersist オブジェクトを返します。

構文

```
ConfigNonPersistent DatabaseManager.createConfigurationNonPersistent(
    String db_name
) throws ULjException
```

パラメーター

- **db_name**　非永続的なデータベースの名前。

戻り値

データベースの設定に使用された ConfigNonPersistent オブジェクト。

参照

- ConfigNonPersistent インターフェイス [BlackBerry] [Ultra Light J]84 ページ

createConfigurationObjectStore メソッド [BlackBerry]

RIM オブジェクトストア用の Configuration オブジェクトを作成し、ConfigObjectStore オブジェクトを返します。

構文

```
ConfigObjectStore DatabaseManager.createConfigurationObjectStore(
    String db_name
) throws ULjException
```

パラメーター

- **db_name**　データベースの名前。

戻り値

データベースの設定に使用された ConfigObjectStore オブジェクト。

参照

- ConfigObjectStore インターフェイス [BlackBerry] [Ultra Light J]84 ページ

createConfigurationRecordStore メソッド [J2ME]

物理データベースストアとするレコードストア用の Configuration オブジェクトを作成し、ConfigRecordStore オブジェクトを返します。

構文

```
ConfigRecordStore DatabaseManager.createConfigurationRecordStore(
    String db_name
) throws ULjException
```

パラメーター

- **db_name**　データベースの名前。

戻り値

データベースの設定に使用された ConfigRecordStore オブジェクト。

参照

- ConfigRecordStore インターフェイス (J2ME のみ) [UltraLiteJ]100 ページ

createDatabase メソッド

設定セットに基づいて新しいデータベースを作成し、データベースに接続します。

構文

```
Connection DatabaseManager.createDatabase(
    Configuration config
) throws ULjException
```

パラメーター

- **config**　新規のデータベースの仕様が含まれている Configuration オブジェクト。

戻り値

新規のデータベースへの接続を確立する Connection オブジェクト。

備考

このメソッドは、デバイス上の同じ名前の既存データベースを置換します。

参照

- Configuration インターフェイス [Ultra LightJ]103 ページ
- Connection インターフェイス [Ultra Light J]105 ページ

createFileTransfer メソッド

Mobile Link との間でファイルを転送するための FileTransfer オブジェクトを作成します。

構文

```
FileTransfer DatabaseManager.createFileTransfer(
    String fileName,
    int streamType,
    String userName,
    String version
) throws ULjException
```


パラメーター

- **fileName** 転送するサーバーファイルの名前。このパラメーターにパス情報を含めることはできません。
- **streamType** 通信ストリームタイプの識別に使用する、SyncParms クラス内で定義されている定数の 1 つ。
- **userName** Mobile Link ユーザー名。
- **version** Mobile Link スクリプトのバージョン。

戻り値

FileTransfer オブジェクト。

参照

- SyncParms クラス [Ultra Light J]241 ページ

createObjectStoreTransfer メソッド [BlackBerry]

Mobile Link との間で Ultra Light Java Edition データベースを転送し、RIM オブジェクトストアに格納するための FileTransfer オブジェクトを作成します。

構文

```
FileTransfer DatabaseManager.createObjectStoreTransfer(
    String fileName,
    int streamType,
    String userName,
    String version
) throws ULjException
```

パラメーター

- **fileName** 転送するサーバーファイルの名前。このパラメーターにパス情報を含めることはできません。
- **streamType** 通信ストリームタイプの識別に使用する、SyncParms クラス内で定義されている定数の 1 つ。
- **userName** Mobile Link ユーザー名。
- **version** Mobile Link スクリプトのバージョン。

戻り値

FileTransfer オブジェクト。

参照

- SyncParms クラス [Ultra Light J]241 ページ

createSISHTTPListener メソッド [BlackBerry]

サーバー起動同期用の SISListener オブジェクトを作成します。

構文

```
SISListener DatabaseManager.createSISHTTPListener(
    SISRequestHandler handler,
    int port,
    String httpOptions
) throws ULjException
```

パラメーター

- **handler** サーバー起動同期要求を処理するために指定された SISRequestHandler オブジェクト。
- **port** サーバーメッセージを受信する HTTP ポート。
- **httpOptions** サーバーに接続するための HTTP オプション。

戻り値

サーバー起動同期に使用される SISListener オブジェクト。

備考

推奨されるポート設定は 4400 です。

BlackBerry シミュレーターに推奨される HTTP オプションは "deviceside=false" です。

BlackBerry HTTP SISListener の Mobile Link .側では、BES 対応デバイスなどの BlackBerry Enterprise Server にターゲットデバイスを関連付ける必要があります。

release メソッド

DatabaseManager オブジェクトを閉じてすべての接続を解放し、すべてのデータベースを停止します。

構文

```
void DatabaseManager.release() throws ULjException
```

備考

Android では、このメソッドによりすべてのリソースが解放されるので、アプリケーションの終了時に DatabaseManager オブジェクトおよびこのオブジェクトによって作成されたオブジェクトを使用し、このメソッドを単一スレッドで 1 回だけ呼び出す必要があります。

コミットされていないトランザクションはロールバックされます。

setErrorLanguage メソッド

エラーメッセージに使用する言語を設定します。

構文

```
void DatabaseManager.setErrorLanguage(String lang)
```

パラメーター

- **lang** 2 文字で表される言語コード。

備考

認識される言語は EN、DE、FR、JA、ZH です。指定された言語を認識できなかった場合は、デフォルトの言語である EN に戻ります。

J2SE と BlackBerry の各環境では、現在のロケールを使用してデフォルトの言語が決定されます。J2ME 環境では、このメソッドが、言語を指定する唯一の方法です。

DecimalNumber インターフェイス

正確な decimal 値を表し、java.math.BigDecimal を使用できない Java プラットフォームに 10 進法計算のサポートを提供します。

構文

```
public interface DecimalNumber
```

メンバー

継承されたメンバーを含む DecimalNumber インターフェイスのすべてのメンバー。

名前	説明
add メソッド	2 つの DecimalNumber オブジェクトを加算し、その和を返します。
divide メソッド	最初の DecimalNumber オブジェクトを 2 番目の DecimalNumber オブジェクトで割って、その商を返します。
getString メソッド	DecimalNumber オブジェクトの String 表現を返します。
isNull メソッド	DecimalNumber オブジェクトが null かどうかを確認します。

名前	説明
multiply メソッド	2 つの DecimalNumber オブジェクトを乗算し、その積を返します。
set メソッド	DecimalNumber オブジェクトに String 値を設定します。
setNull メソッド	DecimalNumber オブジェクトを null に設定します。
subtract メソッド	2 番目の DecimalNumber オブジェクトを最初の DecimalNumber オブジェクトから減算し、その差を返します。

add メソッド

2 つの DecimalNumber オブジェクトを加算し、その和を返します。

構文

```
DecimalNumber DecimalNumber.add(
    DecimalNumber num1,
    DecimalNumber num2
) throws ULjException
```

パラメーター

- **num1** 1 つの数値。
- **num2** 別の数値。

戻り値

num1 と num2 の和。

divide メソッド

最初の DecimalNumber オブジェクトを 2 番目の DecimalNumber オブジェクトで割って、その商を返します。

構文

```
DecimalNumber DecimalNumber.divide(
    DecimalNumber num1,
    DecimalNumber num2
) throws ULjException
```


パラメーター

- **num1** 被除数。
- **num2** 除数。

戻り値

num1 を num2 で割った商。

getString メソッド

DecimalNumber オブジェクトの String 表現を返します。

構文

```
String DecimalNumber.getString() throws ULjException
```

戻り値

String 値。

isNull メソッド

DecimalNumber オブジェクトが null かどうかを確認します。

構文

```
boolean DecimalNumber.isNull()
```

戻り値

オブジェクトが NULL の場合は true、NULL 以外の場合は false。

multiply メソッド

2 つの DecimalNumber オブジェクトを乗算し、その積を返します。

構文

```
DecimalNumber DecimalNumber.multiply(
    DecimalNumber num1,
    DecimalNumber num2
) throws ULjException
```

パラメーター

- **num1** 被乗数。
- **num2** 乗数。

戻り値

num1 と num2 の積。

set メソッド

DecimalNumber オブジェクトに String 値を設定します。

構文

```
void DecimalNumber.set(String value) throws ULjException
```

パラメーター

- **value** String として表した数値。

setNull メソッド

DecimalNumber オブジェクトを null に設定します。

構文

```
void DecimalNumber.setNull() throws ULjException
```

subtract メソッド

2 番目の DecimalNumber オブジェクトを最初の DecimalNumber オブジェクトから減算し、その差を返します。

構文

```
DecimalNumber DecimalNumber.subtract(
    DecimalNumber num1,
    DecimalNumber num2
) throws ULjException
```

パラメーター

- **num1** 被減数。
- **num2** 減数。

戻り値

num1 と num2 の差。

Domain インターフェイス

テーブル内のカラムの Domain オブジェクトの型情報を表します。

構文

```
public interface Domain
```

メンバー

継承されたメンバーを含む Domain インターフェイスのすべてのメンバー。

名前	説明
BIG 変数	64 ビット整数 (BIGINT SQL 型) のドメイン ID 定数です。
BINARY 変数	最大 *size* バイトの可変長のバイナリオブジェクト (BINARY (*size*) SQL 型) のドメイン ID 定数です。
BIT 変数	ビット (BIT SQL 型) のドメイン ID 定数です。
DATE 変数	日付 (DATE SQL 型) のドメイン ID 定数です。
DOMAIN_MAX 変数	Domain 型の最大数です。
DOUBLE 変数	8 バイトの浮動小数点数 (DOUBLE SQL 型) のドメイン ID 定数です。
INTEGER 変数	32 ビット整数 (INTEGER SQL 型) のドメイン ID 定数です。
LONGBINARY 変数	任意の長いブロックのバイナリデータ (BLOB) (LONG BINARY SQL 型) のドメイン ID 定数です。
LONGBINARYFILE 変数	任意のデータファイルのドメイン ID 定数です。
LONGVARCHAR 変数	任意の長いブロックの文字データ (CLOB) (LONG VARCHAR SQL 型) のドメイン ID 定数です。
NUMERIC 変数	合計桁数が固定 precision (サイズ)、小数点以下の桁数が *scale* 桁の数値 (NUMERIC(*precision,scale*) SQL 型) のドメイン ID 定数です。

名前	説明
REAL 変数	4 バイトの浮動小数点数 (REAL SQL 型) のドメイン ID 定数です。
SHORT 変数	16 ビット整数 (SMALLINT SQL 型) のドメイン ID 定数です。
ST_GEOMETRY 変数	ジオメトリ (GEOMETRY SQL 型) のドメイン ID 定数です。
TIME 変数	時刻 (TIME SQL 型) のドメイン ID 定数です。
TIMESTAMP 変数	タイムスタンプ (TIMESTAMP SQL 型) のドメイン ID 定数です。
TIMESTAMP_ZONE 変数	タイムゾーン付きタイムスタンプ (DATETIMEOFFSET SQL 型) のドメイン ID 定数です。
TINY 変数	符号なし 8 ビット整数 (TINYINT SQL 型) のドメイン ID 定数です。
UNSIGNED_BIG 変数	符号なし 64 ビット整数 (UNSIGNED BIGINT SQL 型) のドメイン ID 定数です。
UNSIGNED_INTEGER 変数	符号なし 32 ビット整数 (UNSIGNED INTEGER SQL 型) のドメイン ID 定数です。
UNSIGNED_SHORT 変数	符号なし 16 ビット整数 (UNSIGNED SMALLINT SQL 型) のドメイン ID 定数です。
UUID 変数	UniqueIdentifier (UNIQUEIDENTIFIER SQL 型) のドメイン ID 定数です。
VARCHAR 変数	最大 *size* バイトの可変長文字オブジェクト (VARCHAR (*size*) SQL 型) のドメイン ID 定数です。

備考

このインターフェイスには、さまざまなドメインを表す定数と、Domain オブジェクトから情報を抽出するためのメソッドがあります。

単純なデータベースのスキーマの作成例については、Connection インターフェイスを参照してください。

型は次のように分類できます。

整数型：

ドメイン定数	SQL 型	値の範囲
BIT	BIT	0 または 1
TINY	TINYINT	0 ～ 255 (1 バイトの記憶領域を使用する符号なし整数)
SHORT	SMALLINT	-32768 ～ 32767 (2 バイトの記憶領域を使用する符号付き整数)
UNSIGNED_SHORT	UNSIGNED SMALLINT	0 ～ 65535 (2 バイトの記憶領域を使用する符号なし整数)
INTEGER	INTEGER	-2^{31} ～ 2^{31} - 1、または -2147483648 ～ 2147483647 (4 バイトの記憶領域を使用する符号付き整数)
UNSIGNED_INTEGER	UNSIGNED INTEGER	0 ～ 2^{32} - 1、または 0 ～ 4294967295 (4 バイトの記憶領域を使用する符号なし整数)
BIG	BIGINT	-2^{63} ～ 2^{63} - 1、または -9223372036854775808 ～ 9223372036854775807 (8 バイトの記憶領域を使用する符号付き整数)
UNSIGNED_BIG	UNSIGNED BIGINT	0 ～ 2^{64} - 1、または 0 ～ 18446744073709551615 (8 バイトの記憶領域を使用する符号なし整数)

整数以外の数値型：

ドメイン定数	SQL 型	値の範囲
REAL	REAL	-3.402823e+38 ～ 3.402823e +38、0 に最も近い最小の数値は 1.175495e-38 (4 バイトの記憶領域を使用する単精度の浮動小数点数、6 桁目の後に丸め誤差が生じる可能性があります)

ドメイン定数	SQL 型	値の範囲
DOUBLE	DOUBLE	-1.79769313486231e+308 ～ 1.79769313486231e+308、0 に最も近い最小の数値は 2.22507385850721e-308 (8 バイトの記憶領域を使用する単精度の浮動小数点数、15 桁目の後に丸め誤差が生じる可能性があります)
NUMERIC	NUMERIC(precision,scale)	合計桁数が *precision* (サイズ)、小数点以下の桁数が *scale* 桁の任意の 10 進数 (precision 内の丸めなし)

文字型とバイナリ型：

ドメイン定数	SQL 型	サイズの範囲
VARCHAR	VARCHAR(size)	1 ～ 32767 バイト (文字は 1 ～ 3 バイトの UTF-8 文字として格納)。式を評価するときのテンポラリ文字値の最大長は 2048 バイトです。
LONGVARCHAR	LONG VARCHAR	任意の長さ (メモリで許容される範囲内)。LONG VARCHAR カラムで実行可能な演算は、これらの挿入、更新、削除、またはクエリの select リストへのこれらの指定のみです。
BINARY	BINARY(size)	1 ～ 32767 バイト。式を評価するときのテンポラリ文字値の最大長は 2048 バイトです。
LONGBINARY	LONG BINARY	任意の長さ (メモリで許容される範囲内)。LONG BINARY カラムで実行可能な演算は、これらの挿入、更新、削除、またはクエリの select リストへのこれらの指定のみです。

ドメイン定数	SQL 型	サイズの範囲
UUID	UNIQUEIDENTIFIER	常に 16 バイトの解釈が特殊なバイナリ

日付型と時間型：

ドメイン定数	SQL 型	値
DATE	DATE	年、月、日。
TIME	TIME	時、分、秒 (小数位あり) で構成される時刻。
TIMESTAMP	TIMESTAMP	DATE と TIME。
TIMESTAMP_ZONE	TIMESTAMP_ZONE	タイムゾーン付きの DATE と TIME。

BIT カラムはデフォルトでは NULL 入力不可です。その他の型はデフォルトで NULL 入力可です。

参照

- Connection インターフェイス [Ultra Light J]105 ページ

BIG 変数

64 ビット整数 (BIGINT SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.BIG
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

BINARY 変数

最大 *size* バイトの可変長のバイナリオブジェクト (BINARY (*size*) SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.BINARY
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

BIT 変数

ビット (BIT SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.BIT
```

備考

BIT カラムはデフォルトでは NULL 入力不可です。

参照

- Domain インターフェイス [Ultra Light J]143 ページ

DATE 変数

日付 (DATE SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.DATE
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

DOMAIN_MAX 変数

Domain 型の最大数です。

構文

```
final short Domain.DOMAIN_MAX
```

DOUBLE 変数

8 バイトの浮動小数点数 (DOUBLE SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.DOUBLE
```


参照

- Domain インターフェイス [Ultra Light J]143 ページ

INTEGER 変数

32 ビット整数 (INTEGER SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.INTEGER
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

LONGBINARY 変数

任意の長いブロックのバイナリデータ (BLOB) (LONG BINARY SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.LONGBINARY
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

LONGBINARYFILE 変数

任意のデータファイルのドメイン ID 定数です。

構文

```
final short Domain.LONGBINARYFILE
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

LONGVARCHAR 変数

任意の長いブロックの文字データ (CLOB) (LONG VARCHAR SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.LONGVARCHAR
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

NUMERIC 変数

合計桁数が固定 precision (サイズ)、小数点以下の桁数が *scale* 桁の数値 (NUMERIC(*precision,scale*) SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.NUMERIC
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

REAL 変数

4 バイトの浮動小数点数 (REAL SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.REAL
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

SHORT 変数

16 ビット整数 (SMALLINT SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.SHORT
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

ST_GEOMETRY 変数

ジオメトリ (GEOMETRY SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.ST_GEOMETRY
```


参照

- Domain インターフェイス [Ultra Light J]143 ページ

TIME 変数

時刻 (TIME SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.TIME
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

TIMESTAMP 変数

タイムスタンプ (TIMESTAMP SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.TIMESTAMP
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

TIMESTAMP_ZONE 変数

タイムゾーン付きタイムスタンプ (DATETIMEOFFSET SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.TIMESTAMP_ZONE
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

TINY 変数

符号なし 8 ビット整数 (TINYINT SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.TINY
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

UNSIGNED_BIG 変数

符号なし 64 ビット整数 (UNSIGNED BIGINT SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.UNSIGNED_BIG
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

UNSIGNED_INTEGER 変数

符号なし 32 ビット整数 (UNSIGNED INTEGER SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.UNSIGNED_INTEGER
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

UNSIGNED_SHORT 変数

符号なし 16 ビット整数 (UNSIGNED SMALLINT SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.UNSIGNED_SHORT
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

UUID 変数

UniqueIdentifier (UNIQUEIDENTIFIER SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.UUID
```


参照

- Domain インターフェイス [Ultra Light J]143 ページ

VARCHAR 変数

最大 *size* バイトの可変長文字オブジェクト (VARCHAR (*size*) SQL 型) のドメイン ID 定数です。

構文

```
final short Domain.VARCHAR
```

参照

- Domain インターフェイス [Ultra Light J]143 ページ

EncryptionControl インターフェイス

データベースに対して EncryptionControl オブジェクト機能を提供します。

構文

```
public interface EncryptionControl
```

メンバー

継承されたメンバーを含む EncryptionControl インターフェイスのすべてのメンバー。

名前	説明
decrypt メソッド	データベース内の byte 配列を復号化します。
encrypt メソッド	データベース内の byte 配列を暗号化します。
initialize メソッド	パスワードを使用して暗号化制御を初期化します。

備考

このインターフェイスは、独自の暗号化または難読化の手法を実装するために使用します。データベースを暗号化するには、EncryptionControl インターフェイスを実装する新しいクラスを作成して、このクラスで独自の暗号化手法を指定します。次に、ConfigPersistent インターフェイスの setEncryption メソッドを使用して、暗号化されたデータベースを作成してそれにアクセスする Configuration オブジェクトを作成します。

注意
ファイルとして格納されるカラム (LONG BINARY STORE AS FILE として定義される) は、暗号化も復号化もされません。追加のセキュリティが必要な場合は、BlackBerry に組み込まれた SDCard 暗号化を使用します。組み込みセキュリティを使用して暗号化されたファイルには、デバイスがロックされている間はアクセスできません。

通常の LONG BINARY カラムは、データベースに格納されます。したがって、データベースの他の部分で暗号化および復号化されます。

BlackBerry デバイスで EncryptionControl オブジェクトを使用して暗号化されたデータベースには、デバイスがロックされていてもアクセスできます。**UltraLiteJ Security on BlackBerry Devices** ホワイトペーパーには、セキュリティオプションに関する情報や、BlackBerry デバイスでの Ultra Light J アプリケーションに関する考慮事項 (デバイスの組み込みセキュリティをアプリケーションで操作する場合の利点と短所など) が記載されています。UltraLiteJ Security on BlackBerry Devices を参照してください。

暗号化のコードサンプルの詳細については、Wandering Data blog を参照してください。

参照

- ConfigPersistent.setEncryption メソッド [Android] [Ultra Light J]95 ページ

decrypt メソッド

データベース内の byte 配列を復号化します。

構文

```
void EncryptionControl.decrypt(
    int page_no,
    byte[] src,
    byte[] tgt,
    int num_bytes
) throws ULjException
```

パラメーター

- **page_no**　配列データのページ番号。
- **src**　暗号化されているソースページ。この配列は変更できません。
- **tgt**　メソッドで復号化されたページ。
- **num_bytes**　復号化するバイト数。常に、ページサイズまたは 128 のいずれかになります。

備考

このメソッドには、暗号化された byte 配列、ソース、関連するページ番号を指定します。メソッドでは、src byte 配列の最初の num_bytes バイトを復号化し、その結果を tgt byte 配列に格納します。その後、アプリケーション内のデータ操作に tgt を使用します。

使用するアルゴリズムでは、データのサイズ (暗号化されたデータは元のデータと同じ長さになります) を保持し、一部のページ (ページ 0 は最初は 128 バイトとして読み込みと復号化が行われ、以降はフルサイズのページとして読み込みと復号化が行われます) を復号化できる必要があります。src 配列は変更できません。

encrypt メソッド

データベース内の byte 配列を暗号化します。

構文

```
void EncryptionControl.encrypt(
    int page_no,
    byte[] src,
    byte[] tgt
) throws ULjException
```

パラメーター

- **page_no**　配列データのページ番号。
- **src**　復号化されているソースページ。この配列は変更できません。
- **tgt**　メソッドで暗号化されたページ。

備考

このメソッドには、暗号化されていない byte 配列、ソース、関連するページ番号を指定します。メソッドでは、src を暗号化または難読化し、その結果を tgt byte 配列に格納します。その後 tgt をデータベースに格納します。

使用するアルゴリズムでは、データのサイズ (暗号化されたデータは元のデータと同じ長さになります) を保持し、一部のページ (ページ 0 は最初は 128 バイトとして読み込みと復号化が行われ、以降はフルサイズのページとして読み込みと復号化が行われます) を復号化できる必要があります。src 配列は変更できません。

initialize メソッド

パスワードを使用して暗号化制御を初期化します。

構文

```
void EncryptionControl.initialize(String password) throws ULjException
```

パラメーター

- **password**　暗号化と復号化に使用するパスワード。

FileTransfer インターフェイス

クライアントと Mobile Link サーバー間のファイル転送メカニズムを提供します。

構文

```
public interface FileTransfer
```

メンバー

継承されたメンバーを含む FileTransfer インターフェイスのすべてのメンバー。

名前	説明
downloadFile メソッド	このオブジェクトの指定されたプロパティのファイルをダウンロードします。
getAuthenticationParms メソッド	カスタムのユーザー認証スクリプトに渡されるパラメーターを返します。
getAuthStatus メソッド	前回行われたファイル転送の認証ステータスコードを返します。
getAuthValue メソッド	カスタムユーザー認証同期スクリプトで指定されている値を返します。
getFileAuthCode メソッド	前回行われたファイル転送の authenticate_file_transfer スクリプトからの戻り値を返します。
getLivenessTimeout メソッド	活性タイムアウトの長さを秒単位で返します。
getLocalFileName メソッド	ファイルのローカル名を決定します。
getLocalPath メソッド	ローカルファイルシステムにおけるファイルの検索場所または格納場所を指定します。
getPassword メソッド	setUserName メソッドで指定されたユーザーの Mobile Link パスワードを返します。
getRemoteKey メソッド	現在のリモートキーの値を決定します。
getServerFileName メソッド	サーバー上のファイルの名前を返します。
getStreamErrorCode メソッド	ストリームによってレポートされたエラーコードを返します。

名前	説明
getStreamErrorMessage メソッド	ストリーム自体によってレポートされるエラーメッセージを返します。
getStreamParms メソッド	同期ストリームを設定するパラメーターを返します。
getUserName メソッド	Mobile Link サーバーがクライアントをユニークに識別する Mobile Link ユーザー名を返します。
getVersion メソッド	使用する同期スクリプトを返します。
isResumePartialTransfer メソッド	前の部分的な転送を再開するか、破棄するかを決定します。
isTransferredFile メソッド	前回行われたファイル転送時にファイルが実際にダウンロードされたかどうかを確認します。
setAuthenticationParms メソッド	カスタムユーザー認証スクリプト (Mobile Link authenticate_parameters 接続イベント) のパラメーターを指定します。
setLivenessTimeout メソッド	活性タイムアウトの長さを秒単位で設定します。
setLocalFileName メソッド	ファイルのローカル名を指定します。
setLocalPath メソッド	ローカルファイルシステムにおけるファイルの検索場所または格納場所を指定します。
setPassword メソッド	setUserName メソッドで指定されたユーザーの Mobile Link パスワードを設定します。
setRemoteKey メソッド	リモートキーを指定します。
setResumePartialTransfer メソッド	前の部分的な転送を再開するか、破棄するかを指定します。
setServerFileName メソッド	サーバー上のファイルの名前を指定します。
setUserName メソッド	Mobile Link サーバーがクライアントをユニークに識別する Mobile Link ユーザー名を設定します。
setVersion メソッド	使用する同期スクリプトを設定します。

名前	説明
uploadFile メソッド	このオブジェクトの指定されたプロパティのファイルをアップロードします。

備考

FileTransfer オブジェクトは、DatabaseManager.createFileTransfer メソッドまたは DatabaseManager.createObjectStoreTransfer [BlackBerry] メソッドを呼び出すことによって取得します。

createFileTransfer メソッドによって返されたインスタンスを使用して、Mobile Link とローカルファイルシステム間で任意のファイルを転送できます。

BlackBerry デバイスとシミュレーターの場合、ローカルファイルシステムは、メディアカードまたは内部フラッシュファイルシステム (デバイスで使用できる場合) のいずれかになります。ファイル名の制約については、DatabaseManager.createConfigurationFileME メソッドの説明を参照してください。

Android デバイスとシミュレーターの場合、ローカルファイルシステムは、メディアカードか、またはアプリケーションに適切なパーミッションがある内部フラッシュファイルシステム (/sdcard/Android/data/your.package.name/files/など) のいずれかになります。

createObjectStoreTransfer メソッドから返されたインスタンスを使用して、ローカルの BlackBerry オブジェクトストアへの Ultra Light Java データベースファイルのダウンロード (またはその逆) を実行できます。

転送できるのは、有効で暗号化されていない Ultra Light Java Edition データベースファイルのみです。Ultra Light Java Edition データベース以外のファイルをダウンロードしようとすると、例外がスローされます。

注意
アプリケーションで、同じローカルファイルに対して、2 つのダウンロードを同時に行うことはできません。

参照

- DatabaseManager.createConfigurationFileME メソッド [BlackBerry] [Ultra Light J]134 ページ
- DatabaseManager.createFileTransfer メソッド [Ultra Light J]136 ページ
- DatabaseManager.createObjectStoreTransfer メソッド [BlackBerry] [Ultra Light J]137 ページ

downloadFile メソッド

このオブジェクトの指定されたプロパティのファイルをダウンロードします。

オーバーロードリスト

名前	説明
downloadFile() メソッド	このオブジェクトの指定されたプロパティのファイルをダウンロードします。
downloadFile(FileTransferProgressListener) メソッド	指定されたリスナーに送信されるプログレスイベントとともに、このオブジェクトのプロパティで指定されたファイルをダウンロードします。

downloadFile() メソッド

このオブジェクトの指定されたプロパティのファイルをダウンロードします。

構文

```
abstract boolean FileTransfer.downloadFile() throws ULjException
```

戻り値

ダウンロードに成功した場合は true 、そうでない場合は ULjException がスローされメソッドは正常に返されません。

備考

setServerFileName メソッドで指定されたファイルは、指定のストリーム、userName、パスワード、スクリプトバージョンを使用して、setLocalPath メソッドで指定されたパスに、Mobile Link サーバーからダウンロードされます。

setLocalFileName()、setAuthenticationParms()、setResumePartialTransfer() の各メソッドでは、他のオプションも指定できます。

ファイルが破損することを防ぐため、デスクトップおよび BlackBerry ファイルシステムのダウンロードでは、Ultra Light J はテンポラリファイルにダウンロードし、ダウンロードが完了したときにのみローカルファイルと置換します。

BlackBerry オブジェクトストアのダウンロードでは、Ultra Light J はローカルストアに直接書き込みを開始します。これは、この場合、アトミックテンポラリオブジェクトを作成できないからです。同じ名前の既存のローカルデータベースストアは、transferFile メソッドを呼び出した時点で破損されます。

getAuthStatus()、getAuthValue()、getFileAuthCode()、isTransferredFile()、getStreamErrorCode()、getStreamErrorMessage() の各メソッドを使用して、結果の詳細なステータスを取得できます。

downloadFile(FileTransferProgressListener) メソッド

指定されたリスナーに送信されるプログレスイベントとともに、このオブジェクトのプロパティで指定されたファイルをダウンロードします。

構文

```
abstract boolean FileTransfer.downloadFile(
    FileTransferProgressListener listener
) throws ULjException
```

パラメーター

- **listener**　ファイル転送プログレスイベントを受信するオブジェクト。

戻り値

ダウンロードに成功した場合は true 、そうでない場合は ULjException がスローされメソッドは正常に返されません。

備考

エラーが発生すると、リスナーにはデータが送信されないことがあります。

参照

- FileTransfer.downloadFile メソッド [Ultra Light J]158 ページ

getAuthenticationParms メソッド

カスタムのユーザー認証スクリプトに渡されるパラメーターを返します。

構文

```
abstract String FileTransfer.getAuthenticationParms()
```

戻り値

認証パラメーターのリスト、またはパラメーターが指定されていない場合は NULL。

参照

- FileTransfer.setAuthenticationParms メソッド [Ultra Light J]166 ページ

getAuthStatus メソッド

前回行われたファイル転送の認証ステータスコードを返します。

構文

```
abstract int FileTransfer.getAuthStatus()
```

戻り値

AuthStatusCode クラスの値。

getAuthValue メソッド

カスタムユーザー認証同期スクリプトで指定されている値を返します。

構文

```
abstract long FileTransfer.getAuthValue()
```

戻り値

カスタムユーザー認証同期スクリプトから返された整数。

getFileAuthCode メソッド

前回行われたファイル転送の authenticate_file_transfer スクリプトからの戻り値を返します。

構文

```
abstract int FileTransfer.getFileAuthCode()
```

戻り値

前回行われたファイル転送の authenticate_file_transfer スクリプトから返された整数。

getLivenessTimeout メソッド

活性タイムアウトの長さを秒単位で返します。

構文

```
abstract int FileTransfer.getLivenessTimeout()
```

戻り値

タイムアウト。

参照

- FileTransfer.setLivenessTimeout メソッド [Ultra Light J]166 ページ

getLocalFileName メソッド

ファイルのローカル名を決定します。

構文

```
abstract String FileTransfer.getLocalFileName()
```

戻り値

ダウンロードしたファイルのローカルファイル名。

備考

ファイルのダウンロードの場合は、ダウンロードしたファイルの名前になります。ファイルのアップロードの場合は、アップロードするファイルの名前になります。

参照

- FileTransfer.setLocalFileName メソッド [Ultra Light J]167 ページ

getLocalPath メソッド

ローカルファイルシステムにおけるファイルの検索場所または格納場所を指定します。

構文

```
abstract String FileTransfer.getLocalPath()
```

戻り値

ローカルディレクトリ。

参照

- FileTransfer.setLocalPath メソッド [Ultra Light J]167 ページ

getPassword メソッド

setUserName メソッドで指定されたユーザーの Mobile Link パスワードを返します。

構文

```
abstract String FileTransfer.getPassword()
```

戻り値

Mobile Link ユーザーのパスワード。

参照

- FileTransfer.setPassword メソッド [Ultra Light J]168 ページ

getRemoteKey メソッド

現在のリモートキーの値を決定します。

構文

```
abstract String FileTransfer.getRemoteKey()
```

戻り値

リモートキーの値、またはリモートキーが指定されていない場合は NULL。

参照

- FileTransfer.setRemoteKey メソッド [Ultra Light J]168 ページ

getServerFileName メソッド

サーバー上のファイルの名前を返します。

構文

```
abstract String FileTransfer.getServerFileName()
```

戻り値

サーバー側のファイルの名前。

備考

ファイルのダウンロードの場合は、ダウンロードしたファイルの名前になります。ファイルのアップロードの場合は、アップロードするファイルの名前になります。

参照

- FileTransfer.setServerFileName メソッド [Ultra Light J]170 ページ

getStreamErrorCode メソッド

ストリームによってレポートされたエラーコードを返します。

構文

```
abstract int FileTransfer.getStreamErrorCode()
```

戻り値

通信ストリームエラーがなかった場合は 0、それ以外の場合はサーバーからの応答コードを返します。

備考

エラーコードは HTTP の応答コードです。

getStreamErrorMessage メソッド

ストリーム自体によってレポートされるエラーメッセージを返します。

構文

```
abstract String FileTransfer.getStreamErrorMessage()
```

戻り値

メッセージがない場合は null、それ以外の場合は応答メッセージを返します。

備考

これは HTTP 応答メッセージです。

getStreamParms メソッド

同期ストリームを設定するパラメーターを返します。

構文

```
abstract StreamHTTPParms FileTransfer.getStreamParms()
```

戻り値

HTTP または HTTPS ストリームのパラメーターを指定する StreamHTTPParms または StreamHTTPSParms オブジェクト。オブジェクトは参照で返されます。

備考

同期ストリームのタイプは、FileTransfer オブジェクトの作成時に指定します。

getUserName メソッド

Mobile Link サーバーがクライアントをユニークに識別する Mobile Link ユーザー名を返します。

構文

```
abstract String FileTransfer.getUserName()
```

戻り値

Mobile Link ユーザー名。

参照

- FileTransfer.setUserName メソッド [Ultra Light J]170 ページ

getVersion メソッド

使用する同期スクリプトを返します。

構文

```
abstract String FileTransfer.getVersion()
```

戻り値

スクリプトバージョン

参照

- FileTransfer.setVersion メソッド [Ultra Light J]171 ページ

isResumePartialTransfer メソッド

前の部分的な転送を再開するか、破棄するかを決定します。

構文

```
abstract boolean FileTransfer.isResumePartialTransfer()
```

戻り値

ダウンロードを再開する場合は true、それ以外の場合は false。

参照

- FileTransfer.setResumePartialTransfer メソッド [Ultra Light J]169 ページ

isTransferredFile メソッド

前回行われたファイル転送時にファイルが実際にダウンロードされたかどうかを確認します。

構文

```
abstract boolean FileTransfer.isTransferredFile()
```

戻り値

ファイルが転送された場合は true、それ以外の場合は false。

備考

transferFile() メソッドが呼び出された時点でファイルがすでに最新だった場合、isTransferredFile() メソッドでは false が返されますが、このメソッドでは true が返されます。

エラーが発生し、transferFile() メソッドによって例外がスローされる場合は、isTransferredFile() メソッドにより false が返されます。

setAuthenticationParms メソッド

カスタムユーザー認証スクリプト (Mobile Link authenticate_parameters 接続イベント) のパラメーターを指定します。

構文

```
abstract void FileTransfer.setAuthenticationParms(
    String authParms
) throws ULjException
```

パラメーター

- **authParms**　認証パラメーターのカンマ区切りのリスト、または NULL 参照。カンマ区切りのリストの詳細については、SyncParms クラスの説明を参照してください。

備考

最初の 255 文字列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)。

参照

- FileTransfer.getAuthenticationParms メソッド [Ultra Light J]160 ページ

setLivenessTimeout メソッド

活性タイムアウトの長さを秒単位で設定します。

構文

```
abstract void FileTransfer.setLivenessTimeout(
    int timeout
) throws ULjException
```

パラメーター

- **timeout**　新しい活性タイムアウト値。

備考

活性タイムアウトは、サーバーで許容される、リモートのアイドル時間の長さです。リモートが 1 秒間サーバーと通信しなかった場合、サーバーはリモートとの接続が失われたとみなし、ファイル転送を終了します。リモートは、接続を継続するために、自動的に定期メッセージをサーバーに送信します。

負の値を設定すると、例外がスローされます。値は Mobile Link サーバーによって予告なく変更される場合があります。変更は、値が低すぎるか高すぎる場合に行われます。

デフォルトの値は、BlackBerry/J2SE/J2ME プラットフォームでは 100 秒、Android プラットフォームでは 240 秒です。

参照

- FileTransfer.getLivenessTimeout メソッド [Ultra Light J]161 ページ

setLocalFileName メソッド

ファイルのローカル名を指定します。

構文

```
abstract void FileTransfer.setLocalFileName(String localFileName)
```

パラメーター

- **localFileName**　ダウンロードファイルのローカル名を指定する文字列。値が NULL 参照の場合は、fileName が使用されます。デフォルトは NULL 参照です。

備考

ファイルのダウンロードの場合は、ダウンロードしたファイルの名前になります。ファイルのアップロードの場合は、アップロードするファイルの名前になります。ファイル名にドライブまたはパスの情報を含めることはできません。

参照

- FileTransfer.getLocalFileName メソッド [Ultra Light J]161 ページ
- FileTransfer.setLocalPath メソッド [Ultra Light J]167 ページ

setLocalPath メソッド

ローカルファイルシステムにおけるファイルの検索場所または格納場所を指定します。

構文

```
abstract void FileTransfer.setLocalPath(String localPath)
```

パラメーター

- **localPath**　ファイルのローカルディレクトリを指定する文字列。デフォルトは NULL 参照です。

備考

ローカルディレクトリの構文は、プラットフォームによって異なります。

●デスクトップの場合の構文は、"C:¥¥ulj¥¥" のようになります。

●BlackBerry ファイルシステムの場合の構文は、"file:///SDCard/ulj/" のようになります。

●BlackBerry オブジェクトストアの場合、このオプションは無視されます。

●Android ファイルシステムの場合の構文は、"/sdcard/Android/data/your.package.name/files/" のようになります。

デフォルトのローカルディレクトリも、デバイスのオペレーティングシステムによって異なります。

●デスクトップコンピューターでは、localPath パラメーターが null の場合、ファイルは現在のディレクトリに格納されます。

●BlackBerry ファイルシステムストアの場合、localPath パラメーターにデフォルトの値がないので、明示的に設定する必要があります。

●Android ファイルシステムストアの場合、localPath パラメーターにデフォルトの値がないので、明示的に設定する必要があります。

参照

- FileTransfer.getLocalPath メソッド [Ultra Light J]162 ページ
- FileTransfer.setLocalFileName メソッド [Ultra Light J]167 ページ

setPassword メソッド

setUserName メソッドで指定されたユーザーの Mobile Link パスワードを設定します。

構文

```
abstract void FileTransfer.setPassword(
    String password
) throws ULjException
```

パラメーター

- **password**　Mobile Link ユーザーのパスワード。

備考

このユーザー名とパスワードは、データベースの他のユーザー ID とパスワードと異なります。このメソッドで、Mobile Link サーバーに対してアプリケーションが認証されます。

デフォルトは空の文字列で、これはパスワードなしを示します。

参照

- FileTransfer.getPassword メソッド [Ultra Light J]162 ページ
- FileTransfer.setUserName メソッド [Ultra Light J]170 ページ

setRemoteKey メソッド

リモートキーを指定します。

構文

```
abstract void FileTransfer.setRemoteKey(String remoteKey)
```

パラメーター

- **remoteKey** リモートキーの値、またはリモートキーが指定されていない場合は NULL。

備考

リモートキーは、サーバーの authenticate_file_upload スクリプトに渡されるパラメーターです。

スクリプトでは、このパラメーターを使用して、サーバーに格納するファイルの名前と場所を決定します。

この値が指定されていない場合は、getFileName() の値がリモートキーとして使用されます。

参照

- FileTransfer.getRemoteKey メソッド [Ultra Light J]162 ページ

setResumePartialTransfer メソッド

前の部分的な転送を再開するか、破棄するかを指定します。

構文

```
abstract void FileTransfer.setResumePartialTransfer(boolean resume)
```

パラメーター

- **resume** 前回の部分的なダウンロードを再開する場合は true、破棄する場合は false に設定します。

備考

デフォルトは true です。

Ultra Light J では、通信エラーや、ユーザーによる FileTransferProgressListener オブジェクトからのアボートが原因で失敗したファイル転送を再起動できます。

ファイルのダウンロードの場合、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ファイルが保持されるため、次の転送中に再開できます。ファイルがサーバー上で更新されている場合は、部分的なダウンロードは破棄され、新しくダウンロードが開始されます。

ファイルのアップロードの場合、以降のファイルのアップロードで前回のアップロードを再開できるようにするため、Mobile Link サーバーは部分的にアップロードされたファイルを保持します。ただし、ファイルがローカルで更新されている場合は、部分的なアップロードは破棄され、新しくアップロードが開始されます。

参照

- FileTransfer.isResumePartialTransfer メソッド [Ultra Light J]165 ページ

setServerFileName メソッド

サーバー上のファイルの名前を指定します。

構文

```
abstract void FileTransfer.setServerFileName(
    String fileName
) throws ULjException
```

パラメーター

- **fileName** Mobile Link サーバーによって認識されたファイルの名前を指定する文字列。

備考

ファイルのダウンロードの場合は、ダウンロードしたファイルの名前になります。ファイルのアップロードの場合は、アップロードするファイルの名前になります。

このパラメーターは、FileTransfer オブジェクトの作成時に初期化されます。

Mobil Link は、指定されたファイルを最初に userName サブディレクトリで、次にルートディレクトリで検索します。ルートダウンロードディレクトリは、Mobile Link サーバーの -ftr オプションで指定され、ルートアップロードディレクトリは -ftru オプションで指定されます。

fileName にはドライブまたはパスの情報を含めないでください。そのような情報を含めると、ファイルが見つからなくなります。たとえば、"myfile.txt" は有効ですが、"somedir¥myfile.txt"、"..¥myfile.txt"、"c:¥myfile.txt" は無効です。

参照

- FileTransfer.getServerFileName メソッド [Ultra Light J]163 ページ

setUserName メソッド

Mobile Link サーバーがクライアントをユニークに識別する Mobile Link ユーザー名を設定します。

構文

```
abstract void FileTransfer.setUserName(
    String userName
) throws ULjException
```

パラメーター

- **userName** Mobile Link ユーザー名。

備考

Mobile Link サーバーは、この値を使用して、サーバー側でファイルを特定します。Mobile Link ユーザー名とパスワードは他のデータベースユーザー 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 サーバーに対して識別し、認証するために使用されます。

このパラメーターは、FileTransfer オブジェクトの作成時に初期化されます。

参照

- FileTransfer.getUserName メソッド [Ultra Light J]164 ページ
- FileTransfer.setPassword メソッド [Ultra Light J]168 ページ

setVersion メソッド

使用する同期スクリプトを設定します。

構文

```
abstract void FileTransfer.setVersion(
    String version
) throws ULjException
```

パラメーター

- **version**　スクリプトバージョン

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区別されます。Ultra Light J 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。

このパラメーターは、FileTransfer オブジェクトの作成時に初期化されます。

参照

- FileTransfer.getVersion メソッド [Ultra Light J]165 ページ

uploadFile メソッド

このオブジェクトの指定されたプロパティのファイルをアップロードします。

オーバーロードリスト

名前	説明
uploadFile() メソッド	このオブジェクトの指定されたプロパティのファイルをアップロードします。
uploadFile(FileTransferProgressListener) メソッド	指定されたリスナーに送信されるプログレスイベントとともに、このオブジェクトのプロパティで指定されたファイルをアップロードします。

uploadFile() メソッド

このオブジェクトの指定されたプロパティのファイルをアップロードします。

構文

```
abstract boolean FileTransfer.uploadFile() throws ULjException
```

戻り値

アップロードに成功した場合は true 、そうでない場合は ULjException がスローされメソッドは正常に返されません。

備考

setLocalFileName メソッドと setLocalPath メソッドで指定されたファイルは、指定のストリーム、ユーザー名、パスワード、スクリプトバージョンを使用して、Mobile Link サーバーの、setServerFileName メソッドで指定されたファイルにアップロードされます。

setAuthenticationParms() メソッドと setResumePartialTransfer() メソッドでは、他のオプションも指定できます。

getAuthStatus()、getAuthValue()、getFileAuthCode()、isTransferredFile()、getStreamErrorCode()、getStreamErrorMessage() の各メソッドを使用して、結果の詳細なステータスを取得できます。

uploadFile(FileTransferProgressListener) メソッド

指定されたリスナーに送信されるプログレスイベントとともに、このオブジェクトのプロパティで指定されたファイルをアップロードします。

構文

```
abstract boolean FileTransfer.uploadFile(
    FileTransferProgressListener listener
) throws ULjException
```

パラメーター

- **listener**　ファイル転送プログレスイベントを受信するオブジェクト。

戻り値

アップロードに成功した場合は true 、そうでない場合は ULjException がスローされメソッドは正常に返されません。

備考

エラーが発生すると、リスナーにはデータが送信されないことがあります。

参照

- FileTransfer.uploadFile メソッド [Ultra Light J]171 ページ

FileTransferProgressData インターフェイス

ファイル転送の進行状況のモニタリングデータを通知します。

構文

```
public interface FileTransferProgressData
```

メンバー

継承されたメンバーを含む FileTransferProgressData インターフェイスのすべてのメンバー。

名前	説明
getBytesTransferred メソッド	現在までに転送されたバイト数を返します。
getFileSize メソッド	転送されるファイルのサイズを返します。
getResumedAtSize メソッド	転送が再開されるファイル内のポイントを返します。

getBytesTransferred メソッド

現在までに転送されたバイト数を返します。

構文

```
abstract long FileTransferProgressData.getBytesTransferred()
```

戻り値

現在までに転送されたバイト数。

備考

このメソッドでは、現在のファイル転送セッションによって転送されたバイト数のカウントに、以前に中断された転送によって転送されたバイト数が加算されます。

getResumedAtSize メソッドによって返された値を引くと、現在のセッションによって転送されたバイト数を算出できます。

参照

- FileTransferProgressData.getResumedAtSize メソッド [Ultra Light J]174 ページ

getFileSize メソッド

転送されるファイルのサイズを返します。

構文

```
abstract long FileTransferProgressData.getFileSize()
```

戻り値

ファイルのサイズ (バイト単位)。

備考

戻り値は、ファイル転送セッションの継続中は一定の値になります。

getResumedAtSize メソッド

転送が再開されるファイル内のポイントを返します。

構文

```
abstract long FileTransferProgressData.getResumedAtSize()
```

戻り値

以前に転送されたバイト数。

備考

戻り値は、ファイル転送セッションの継続中は一定の値になります。

FileTransferProgressListener インターフェイス

ファイル転送の進行状況イベントを受信します。

構文

```
public interface FileTransferProgressListener
```

メンバー

継承されたメンバーを含む FileTransferProgressListener インターフェイスのすべてのメンバー。

名前	説明
fileTransferProgressed メソッド	ユーザーに転送の進行状況を通知するために、ファイル転送中に呼び出されます。

備考

ファイル転送中に進行状況レポートを受信するために、新しいクラスが作成されます。

次の例は、FileTransferProgressListener インターフェイスを実装する単純な SyncObserver インターフェイスを示しています。


```
class MyObserver implements FileTransferProgressListener {
  public boolean fileTransferProgressed( FileTransferProgressData data ) {
     System.out.println(
        "file transfer progress "
        + " bytes received = " + data.getBytesTransferred()
     );
     return false;   // Always continue file transfer.
  }
  public MyObserver() {} // The default constructor.
}
```

fileTransferProgressed メソッド

ユーザーに転送の進行状況を通知するために、ファイル転送中に呼び出されます。

構文

```
boolean FileTransferProgressListener.fileTransferProgressed(
    FileTransferProgressData data
)
```

パラメーター

- **data**　最新のファイル転送プログレスデータを保持している FileTransferProgressData オブジェクト。

戻り値

転送をキャンセルする場合は true を、続行する場合は false を返します。

備考

リスナーは、次の状況で呼び出されます。

●最初のディスク書き込みの前

●ディスク書き込みごとまたは 0.5 秒ごとのどちらか後のほう

●ファイルのダウンロードの完了後

通常、キャンセル要求は Ultra Light J API で受け入れられます。その結果、errorCode が ULjException.SQLE_INTERRUPTED 定数に設定された ULjException オブジェクトがスローされます。

ただし、BlackBerry オブジェクトストアに対してダウンロードが完了した場合、Ultra Light J は転送をキャンセルしません。

fileTransferProgressed の呼び出し中に、Ultra Light J API のメソッドを呼び出さないでください。

IndexSchema インターフェイス

インデックスのスキーマを指定し、システムテーブルの問い合わせに便利な定数を提供します。

構文

```
public interface IndexSchema
```

メンバー

継承されたメンバーを含む IndexSchema インターフェイスのすべてのメンバー。

名前	説明
ASCENDING 変数	カラムのインデックスが昇順でソートされます。
DESCENDING 変数	カラムのインデックスが降順でソートされます。
PERSISTENT 変数	インデックスが永続的であることを示します。
PRIMARY_INDEX 変数	インデックスがプライマリキーであることを示します。
UNIQUE_INDEX 変数	インデックスがユニークインデックスであることを示します。
UNIQUE_KEY 変数	インデックスがユニークキーであることを示します。

備考

このインターフェイスには、インデックスのフラグやとソート順など、インデックス関連の定数のみが含まれます。

参照

- TableSchema.SYS_INDEXES 変数 [Ultra Light J]267 ページ

ASCENDING 変数

カラムのインデックスが昇順でソートされます。

構文

```
final byte IndexSchema.ASCENDING
```

DESCENDING 変数

カラムのインデックスが降順でソートされます。

構文

```
final byte IndexSchema.DESCENDING
```

PERSISTENT 変数

インデックスが永続的であることを示します。

構文

```
final byte IndexSchema.PERSISTENT
```

備考

この値は、SYS_TABLES システムテーブルの table_flags カラムで他のフラグと論理的に組み合わせることができます。

参照

- TableSchema.SYS_INDEXES 変数 [Ultra Light J]267 ページ

PRIMARY_INDEX 変数

インデックスがプライマリキーであることを示します。

構文

```
final byte IndexSchema.PRIMARY_INDEX
```

備考

この値は、SYS_TABLES システムテーブルの table_flags カラムで他のフラグと論理的に組み合わせることができます。

参照

- TableSchema.SYS_INDEXES 変数 [Ultra Light J]267 ページ

UNIQUE_INDEX 変数

インデックスがユニークインデックスであることを示します。

構文

```
final byte IndexSchema.UNIQUE_INDEX
```

備考

この値は、SYS_TABLES システムテーブルの table_flags カラムで他のフラグと論理的に組み合わせることができます。

参照

- TableSchema.SYS_INDEXES 変数 [Ultra Light J]267 ページ

UNIQUE_KEY 変数

インデックスがユニークキーであることを示します。

構文

```
final byte IndexSchema.UNIQUE_KEY
```

備考

この値は、SYS_TABLES システムテーブルの table_flags カラムで他のフラグと論理的に組み合わせることができます。

参照

- TableSchema.SYS_INDEXES 変数 [Ultra Light J]267 ページ

PreparedStatement インターフェイス

SQL クエリを実行して ResultSet オブジェクトを生成するか、準備された SQL 文をデータベースに対して実行するメソッドを提供します。

構文

```
public interface PreparedStatement
```

メンバー

継承されたメンバーを含む PreparedStatement インターフェイスのすべてのメンバー。

名前	説明
close メソッド	PreparedStatement を閉じて、関連付けられているメモリリソースを解放します。
execute メソッド	準備された SQL 文を実行します。
executeQuery メソッド	準備された SQL SELECT 文を実行し、ResultSet オブジェクトを返します。
getBlobOutputStream メソッド	OutputStream オブジェクトを返します。
getClobWriter メソッド	Writer オブジェクトを返します。

名前	説明
getOrdinal メソッド	name で指定された値の (1 から始まる) 順序を返します。
getPlan メソッド	SQL クエリ実行プランのテキストベースの記述を返します。
getPlanTree メソッド	SQL クエリ実行プランのテキストベースの記述をツリー形式で返します。
getResultSet メソッド	準備された SQL 文の ResultSet オブジェクトを返します。
getUpdateCount メソッド	最後の実行文の後に挿入、更新、または削除されたロー数を返します。
hasResultSet メソッド	PreparedStatement オブジェクトに ResultSet オブジェクトが含まれるかどうかを確認します。
set メソッド	SQL 文のホスト変数に値を設定します。
setNull メソッド	SQL 文のホスト変数に NULL 値を設定します。

備考

次の例は、PreparedStatement オブジェクトを実行し、SELECT 文によって ResultSet オブジェクトが作成されたかどうかを確認し、ResultSet オブジェクトをローカル変数に格納し、PreparedStatement を閉じる方法を示しています。

```
// Create a new PreparedStatement object from an existing connection.
String sql_string = "SELECT * FROM SampleTable";
PreparedStatement ps = conn.prepareStatement(sql_string);

// Result returns true if the statement runs successfully.
boolean result = ps.execute();

// Check if the PreparedStatement object contains a ResultSet object.
if (ps.hasResultSet()) {
    // Store the ResultSet in the rs variable.
    ResultSet rs = ps.getResultSet;
}
// Close the PreparedStatement object to release resources.
ps.close();
```

文に式が含まれる場合、カラム名があるところにはどこでもホスト変数が含まれる可能性があります。ホスト変数は、'?' 文字 (名前なしホスト変数) または ":name" (名前付きホスト変数) のいずれかとして入力されます。

次の例には、対象の SQL 文用に準備された PreparedStatement オブジェクトを使って設定できる 2 つのホスト変数があります。

```
SELECT * FROM SampleTable WHERE pk > :bound AND pk < ?
```

参照

- Connection インターフェイス [Ultra Light J]105 ページ
- Connection.prepareStatement メソッド [Ultra Light J]117 ページ

close メソッド

PreparedStatement を閉じて、関連付けられているメモリリソースを解放します。

構文

```
void PreparedStatement.close() throws ULjException
```

備考

このオブジェクトに対してこれ以上メソッドを使用できなくなります。PreparedStatement オブジェクトに ResultSet オブジェクトが含まれている場合は、両方のオブジェクトが閉じます。

execute メソッド

準備された SQL 文を実行します。

構文

```
boolean PreparedStatement.execute() throws ULjException
```

戻り値

文が正常に実行された場合は true、それ以外の場合は false。

参照

- ResultSet インターフェイス [Ultra Light J]194 ページ

executeQuery メソッド

準備された SQL SELECT 文を実行し、ResultSet オブジェクトを返します。

構文

```
ResultSet PreparedStatement.executeQuery() throws ULjException
```

戻り値

準備された SQL SELECT 文のクエリの結果を含む ResultSet オブジェクト。

参照

- ResultSet インターフェイス [Ultra Light J]194 ページ

getBlobOutputStream メソッド

OutputStream オブジェクトを返します。

オーバーロードリスト

名前	説明
getBlobOutputStream(int) メソッド	OutputStream オブジェクトを返します。
getBlobOutputStream(String) メソッド	OutputStream オブジェクトを返します。

getBlobOutputStream(int) メソッド

OutputStream オブジェクトを返します。

構文

```
java.io.OutputStream PreparedStatement.getBlobOutputStream(
    int ordinal
) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのホスト変数の順序を表す 1 から始まる整数。

戻り値

指定された値の OutputStream オブジェクト。

getBlobOutputStream(String) メソッド

OutputStream オブジェクトを返します。

構文

```
java.io.OutputStream PreparedStatement.getBlobOutputStream(
    String name
) throws ULjException
```

パラメーター

- **name**　ホスト変数名を表す文字列値。

戻り値

指定された値の OutputStream オブジェクト。

getClobWriter メソッド

Writer オブジェクトを返します。

オーバーロードリスト

名前	説明
getClobWriter(int) メソッド	Writer オブジェクトを返します。
getClobWriter(String) メソッド	Writer オブジェクトを返します。

getClobWriter(int) メソッド

Writer オブジェクトを返します。

構文

```
java.io.Writer PreparedStatement.getClobWriter(
    int ordinal
) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのホスト変数の順序を表す 1 から始まる整数。

戻り値

指定された値の Writer オブジェクト。

getClobWriter(String) メソッド

Writer オブジェクトを返します。

構文

```
java.io.Writer PreparedStatement.getClobWriter(
    String name
) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。

戻り値

指定された値の Writer オブジェクト。

getOrdinal メソッド

name で指定された値の (1 から始まる) 順序を返します。

構文

```
int PreparedStatement.getOrdinal(String name) throws ULjException
```

パラメーター

- **name**　テーブルのカラム名を表す文字列値。

戻り値

name で指定された値の (1 から始まる) 順序。

getPlan メソッド

SQL クエリ実行プランのテキストベースの記述を返します。

構文

```
String PreparedStatement.getPlan() throws ULjException
```

戻り値

プランの String 表現。

備考

このメソッドは、開発中の使用を目的とします。

このプランには、getPlanTree メソッドで示される情報と同じものが含まれます。違いは表示形式です。

プランがない場合は、空の文字列を返します。準備された文が SQL クエリの場合には、プランが存在します。

関連するクエリの実行前にプランが取得された場合は、クエリの実行に使用される操作がプランに表示されます。また、クエリの実行後にプランが取得された場合は、各操作で生成されるロー数も表示されます。このプランを使用して、クエリの実行に関する理解を深めることができます。

次に、String として表されたプランの例を示します。構造を表す '|' 文字を使用して複数行に表示されます。

```
SELECT * FROM tab1, tab2 WHERE col1 > pk2
row: 2 20 10 banana
row: 3 30 10 banana
row: 4 40 10 banana
row: 4 40 30 peach
row: 5 50 10 banana
row: 5 50 30 peach
```

```
row: 5 50 40 apple
plan: root:7(inner-join:7(table-scan:5[tab1,prime_key],index-scan:7[tab2,prime_key]))
```

参照

- PreparedStatement.getPlanTree メソッド [Ultra Light J]184 ページ

getPlanTree メソッド

SQL クエリ実行プランのテキストベースの記述をツリー形式で返します。

構文

```
String PreparedStatement.getPlanTree() throws ULjException
```

戻り値

ツリーで表されるプランの String 表現。

備考

このメソッドは、開発中の使用を目的とします。

このプランには、getPlan メソッドで示される情報と同じものが含まれます。違いは表示形式です。

プランがない場合は、空の文字列を返します。準備された文が SQL クエリの場合には、プランが存在します。

関連するクエリの実行前にプランが取得された場合は、クエリの実行に使用される操作がプランに表示されます。また、クエリの実行後にプランが取得された場合は、各操作で生成されるロー数も表示されます。このプランを使用して、クエリの実行に関する理解を深めることができます。

次に、String として表されたプランの例を示します。構造を表す '|' 文字を使用して複数行に表示されます。

```
SELECT * FROM tab1, tab2 WHERE col1 > pk2
row: 2 20 10 banana
row: 3 30 10 banana
row: 4 40 10 banana
row: 4 40 30 peach
row: 5 50 10 banana
row: 5 50 30 peach
row: 5 50 40 apple
plan:
 root:7
 |
 inner-join:7
 | |
 | index-scan:7[tab2,prime_key]
 |
 table-scan:5[tab1,prime_key]
```


参照

- PreparedStatement.getPlan メソッド [Ultra Light J]183 ページ

getResultSet メソッド

準備された SQL 文の ResultSet オブジェクトを返します。

構文

```
ResultSet PreparedStatement.getResultSet() throws ULjException
```

戻り値

準備された SQL 文のクエリの結果を含む ResultSet オブジェクト。

参照

- ResultSet インターフェイス [Ultra Light J]194 ページ

getUpdateCount メソッド

最後の実行文の後に挿入、更新、または削除されたロー数を返します。

構文

```
int PreparedStatement.getUpdateCount() throws ULjException
```

戻り値

文で変更を実行できない場合は -1 を返し、そうでない場合は変更されたロー数を返します。

hasResultSet メソッド

PreparedStatement オブジェクトに ResultSet オブジェクトが含まれるかどうかを確認します。

構文

```
boolean PreparedStatement.hasResultSet() throws ULjException
```

戻り値

ResultSet が見つかった場合は true、それ以外の場合は false。

参照

- ResultSet インターフェイス [Ultra Light J]194 ページ

set メソッド

SQL 文のホスト変数に値を設定します。

オーバーロードリスト

名前	説明
set(int, boolean) メソッド	SQL 文内の ordinal で定義されたホスト変数に boolean 値を設定します。
set(int, byte[]) メソッド	SQL 文内の ordinal で定義されたホスト変数に byte 配列値を設定します。
set(int, Date) メソッド	SQL 文内の ordinal で定義されたホスト変数に java.util.Date を設定します。
set(int, DecimalNumber) メソッド	SQL 文内の ordinal で定義されたホスト変数に DecimalNumber オブジェクトを設定します。
set(int, double) メソッド	SQL 文内の ordinal で定義されたホスト変数に double 値を設定します。
set(int, float) メソッド	SQL 文内の ordinal で定義されたホスト変数に float 値を設定します。
set(int, int) メソッド	SQL 文内の ordinal で定義されたホスト変数に int 値を設定します。
set(int, long) メソッド	SQL 文内の ordinal で定義されたホスト変数に long integer 値を設定します。
set(int, String) メソッド	SQL 文内の ordinal で定義されたホスト変数に String 値を設定します。
set(int, UUIDValue) メソッド	SQL 文内の ordinal で定義されたホスト変数に UUIDValue 値を設定します。
set(String, boolean) メソッド	SQL 文内の name で定義されたホスト変数に boolean 値を設定します。
set(String, byte[]) メソッド	SQL 文内の name で定義されたホスト変数に byte 配列値を設定します。
set(String, Date) メソッド	SQL 文内の namel で定義されたホスト変数に java.util.Date を設定します。

名前	説明
set(String, DecimalNumber) メソッド	SQL 文内の name で定義されたホスト変数に DecimalNumber をオブジェクトを設定します。
set(String, double) メソッド	SQL 文内の name で定義されたホスト変数に double 値を設定します。
set(String, float) メソッド	SQL 文内の name で定義されたホスト変数に float 値を設定します。
set(String, int) メソッド	SQL 文内の name で定義されたホスト変数に int 値を設定します。
set(String, long) メソッド	SQL 文内の name で定義されたホスト変数に long integer 値を設定します。
set(String, String) メソッド	SQL 文内の name で定義されたホスト変数に String 値を設定します。
set(String, UUIDValue) メソッド	SQL 文内の name で定義されたホスト変数に UUIDValue 値を設定します。

set(int, boolean) メソッド

SQL 文内の ordinal で定義されたホスト変数に boolean 値を設定します。

構文

```
void PreparedStatement.set(
    int ordinal,
    boolean value
) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのホスト変数の順序を表す 1 から始まる整数。
- **value**　設定する値。

set(int, byte[]) メソッド

SQL 文内の ordinal で定義されたホスト変数に byte 配列値を設定します。

構文

```
void PreparedStatement.set(
    int ordinal,
```

```
    byte[] value
) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのホスト変数の順序を表す 1 から始まる整数。
- **value** 設定する値。

set(int, Date) メソッド

SQL 文内の ordinal で定義されたホスト変数に java.util.Date を設定します。

構文

```
void PreparedStatement.set(
    int ordinal,
    java.util.Date value
) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのホスト変数の順序を表す 1 から始まる整数。
- **value** 設定する値。

set(int, DecimalNumber) メソッド

SQL 文内の ordinal で定義されたホスト変数に DecimalNumber オブジェクトを設定します。

構文

```
void PreparedStatement.set(
    int ordinal,
    DecimalNumber value
) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのホスト変数の順序を表す 1 から始まる整数。
- **value** 設定する DecimalNumber の値。

set(int, double) メソッド

SQL 文内の ordinal で定義されたホスト変数に double 値を設定します。

構文

```
void PreparedStatement.set(
    int ordinal,
    double value
) throws ULjException
```


パラメーター

- **ordinal**　SQL 文でのホスト変数の順序を表す 1 から始まる整数。
- **value**　設定する値。

set(int, float) メソッド

SQL 文内の ordinal で定義されたホスト変数に float 値を設定します。

構文

```
void PreparedStatement.set(int ordinal, float value) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのホスト変数の順序を表す 1 から始まる整数。
- **value**　設定する値。

set(int, int) メソッド

SQL 文内の ordinal で定義されたホスト変数に int 値を設定します。

構文

```
void PreparedStatement.set(int ordinal, int value) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのホスト変数の順序を表す 1 から始まる整数。
- **value**　設定する値。

set(int, long) メソッド

SQL 文内の ordinal で定義されたホスト変数に long integer 値を設定します。

構文

```
void PreparedStatement.set(int ordinal, long value) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのホスト変数の順序を表す 1 から始まる整数。
- **value**　設定する値。

set(int, String) メソッド

SQL 文内の ordinal で定義されたホスト変数に String 値を設定します。

構文

```
void PreparedStatement.set(
    int ordinal,
    String value
) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのホスト変数の順序を表す 1 から始まる整数。
- **value** 設定する値。

set(int, UUIDValue) メソッド

SQL 文内の ordinal で定義されたホスト変数に UUIDValue 値を設定します。

構文

```
void PreparedStatement.set(
    int ordinal,
    UUIDValue value
) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのホスト変数の順序を表す 1 から始まる整数。
- **value** 設定する値。

set(String, boolean) メソッド

SQL 文内の name で定義されたホスト変数に boolean 値を設定します。

構文

```
void PreparedStatement.set(
    String name,
    boolean value
) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。
- **value** 設定する値。

set(String, byte[]) メソッド

SQL 文内の name で定義されたホスト変数に byte 配列値を設定します。

構文

```
void PreparedStatement.set(
    String name,
    byte[] value
) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。
- **value** 設定する値。

set(String, Date) メソッド

SQL 文内の namel で定義されたホスト変数に java.util.Date を設定します。

構文

```
void PreparedStatement.set(
    String name,
    java.util.Date value
) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。
- **value** 設定する値。

set(String, DecimalNumber) メソッド

SQL 文内の name で定義されたホスト変数に DecimalNumber をオブジェクトを設定します。

構文

```
void PreparedStatement.set(
    String name,
    DecimalNumber value
) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。
- **value** 設定する DecimalNumber の値。

set(String, double) メソッド

SQL 文内の name で定義されたホスト変数に double 値を設定します。

構文

```
void PreparedStatement.set(
    String name,
    double value
) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。
- **value** 設定する値。

set(String, float) メソッド

SQL 文内の name で定義されたホスト変数に float 値を設定します。

構文

```
void PreparedStatement.set(String name, float value) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。
- **value** 設定する値。

set(String, int) メソッド

SQL 文内の name で定義されたホスト変数に int 値を設定します。

構文

```
void PreparedStatement.set(String name, int value) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。
- **value** 設定する値。

set(String, long) メソッド

SQL 文内の name で定義されたホスト変数に long integer 値を設定します。

構文

```
void PreparedStatement.set(String name, long value) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。
- **value** 設定する値。

set(String, String) メソッド

SQL 文内の name で定義されたホスト変数に String 値を設定します。

構文

```
void PreparedStatement.set(
    String name,
    String value
) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。
- **value** 設定する値。

set(String, UUIDValue) メソッド

SQL 文内の name で定義されたホスト変数に UUIDValue 値を設定します。

構文

```
void PreparedStatement.set(
    String name,
    UUIDValue value
) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。
- **value** 設定する値。

setNull メソッド

SQL 文のホスト変数に NULL 値を設定します。

オーバーロードリスト

名前	説明
setNull(int) メソッド	SQL 文のホスト変数に NULL 値を設定します。
setNull(String) メソッド	SQL 文内の name で定義されたホスト変数に NULL 値を設定します。

setNull(int) メソッド

SQL 文のホスト変数に NULL 値を設定します。

構文

```
void PreparedStatement.setNull(int ordinal) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのホスト変数の順序を表す 1 から始まる整数。

setNull(String) メソッド

SQL 文内の name で定義されたホスト変数に NULL 値を設定します。

構文

```
void PreparedStatement.setNull(String name) throws ULjException
```

パラメーター

- **name** ホスト変数名を表す文字列値。

ResultSet インターフェイス

テーブルをローごとにトラバースし、カラムデータにアクセスするメソッドを提供します。

構文

```
public interface ResultSet
```

メンバー

継承されたメンバーを含む ResultSet インターフェイスのすべてのメンバー。

名前	説明
afterLast メソッド [Android]	カーソルを最後のローの後に移動します。
beforeFirst メソッド [Android]	カーソルを最初のローの前に移動します。
close メソッド	ResultSet オブジェクトを閉じて、関連付けられているメモリリソースを解放します。
first メソッド [Android]	カーソルを最初のローに移動します。
getBlobInputStream メソッド	ファイルベースの long binary を含めた、long binary 型の InputStream オブジェクトを返します。

名前	説明
getBoolean メソッド	boolean 値を返します。
getBytes メソッド	byte 配列を返します。
getClobReader メソッド	Reader オブジェクトを返します。
getDate メソッド	java.util.Date オブジェクトを返します。
getDecimalNumber メソッド	DecimalNumber オブジェクトを返します。
getDouble メソッド	カラム番号に基づいた double 値を返します。
getFloat メソッド	float 値を返します。
getInt メソッド	integer 値を返します。
getLong メソッド	long integer 値を返します。
getOrdinal メソッド	String で表現された値の (1 から始まる) 順序を返します。
getResultSetMetadata メソッド	ResultSet オブジェクトのメタデータを含む ResultSetMetadata オブジェクトを返します。
getRowCount メソッド [Android]	テーブルのローの数を取得します。
getSize メソッド	結果セットカラムの実際のサイズを取得します。
getString メソッド	String 値を返します。
getUUIDValue メソッド	UUIDValue オブジェクトを返します。
isNull メソッド	指定されたカラムの値が NULL かどうかをテストします。
last メソッド [Android]	カーソルを最後のローに移動します。
next メソッド	ResultSet オブジェクト内の次のデータローをフェッチします。
previous メソッド	ResultSet オブジェクト内の前のデータローをフェッチします。
relative メソッド [Android]	カーソルを、現在のカーソルの位置から、offset で指定したロー数分移動します。

備考

ResultSet オブジェクトは、SQL SELECT 文により、PreparedStatement オブジェクト上で execute または executeQuery メソッドが呼び出されたときに生成されます。

次の例は、ResultSet オブジェクトでローをフェッチし、指定したカラムのデータにアクセスする方法を示しています。

```
// Define a new SQL SELECT statement.
String sql_string = "SELECT column1, column2 FROM SampleTable";

// Create a new PreparedStatement from an existing connection.
PreparedStatement ps = conn.prepareStatement(sql_string);

// Create a new ResultSet to contain the query results of the SQL statement.
ResultSet rs = ps.executeQuery();

// Check if the PreparedStatement contains a ResultSet.
if (ps.hasResultSet()) {
    // Retrieve the column1 value from the first row using getString.
    String row1_col1 = rs.getString(1);
    // Get the next row in the table.
    if (rs.next) {
        // Retrieve the value of column1 from the second row.
        String row2_col1 = rs.getString(1);
    }
}

rs.close();
ps.close();
```

参照

- PreparedStatement インターフェイス [Ultra Light J]178 ページ
- PreparedStatement.execute メソッド [Ultra Light J]180 ページ
- PreparedStatement.executeQuery メソッド [Ultra Light J]180 ページ
- Connection インターフェイス [Ultra Light J]105 ページ

afterLast メソッド [Android]

カーソルを最後のローの後に移動します。

構文

```
boolean ResultSet.afterLast() throws ULjException
```

戻り値

成功した場合は true、失敗した場合は false。

beforeFirst メソッド [Android]

カーソルを最初のローの前に移動します。

構文

```
boolean ResultSet.beforeFirst() throws ULjException
```

戻り値

成功した場合は true、失敗した場合は false。

close メソッド

ResultSet オブジェクトを閉じて、関連付けられているメモリリソースを解放します。

構文

```
void ResultSet.close() throws ULjException
```

備考

以降で、閉じた ResultSet オブジェクトからローをフェッチしようとするとエラーが発生します。

first メソッド [Android]

カーソルを最初のローに移動します。

構文

```
boolean ResultSet.first() throws ULjException
```

戻り値

成功した場合は true、失敗した場合は false。

getBlobInputStream メソッド

ファイルベースの long binary を含めた、long binary 型の InputStream オブジェクトを返します。

オーバーロードリスト

名前	説明
getBlobInputStream(int) メソッド	ファイルベースの long binary を含めた、long binary 型の InputStream オブジェクトを返します。
getBlobInputStream(String) メソッド	ファイルベースの long binary を含めた、long binary 型の InputStream オブジェクトを返します。

getBlobInputStream(int) メソッド

ファイルベースの long binary を含めた、long binary 型の InputStream オブジェクトを返します。

構文

```
java.io.InputStream ResultSet.getBlobInputStream(
    int ordinal
) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の InputStream オブジェクトの表現。

getBlobInputStream(String) メソッド

ファイルベースの long binary を含めた、long binary 型の InputStream オブジェクトを返します。

構文

```
java.io.InputStream ResultSet.getBlobInputStream(
    String name
) throws ULjException
```

パラメーター

- **name** テーブルのカラム名を表す文字列値。

戻り値

指定された値の InputStream オブジェクトの表現。

getBoolean メソッド

boolean 値を返します。

オーバーロードリスト

名前	説明
getBoolean(int) メソッド	boolean 値を返します。
getBoolean(String) メソッド	boolean 値を返します。

getBoolean(int) メソッド

boolean 値を返します。

構文

```
boolean ResultSet.getBoolean(int ordinal) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の boolean 表現。

getBoolean(String) メソッド

boolean 値を返します。

構文

```
boolean ResultSet.getBoolean(String name) throws ULjException
```

パラメーター

- **name**　ResultSet オブジェクトでテーブルのカラム名を表す文字列。

戻り値

指定された値の boolean 表現。

getBytes メソッド

byte 配列を返します。

オーバーロードリスト

名前	説明
getBytes(int) メソッド	byte 配列を返します。
getBytes(String) メソッド	byte 配列を返します。

getBytes(int) メソッド

byte 配列を返します。

構文

```
byte[] ResultSet.getBytes(int ordinal) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の byte 配列表現。

getBytes(String) メソッド

byte 配列を返します。

構文

```
byte[] ResultSet.getBytes(String name) throws ULjException
```

パラメーター

- **name** テーブルのカラム名を表す文字列。

戻り値

指定された値の byte 配列表現。

getClobReader メソッド

Reader オブジェクトを返します。

オーバーロードリスト

名前	説明
getClobReader(int) メソッド	Reader オブジェクトを返します。
getClobReader(String) メソッド	Reader オブジェクトを返します。

getClobReader(int) メソッド

Reader オブジェクトを返します。

構文

```
java.io.Reader ResultSet.getClobReader(int ordinal) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の Reader オブジェクト表現。

getClobReader(String) メソッド

Reader オブジェクトを返します。

構文

```
java.io.Reader ResultSet.getClobReader(String name) throws ULjException
```

パラメーター

- **name**　テーブルのカラム名を表す文字列。

戻り値

指定された値の Reader オブジェクト表現。

getDate メソッド

java.util.Date オブジェクトを返します。

オーバーロードリスト

名前	説明
getDate(int) メソッド	java.util.Date オブジェクトを返します。
getDate(String) メソッド	java.util.Date オブジェクトを返します。

getDate(int) メソッド

java.util.Date オブジェクトを返します。

構文

```
java.util.Date ResultSet.getDate(int ordinal) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の java.util.Date オブジェクト表現。

getDate(String) メソッド

java.util.Date オブジェクトを返します。

構文

```
java.util.Date ResultSet.getDate(String name) throws ULjException
```

パラメーター

- **name**　テーブルのカラム名を表す文字列。

戻り値

指定された値の java.util.date オブジェクト表現。

getDecimalNumber メソッド

DecimalNumber オブジェクトを返します。

オーバーロードリスト

名前	説明
getDecimalNumber(int) メソッド	DecimalNumber オブジェクトを返します。
getDecimalNumber(String) メソッド	DecimalNumber オブジェクトを返します。

getDecimalNumber(int) メソッド

DecimalNumber オブジェクトを返します。

構文

```
DecimalNumber ResultSet.getDecimalNumber(
    int ordinal
) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の DecimalNumber オブジェクト表現。

参照

- DecimalNumber インターフェイス [Ultra Light J]139 ページ

getDecimalNumber(String) メソッド

DecimalNumber オブジェクトを返します。

構文

```
DecimalNumber ResultSet.getDecimalNumber(
    String name
) throws ULjException
```

パラメーター

- **name**　テーブルのカラム名を表す文字列。

戻り値

指定された値の DecimalNumber オブジェクト表現。

参照

- DecimalNumber インターフェイス [Ultra Light J]139 ページ

getDouble メソッド

カラム番号に基づいた double 値を返します。

オーバーロードリスト

名前	説明
getDouble(int) メソッド	カラム番号に基づいた double 値を返します。
getDouble(String) メソッド	DecimalNumber を返します。

getDouble(int) メソッド

カラム番号に基づいた double 値を返します。

構文

```
double ResultSet.getDouble(int ordinal) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の double 表現。

getDouble(String) メソッド

DecimalNumber を返します。

構文

```
double ResultSet.getDouble(String name) throws ULjException
```

パラメーター

- **name**　テーブルのカラム名を表す文字列。

戻り値

指定された値の double 表現。

getFloat メソッド

float 値を返します。

オーバーロードリスト

名前	説明
getFloat(int) メソッド	float 値を返します。
getFloat(String) メソッド	float 値を返します。

getFloat(int) メソッド

float 値を返します。

構文

```
float ResultSet.getFloat(int ordinal) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の float 表現。

getFloat(String) メソッド

float 値を返します。

構文

```
float ResultSet.getFloat(String name) throws ULjException
```

パラメーター

- **name** テーブルのカラム名を表す文字列。

戻り値

指定された値の float 表現。

getInt メソッド

integer 値を返します。

オーバーロードリスト

名前	説明
getInt(int) メソッド	integer 値を返します。
getInt(String) メソッド	integer 値を返します。

getInt(int) メソッド

integer 値を返します。

構文

```
int ResultSet.getInt(int ordinal) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の integer 表現。

getInt(String) メソッド

integer 値を返します。

構文

```
int ResultSet.getInt(String name) throws ULjException
```

パラメーター

- **name**　テーブルのカラム名を表す文字列。

戻り値

指定された値の integer 表現。

getLong メソッド

long integer 値を返します。

オーバーロードリスト

名前	説明
getLong(int) メソッド	long integer 値を返します。
getLong(String) メソッド	long integer 値を返します。

getLong(int) メソッド

long integer 値を返します。

構文

```
long ResultSet.getLong(int ordinal) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の long integer 表現。

getLong(String) メソッド

long integer 値を返します。

構文

```
long ResultSet.getLong(String name) throws ULjException
```

パラメーター

- **name** テーブルのカラム名を表す文字列。

戻り値

指定された値の long integer 表現。

getOrdinal メソッド

String で表現された値の (1 から始まる) 順序を返します。

構文

```
int ResultSet.getOrdinal(String name) throws ULjException
```

パラメーター

- **name** テーブルのカラム名を表す文字列。

戻り値

順序の値。

getResultSetMetadata メソッド

ResultSet オブジェクトのメタデータを含む ResultSetMetadata オブジェクトを返します。

構文

```
ResultSetMetadata ResultSet.getResultSetMetadata() throws ULjException
```

戻り値

ResultSetMetadata オブジェクト。

getRowCount メソッド [Android]

テーブルのローの数を取得します。

構文

```
long ResultSet.getRowCount(long threshold) throws ULjException
```

パラメーター

<varlistentry> **threshold** カウントするローの数の制限。0 はエラーがないことを示します。
</varlistentry>

戻り値

テーブル内のローの数。

備考

このメソッドは、"SELECT COUNT(*) FROM table" を実行するのと同じです。

getSize メソッド

結果セットカラムの実際のサイズを取得します。

オーバーロードリスト

名前	説明
getSize(int) メソッド	結果セットカラムの実際のサイズを取得します。

名前	説明
getSize(String) メソッド	結果セットカラムの実際のサイズを取得します。

getSize(int) メソッド

結果セットカラムの実際のサイズを取得します。

構文

```
int ResultSet.getSize(int ordinal) throws ULjException
```

パラメーター

- **ordinal** SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

結果セットカラムの実際のサイズ。

getSize(String) メソッド

結果セットカラムの実際のサイズを取得します。

構文

```
int ResultSet.getSize(String name) throws ULjException
```

パラメーター

- **name** テーブルのカラム名を表す文字列。

戻り値

結果セットカラムの実際のサイズ。

getString メソッド

String 値を返します。

オーバーロードリスト

名前	説明
getString(int) メソッド	String 値を返します。
getString(String) メソッド	String 値を返します。

getString(int) メソッド

String 値を返します。

構文

```
String ResultSet.getString(int ordinal) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の String 表現。

getString(String) メソッド

String 値を返します。

構文

```
String ResultSet.getString(String name) throws ULjException
```

パラメーター

- **name**　テーブルのカラム名を表す文字列。

戻り値

指定された値の String 表現。

getUUIDValue メソッド

UUIDValue オブジェクトを返します。

オーバーロードリスト

名前	説明
getUUIDValue(int) メソッド	UUIDValue オブジェクトを返します。
getUUIDValue(String) メソッド	UUIDValue オブジェクトを返します。

getUUIDValue(int) メソッド

UUIDValue オブジェクトを返します。

構文

```
UUIDValue ResultSet.getUUIDValue(int ordinal) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の UUIDValue オブジェクト表現。

参照

- UUIDValue インターフェイス [Ultra Light J]275 ページ

getUUIDValue(String) メソッド

UUIDValue オブジェクトを返します。

構文

```
UUIDValue ResultSet.getUUIDValue(String name) throws ULjException
```

パラメーター

- **name**　テーブルのカラム名を表す文字列。

戻り値

指定された値の UUIDValue オブジェクト表現。

isNull メソッド

指定されたカラムの値が NULL かどうかをテストします。

オーバーロードリスト

名前	説明
isNull(int) メソッド	指定されたカラムの値が NULL かどうかをテストします。
isNull(String) メソッド	指定されたカラム名が NULL かどうかをテストします。

isNull(int) メソッド

指定されたカラムの値が NULL かどうかをテストします。

構文

```
boolean ResultSet.isNull(int ordinal) throws ULjException
```

パラメーター

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

値が null の場合は true、それ以外の場合は false。

isNull(String) メソッド

指定されたカラム名が NULL かどうかをテストします。

構文

```
boolean ResultSet.isNull(String name) throws ULjException
```

パラメーター

- **name**　テーブルのカラム名を表す文字列。

戻り値

値が null の場合は true、それ以外の場合は false。

last メソッド [Android]

カーソルを最後のローに移動します。

構文

```
boolean ResultSet.last() throws ULjException
```

戻り値

成功した場合は true、失敗した場合は false。

next メソッド

ResultSet オブジェクト内の次のデータローをフェッチします。

構文

```
boolean ResultSet.next() throws ULjException
```

戻り値

次のローが正常にフェッチされた場合は true、それ以外の場合は false。

参照

- ResultSetMetadata インターフェイス [Ultra Light J]212 ページ

previous メソッド

ResultSet オブジェクト内の前のデータローをフェッチします。

構文

```
boolean ResultSet.previous() throws ULjException
```

戻り値

前のローが正常にフェッチされた場合は true、それ以外の場合は false。

relative メソッド [Android]

カーソルを、現在のカーソルの位置から、offset で指定したロー数分移動します。

構文

```
boolean ResultSet.relative(int offset) throws ULjException
```

パラメーター

- **offset**　移動するローの数。

戻り値

成功した場合は true、失敗した場合は false。

ResultSetMetadata インターフェイス

ResultSet オブジェクトに関連付けられ、カラム情報を提供するメソッドが含まれます。

構文

```
public interface ResultSetMetadata
```

メンバー

継承されたメンバーを含む ResultSetMetadata インターフェイスのすべてのメンバー。

名前	説明
getAliasName メソッド	カラムのエイリアス名を返します。
getColumnCount メソッド	ResultSet オブジェクト内のカラムの合計数を返します。
getCorrelationName メソッド	カラムの相関名を返します。

名前	説明
getDomainName メソッド	ドメインの名前を返します。
getDomainPrecision メソッド	ドメイン値の精度を返します。
getDomainScale メソッド	ドメイン値の位取りを返します。
getDomainSize メソッド	ドメイン値のサイズを返します。
getDomainType メソッド	ドメインの型を返します。
getQualifiedName メソッド	カラムの修飾名を返します。
getTableColumnName メソッド	テーブルまたは派生テーブルのカラム名を返します。
getTableName メソッド	カラムのテーブル名を返します。
getWrittenName メソッド	カラムの書き込み名を返します。

備考

このインターフェイスは、ResultSet.getResultSetMetadata メソッドによって取得されます。

ResultSet オブジェクトの選択リストのカラムが簡略名または複合名 (table-name.column-name、correlation-name.column-name) の場合、名前に関する次の情報が存在すると、その情報が抽出されます。

●エイリアス名

●相関名

●名前の修飾バージョン

●テーブル名

●書き込まれる名前

ResultSet オブジェクトの選択リストのカラムごとに、そのカラムのドメインに関する次の情報を取得できます。

●カラムの型：Domain インターフェイスの整数

●ドメインの名前

●ドメインのサイズ (VARCHAR ドメインと BINARY ドメインの場合)

●位取りと精度 (NUMERIC ドメインの場合)

参照

- Domain インターフェイス [Ultra Light J]143 ページ
- ResultSet インターフェイス [Ultra Light J]194 ページ
- ResultSet.getResultSetMetadata メソッド [Ultra Light J]207 ページ

getAliasName メソッド

カラムのエイリアス名を返します。

構文

```
String ResultSetMetadata.getAliasName(int column_no) throws ULjException
```

パラメーター

- **column_no**　選択リストの (1 から始まる) カラム番号。

戻り値

カラムにエイリアス名がない場合は NULL、そうでない場合はカラムのエイリアス名を返します。

備考

エイリアス名 ([AS] 名) は、カラムを参照するために指定できます。

getColumnCount メソッド

ResultSet オブジェクト内のカラムの合計数を返します。

構文

```
int ResultSetMetadata.getColumnCount() throws ULjException
```

戻り値

カラムの数。

getCorrelationName メソッド

カラムの相関名を返します。

構文

```
String ResultSetMetadata.getCorrelationName(
    int column_no
) throws ULjException
```


パラメーター

- **column_no**　選択リストの (1 から始まる) カラム番号。

戻り値

カラムに相関名がない場合は NULL、そうでない場合はカラムの相関名を返します。

備考

FROM 句で指定した相関名 ([AS] correlation-name) により、派生テーブルなどのテーブル表現を指定します。

getDomainName メソッド

ドメインの名前を返します。

構文

```
String ResultSetMetadata.getDomainName(
    int column_no
) throws ULjException
```

パラメーター

- **column_no**　選択リストの (1 から始まる) カラム番号。

戻り値

ドメイン名。

getDomainPrecision メソッド

ドメイン値の精度を返します。

構文

```
int ResultSetMetadata.getDomainPrecision(
    int column_no
) throws ULjException
```

パラメーター

- **column_no**

戻り値

精度。

getDomainScale メソッド

ドメイン値の位取りを返します。

構文

```
int ResultSetMetadata.getDomainScale(int column_no) throws ULjException
```

パラメーター

- **column_no**　選択リストの (1 から始まる) カラム番号。

戻り値

位取り。

getDomainSize メソッド

ドメイン値のサイズを返します。

構文

```
int ResultSetMetadata.getDomainSize(int column_no) throws ULjException
```

パラメーター

- **column_no**　選択リストの (1 から始まる) カラム番号。

戻り値

サイズ。

getDomainType メソッド

ドメインの型を返します。

構文

```
short ResultSetMetadata.getDomainType(int column_no) throws ULjException
```

パラメーター

- **column_no**　選択リストの (1 から始まる) カラム番号。

戻り値

整数で表したドメインの型。

getQualifiedName メソッド

カラムの修飾名を返します。

構文

```
String ResultSetMetadata.getQualifiedName(
    int column_no
) throws ULjException
```

パラメーター

- **column_no**　選択リストの (1 から始まる) カラム番号。

戻り値

カラムに修飾名がない場合は NULL、そうでない場合はカラムの修飾名を返します。

備考

ResultSet カラムがテーブルのカラムを参照している場合、相関名 (相関名が指定されていない場合はテーブル名) の後にテーブルのカラム名が続く複合名が返されます。

ResultSet カラムがテーブルのカラムを参照していない場合、エイリアスが指定されていれば、エイリアス名が返されます。

getTableColumnName メソッド

テーブルまたは派生テーブルのカラム名を返します。

構文

```
String ResultSetMetadata.getTableColumnName(
    int column_no
) throws ULjException
```

パラメーター

- **column_no**　選択リストの (1 から始まる) カラム番号。

戻り値

カラムにテーブル名がない場合は NULL、そうでない場合はカラムのテーブル名を返します。

getTableName メソッド

カラムのテーブル名を返します。

構文

```
String ResultSetMetadata.getTableName(int column_no) throws ULjException
```

パラメーター

- **column_no**　選択リストの (1 から始まる) カラム番号。

戻り値

カラムにテーブル名がない場合は NULL、そうでない場合はカラムのテーブル名を返します。

備考

テーブルは、ResultSet カラムが参照するテーブル名 (相関名の可能性あり) です。

getWrittenName メソッド

カラムの書き込み名を返します。

構文

```
String ResultSetMetadata.getWrittenName(
    int column_no
) throws ULjException
```

パラメーター

- **column_no**　選択リストの (1 から始まる) カラム番号。

戻り値

カラムに書き込み名がない場合は NULL、そうでない場合はカラムの書き込み名を返します。

備考

書き込み名は、選択リストに指示カラムとして指定された単純名または複合名です。

SISListener インターフェイス [BlackBerry]

サーバー起動同期のメッセージを受信します。

構文

```
public interface SISListener
```

メンバー

継承されたメンバーを含む SISListener インターフェイスのすべてのメンバー。

名前	説明
startListening メソッド	受信スレッドを作成し、開始します。
stopListening メソッド	受信スレッドを停止します。

備考

アプリケーションは、適切な DatabaseManager.createSISHTTPListener メソッドを使用して、SISListener インターフェイスのインスタンスを作成します。

参照

- DatabaseManager.createSISHTTPListener メソッド [BlackBerry] [Ultra Light J]138 ページ

startListening メソッド

受信スレッドを作成し、開始します。

構文

```
void SISListener.startListening()
```

stopListening メソッド

受信スレッドを停止します。

構文

```
void SISListener.stopListening()
```

SISRequestHandler インターフェイス [BlackBerry]

サーバー起動同期の要求を処理します。

構文

```
public interface SISRequestHandler
```

メンバー

継承されたメンバーを含む SISRequestHandler インターフェイスのすべてのメンバー。

名前	説明
onError メソッド	受信中に発生したサーバー起動同期エラーを処理します。
onRequest メソッド	ワーカースレッド上でサーバー起動同期の要求を処理します。

参照

- SISListener インターフェイス [BlackBerry] [Ultra Light J]218 ページ

onError メソッド

受信中に発生したサーバー起動同期エラーを処理します。

構文

```
void SISRequestHandler.onError(String text)
```

パラメーター

- **text** 例外の string 表現。

備考

受信を停止するには、SISListener インターフェイスの stopListening メソッドを明示的に呼び出します。

onRequest メソッド

ワーカースレッド上でサーバー起動同期の要求を処理します。

構文

```
void SISRequestHandler.onRequest(String text)
```

パラメーター

- **text** 要求によって送信された文字列。

StreamHTTPParms インターフェイス

HTTP を使用して Mobile Link サーバーと通信する方法を定義する HTTP ストリームパラメーターを表します。

構文

```
public interface StreamHTTPParms
```

派生クラス

- StreamHTTPSParms インターフェイス [Ultra Light J]229 ページ

メンバー

継承されたメンバーを含む StreamHTTPParms インターフェイスのすべてのメンバー。

名前	説明
getE2eePublicKey メソッド [Android]	エンドツーエンドのパブリックキーを含むファイルの名前を取得します。
getExtraParameters メソッド [Android]	予備の Mobile Link クライアントネットワークプロトコルオプションを取得します。
getHost メソッド	Mobile Link サーバーのホスト名を返します。
getOutputBufferSize メソッド	データが Mobile Link サーバーに送信される前に格納される出力バッファーのサイズをバイト単位で返します。
getPort メソッド	Mobile Link サーバーへの接続に使用されているポート番号を返します。
getURLSuffix メソッド	Mobile Link サーバーの URL サフィックスを返します。
isRestartable メソッド	再起動可能な HTTP が使用されているかどうかを判断します。
setE2eePublicKey メソッド [Android]	エンドツーエンドのパブリックキーを含むファイルの名前を設定します。
setExtraParameters メソッド [Android]	予備の Mobile Link クライアントネットワークプロトコルオプションを設定します。
setHost メソッド	Mobile Link サーバーのホスト名を設定します。
setOutputBufferSize メソッド	データが Mobile Link サーバーに送信される前に格納される出力バッファーのサイズをバイト単位で設定します。
setPort メソッド	Mobile Link サーバーへの接続に使用するポート番号を設定します。
setRestartable メソッド	再起動可能な HTTP を有効または無効にします。
setURLSuffix メソッド	Mobile Link サーバーに接続するための URL サフィックスを指定します。
setZlibCompression メソッド	ZLIB 圧縮を有効または無効にします。
setZlibDownloadWindowSize メソッド	ZLIB 圧縮のダウンロードウィンドウサイズを設定します。

名前	説明
setZlibUploadWindowSize メソッド	ZLIB 圧縮のアップロードウィンドウサイズを設定します。
zlibCompressionEnabled メソッド	ZLIB 圧縮が有効かどうかを判断します。

備考

次の例では、ホスト名 "MyMLHost" にある Mobile Link サーバーと通信するようにストリームパラメーターを設定しています。サーバーは、パラメーター "-x http(port=1234)" を指定して起動されています。

```
SyncParms syncParms = myConnection.createSyncParms(
        SyncParms.HTTP_STREAM,
        "MyUniqueMLUserID",
        "MyMLScriptVersion"
    );
StreamHTTPParms httpParms = syncParms.getStreamParms();
httpParms.setHost("MyMLHost");
httpParms.setPort(1234);
```

このインターフェイスを実装するインスタンスは、SyncParms.getStreamParms メソッドで返されます。

参照

- SyncParms クラス [Ultra Light J]241 ページ
- SyncParms.getStreamParms メソッド [Ultra Light J]247 ページ

getE2eePublicKey メソッド [Android]

エンドツーエンドのパブリックキーを含むファイルの名前を取得します。

構文

```
String StreamHTTPParms.getE2eePublicKey()
```

戻り値

エンドツーエンドのパブリックキーを含むファイルの名前。

getExtraParameters メソッド [Android]

予備の Mobile Link クライアントネットワークプロトコルオプションを取得します。

構文

```
String StreamHTTPParms.getExtraParameters()
```


戻り値

設定された予備のプロトコルオプション。

getHost メソッド

Mobile Link サーバーのホスト名を返します。

構文

```
String StreamHTTPParms.getHost()
```

戻り値

ホスト名。

参照

- StreamHTTPParms.setHost メソッド [Ultra Light J]225 ページ
- StreamHTTPParms.getPort メソッド [Ultra Light J]223 ページ
- StreamHTTPParms.setPort メソッド [Ultra Light J]226 ページ

getOutputBufferSize メソッド

データが Mobile Link サーバーに送信される前に格納される出力バッファーのサイズをバイト単位で返します。

構文

```
int StreamHTTPParms.getOutputBufferSize()
```

戻り値

バッファーサイズの整数値。

備考

この値を大きくすると、サイズの大きいアップロードの送信に必要なネットワークフラッシュの回数が減る可能性がありますが、メモリの使用量が増えます。HTTP では、フラッシュごとに大容量 (約 250 バイト) の HTTP ヘッダーが送信されるので、フラッシュの回数が減ると帯域幅の使用量が削減されます。

参照

- StreamHTTPParms.setOutputBufferSize メソッド [Ultra Light J]226 ページ

getPort メソッド

Mobile Link サーバーへの接続に使用されているポート番号を返します。

構文

```
int StreamHTTPParms.getPort()
```

戻り値

Mobile Link サーバーのポート番号。

参照

- StreamHTTPParms.setPort メソッド [Ultra Light J]226 ページ

getURLSuffix メソッド

Mobile Link サーバーの URL サフィックスを返します。

構文

```
String StreamHTTPParms.getURLSuffix()
```

戻り値

URL サフィックスを含む文字列値。

参照

- StreamHTTPParms.setURLSuffix メソッド [Ultra Light J]227 ページ

isRestartable メソッド

再起動可能な HTTP が使用されているかどうかを判断します。

構文

```
boolean StreamHTTPParms.isRestartable()
```

戻り値

再起動可能な HTTP が有効である場合は true、それ以外は false。

参照

- StreamHTTPParms.setRestartable メソッド [Ultra Light J]227 ページ

setE2eePublicKey メソッド [Android]

エンドツーエンドのパブリックキーを含むファイルの名前を設定します。

構文

```
void StreamHTTPParms.setE2eePublicKey(String public_key)
```


パラメーター

- **public_key**　暗号化に使用する パブリックキーファイルの名前。

備考

デフォルトでは、この値は null で、エンドツーエンドの暗号化は使用されないことを示します。

Mobile Link クライアントネットワークプロトコルオプションの "e2ee_public_key" に相当します。

setExtraParameters メソッド [Android]

予備の Mobile Link クライアントネットワークプロトコルオプションを設定します。

構文

```
void StreamHTTPParms.setExtraParameters(String parms)
```

パラメーター

- **parms**　セミコロンで区切ったプロトコルオプションのリスト。

備考

これらのオプションは、このクラスのメソッドの結果である設定から構築されるリストに付加されます。

このメソッドによって設定されるオプションにより、他のメソッドによって設定された同じオプションは上書きされます。たとえば、予備のパラメーターに "host=abc" が含まれている場合、setHost("xyz") メソッドが呼び出されると、ホストオプションは "abc" になります。

setHost メソッド

Mobile Link サーバーのホスト名を設定します。

構文

```
void StreamHTTPParms.setHost(String v)
```

パラメーター

- **v**　ホスト名。

備考

デフォルトは NULL で、これは localhost を示します。

参照

- StreamHTTPParms.getHost メソッド [Ultra Light J]223 ページ
- StreamHTTPParms.getPort メソッド [Ultra Light J]223 ページ
- StreamHTTPParms.setPort メソッド [Ultra Light J]226 ページ

setOutputBufferSize メソッド

データが Mobile Link サーバーに送信される前に格納される出力バッファーのサイズをバイト単位で設定します。

構文

```
void StreamHTTPParms.setOutputBufferSize(int size)
```

パラメーター

- **size** 新しいバッファーのサイズ。

備考

デフォルトは Blackberry J2ME 以外では 512 で、Blackberry J2ME では 4096 です。有効な値の範囲は 512 ～ 32768 です。この値を大きくすると Java ランタイムから、Mobile Link サーバーでは処理できないチャンク形式の HTTP が送信される可能性があります。

Mobile Link サーバーから「未知の転送エンコードです」というエラーが出力された場合は、この値を小さくしてみてください。

参照

- StreamHTTPParms.getOutputBufferSize メソッド [Ultra Light J]223 ページ

setPort メソッド

Mobile Link サーバーへの接続に使用するポート番号を設定します。

構文

```
void StreamHTTPParms.setPort(int v)
```

パラメーター

- **v** 1 ～ 65535 のポート番号。範囲外の値はデフォルト値に変更されます。

備考

デフォルトのポートは HTTP 同期の場合は 80、HTTPS 同期の場合は 443 です。

参照

- StreamHTTPParms.getPort メソッド [Ultra Light J]223 ページ

setRestartable メソッド

再起動可能な HTTP を有効または無効にします。

構文

```
void StreamHTTPParms.setRestartable(boolean isRestartable)
```

パラメーター

- **isRestartable**　再起動可能な HTTP を有効にする場合は、true に設定します。デフォルト値は false です。

備考

再開可能な HTTP が有効になっている場合、Ultra Light J では信頼性の低いネットワークでネットワークが頻繁に失敗しないよう、ネットワークの割り込みが許容されます。

再起動可能な HTTP を使用するには、Ultra Light J と Mobile Link サーバーの両方に CR#690250 が適用されている必要があります。

参照

- StreamHTTPParms.isRestartable メソッド [Ultra Light J]224 ページ

setURLSuffix メソッド

Mobile Link サーバーに接続するための URL サフィックスを指定します。

構文

```
void StreamHTTPParms.setURLSuffix(String v)
```

パラメーター

- **v**　URL サフィックスの文字列。

備考

Ultra Light J は次のフォーマットの URL を形成します。

```
[http|https]://host-name:port-number/url-suffix
```

デフォルトでは、*url-suffix* は "Mobilink/" です。「v」 を NULL に設定することによって、URL サフィックスをデフォルトに設定できます。

次のコードは、BlackBerry スマートフォンに対して Wi-Fi 接続だけで接続するように指示する URL サフィックスを指定する方法を示しています。

```
myHTTPParms.setURLSuffix(";deviceside=true;interface=wifi");
```

次のコードは、一部の BlackBerry スマートフォンで必要になることがある BlackBerry Enterprise Server (BES) を使用して HTTPS プロトコル上で同期するように、BlackBerry スマートフォンに指示する URL を指定する方法を示しています。

```
myHTTPParms.setURLSuffix(";EndToEndRequired");
```

参照

- StreamHTTPParms.getURLSuffix メソッド [Ultra Light J]224 ページ

setZlibCompression メソッド

ZLIB 圧縮を有効または無効にします。

構文

```
void StreamHTTPParms.setZlibCompression(boolean enable)
```

パラメーター

- **enable** ZLIB 圧縮を有効にする場合は true、無効にする場合は false に設定します。

備考

デフォルトでは、ZLIB 圧縮は無効です。

Mobile Link クライアントネットワークプロトコルオプションの "compression=zlib" に相当します。

setZlibDownloadWindowSize メソッド

ZLIB 圧縮のダウンロードウィンドウサイズを設定します。

構文

```
void StreamHTTPParms.setZlibDownloadWindowSize(int size)
```

パラメーター

- **size** 圧縮ウィンドウサイズの指定は 9～15 です。このパラメーターは、ウィンドウサイズのバイナリ対数です。(履歴バッファのサイズ)

備考

Mobile Link クライアントネットワークプロトコルオプションの "zlib_download_window_size" に相当します。

setZlibUploadWindowSize メソッド

ZLIB 圧縮のアップロードウィンドウサイズを設定します。

構文

```
void StreamHTTPParms.setZlibUploadWindowSize(int size)
```

パラメーター

- **size**　圧縮ウィンドウサイズの指定は 9～15 です。このパラメーターは、ウィンドウサイズのバイナリ対数です。(履歴バッファのサイズ)

備考

Mobile Link クライアントネットワークプロトコルオプションの "zlib_upload_window_size" に相当します。

zlibCompressionEnabled メソッド

ZLIB 圧縮が有効かどうかを判断します。

構文

```
boolean StreamHTTPParms.zlibCompressionEnabled()
```

戻り値

有効である場合は true、それ以外の場合は false。

StreamHTTPSParms インターフェイス

セキュア HTTPS 接続を使用して Mobile Link サーバーと通信する方法を定義する HTTPS ストリームパラメーターを表します。

構文

```
public interface StreamHTTPSParms
```

基本クラス

- StreamHTTPParms インターフェイス [Ultra Light J]220 ページ

メンバー

継承されたメンバーを含む StreamHTTPSParms インターフェイスのすべてのメンバー。

名前	説明
getCertificateCompany メソッド	セキュア接続の検証に使用する証明書の会社名を返します。
getCertificateName メソッド	セキュア接続の検証に使用する証明書の通称を返します。

名前	説明
getCertificateUnit メソッド	セキュア接続の検証に使用する証明書に記載される部署名を返します。
getE2eePublicKey メソッド [Android]	エンドツーエンドのパブリックキーを含むファイルの名前を取得します。
getExtraParameters メソッド [Android]	予備の Mobile Link クライアントネットワークプロトコルオプションを取得します。
getHost メソッド	Mobile Link サーバーのホスト名を返します。
getOutputBufferSize メソッド	データが Mobile Link サーバーに送信される前に格納される出力バッファーのサイズをバイト単位で返します。
getPort メソッド	Mobile Link サーバーへの接続に使用されているポート番号を返します。
getTrustedCertificates メソッド	安全な同期に使用される信頼できるルート証明書のリストを含むファイルの名前を返します。
getURLSuffix メソッド	Mobile Link サーバーの URL サフィックスを返します。
isRestartable メソッド	再起動可能な HTTP が使用されているかどうかを判断します。
setCertificateCompany メソッド	セキュア接続の検証に使用する証明書の会社名を設定します。
setCertificateName メソッド	セキュア接続の検証に使用する証明書の通称を設定します。
setCertificateUnit メソッド	セキュア接続の検証に使用する証明書に記載される部署名を設定します。
setE2eePublicKey メソッド [Android]	エンドツーエンドのパブリックキーを含むファイルの名前を設定します。
setExtraParameters メソッド [Android]	予備の Mobile Link クライアントネットワークプロトコルオプションを設定します。
setHost メソッド	Mobile Link サーバーのホスト名を設定します。

名前	説明
setOutputBufferSize メソッド	データが Mobile Link サーバーに送信される前に格納される出力バッファーのサイズをバイト単位で設定します。
setPort メソッド	Mobile Link サーバーへの接続に使用するポート番号を設定します。
setRestartable メソッド	再起動可能な HTTP を有効または無効にします。
setTrustedCertificates メソッド	安全な同期に使用される信頼できるルート証明書のリストを含むファイルを設定します。
setURLSuffix メソッド	Mobile Link サーバーに接続するための URL サフィックスを指定します。
setZlibCompression メソッド	ZLIB 圧縮を有効または無効にします。
setZlibDownloadWindowSize メソッド	ZLIB 圧縮のダウンロードウィンドウサイズを設定します。
setZlibUploadWindowSize メソッド	ZLIB 圧縮のアップロードウィンドウサイズを設定します。
zlibCompressionEnabled メソッド	ZLIB 圧縮が有効かどうかを判断します。

備考

次の例では、ホスト名 "MyMLHost" にある Mobile Link サーバーと通信するようにストリームパラメーターを設定しています。サーバーは、パラメーター "-x https(port=1234;certificate=RSAServer.crt;certificate_password=x)" を指定して起動されています。

```
SyncParms syncParms = myConnection.createSyncParms(
      SyncParms.HTTPS_STREAM,
      "MyUniqueMLUserID",
      "MyMLScriptVersion"
   );
StreamHTTPSParms httpsParms =
   (StreamHTTPSParms) syncParms.getStreamParms();
httpsParms.setHost("MyMLHost");
httpsParms.setPort(1234);
```

上記の例では、RSAServer.crt 内の証明書が、クライアントホストまたはデバイスにインストール済みの信頼できるルート証明書のチェーンに追加されていることを前提としています。

J2SE の場合、次のいずれかの方法で、必要な信用されたルート証明書を配備できます。

1. 信頼できるルート証明書を JRE の lib/security/cacerts キーストアにインストールする。

2. Java の keytool ユーティリティを使用して独自のキーストアを構築し、Java システムプロパティ javax.net.ssl.trustStore をその場所に設定する (javax.net.ssl.trustStorePassword メソッドを適切な値に設定する)。

3. setTrustedCertificates(String) パラメーターを使用して、配備された証明書ファイルを参照する。

セキュリティを強化するには、setCertificateName、setCertificateCompany、setCertificateUnit の各メソッドを使用して Mobile Link サーバーの証明書の検証を有効にします。

このインターフェイスを実装するインスタンスは、HTTPS 同期用に SyncParms オブジェクトが作成されたときに、SyncParms.getStreamParms メソッドによって返されます。

参照

- SyncParms クラス [Ultra Light J]241 ページ
- SyncParms.getStreamParms メソッド [Ultra Light J]247 ページ
- StreamHTTPSParms.setCertificateCompany メソッド [Ultra Light J]233 ページ
- StreamHTTPSParms.setCertificateName メソッド [Ultra Light J]233 ページ
- StreamHTTPSParms.setCertificateUnit メソッド [Ultra Light J]234 ページ
- StreamHTTPSParms.setTrustedCertificates メソッド [Ultra Light J]234 ページ

getCertificateCompany メソッド

セキュア接続の検証に使用する証明書の会社名を返します。

構文

```
String StreamHTTPSParms.getCertificateCompany()
```

戻り値

証明書の会社名。

getCertificateName メソッド

セキュア接続の検証に使用する証明書の通称を返します。

構文

```
String StreamHTTPSParms.getCertificateName()
```

戻り値

証明書の名前。

getCertificateUnit メソッド

セキュア接続の検証に使用する証明書に記載される部署名を返します。

構文

```
String StreamHTTPSParms.getCertificateUnit()
```

戻り値

組織単位名。

getTrustedCertificates メソッド

安全な同期に使用される信頼できるルート証明書リストのファイル名を返します。

構文

```
String StreamHTTPSParms.getTrustedCertificates()
```

戻り値

信用されたルート証明書ファイルのファイル名。

参照

- StreamHTTPSParms.setTrustedCertificates メソッド [Ultra Light J]234 ページ

setCertificateCompany メソッド

セキュア接続の検証に使用する証明書の会社名を設定します。

構文

```
void StreamHTTPSParms.setCertificateCompany(String val)
```

パラメーター

- **val**　会社名。

備考

デフォルトは NULL で、この場合、証明書で会社名は検証されません。

setCertificateName メソッド

セキュア接続の検証に使用する証明書の通称を設定します。

構文

```
void StreamHTTPSParms.setCertificateName(String val)
```

パラメーター

- **val**　証明書の通称。

備考

デフォルトは NULL で、この場合、証明書で通称は検証されません。

setCertificateUnit メソッド

セキュア接続の検証に使用する証明書に記載される部署名を設定します。

構文

```
void StreamHTTPSParms.setCertificateUnit(String val)
```

パラメーター

- **val**　会社の組織単位名。

備考

デフォルトは NULL で、この場合、証明書で組織単位名は検証されません。

setTrustedCertificates メソッド

安全な同期に使用される信頼できるルート証明書リストのファイルを設定します。

構文

```
void StreamHTTPSParms.setTrustedCertificates(
    String filename
) throws ULjException
```

パラメーター

- **filename**　信用されたルート証明書のファイル名。

備考

このメソッドは、プラットフォーム上の Java Runtime Environment で許可される任意の X.509 形式をサポートします。J2SE では ルート証明書の格納に JKS KeyStore タイプが使用され、Android スマートフォンでは BKS KeyStore が使用されます。

このメソッドを使用できるのは、J2SE プラットフォームと Android スマートフォン上だけです。

証明書は、次の優先度の規則に従って使用されます。

1. このメソッドが呼び出された場合、指定したファイルの証明書が使用されます。

2. このメソッドが呼び出されず、ulinit または ulload ユーティリティによって証明書がデータベースが設定されている場合、それらの証明書が使用されます。

3. このメソッド、ulinit または ulload ユーティリティのいずれによっても証明書が指定されず、Android を使用している場合は、証明書はオペレーティングシステムの信頼できる証明書ストアから読み込まれます。この証明書ストアは、Web サーバーを安全に管理するために HTTPS を介して接続するときに、Web ブラウザで使用されます。

参照

- StreamHTTPSParms.getTrustedCertificates メソッド [Ultra Light J]233 ページ

SyncObserver インターフェイス

同期の進行状況の情報を受け取ります。

構文

```
public interface SyncObserver
```

メンバー

継承されたメンバーを含む SyncObserver インターフェイスのすべてのメンバー。

名前	説明
syncProgress メソッド	ユーザーに進行状況を通知します。

備考

同期の進行状況レポートを受け取るには、同期を実行する新しいクラスを作成し、SyncParms.setSyncObserver メソッドを使用して実装します。

次に、SyncObserver オブジェクトの簡単な実装例を示します。

```
class MyObserver implements SyncObserver {
  public boolean syncProgress(int state, SyncResult result) {
    System.out.println(
        "sync progress state = " + state
      + " bytes sent = " + result.getSentByteCount()
      + " bytes received = " + result.getReceivedByteCount()
    );
    return false;   // Always continue synchronization.
  }
  public MyObserver() {} // The default constructor.
}
```

上記のクラスは、次のメソッドの呼び出しによって有効にできます。

```
SyncParms.setSyncObserver(new MyObserver());
```

参照

- SyncParms クラス [Ultra Light J]241 ページ
- SyncParms.setSyncObserver メソッド [Ultra Light J]255 ページ

syncProgress メソッド

ユーザーに進行状況を通知します。

構文

```
boolean SyncObserver.syncProgress(int state, SyncResult data)
```

パラメーター

- **state** 同期の現在のステータスを表す SyncOberver.States 定数の 1 つ。
- **data** 同期の最新の結果を含む SyncResult オブジェクト。

戻り値

同期をキャンセルする場合は true、続行する場合は false を返します。

備考

このメソッドは同期中に呼び出されます。

パケットとして通知されるさまざまなステータスが送受信されます。1 つのパケットで複数のテーブルがアップロードまたはダウンロードされることがあるので、任意の同期に対するこのメソッドの呼び出しでは、いくつかのステータスが省略される場合があります。

注意
SyncResult メソッドを除き、syncProgress 呼び出し中に他の Ultra Light J API メソッドを呼び出すことはできません。

参照

- SyncObserver.States インターフェイス [Ultra Light J]236 ページ
- SyncParms.setSyncObserver メソッド [Ultra Light J]255 ページ
- SyncResult クラス [Ultra Light J]258 ページ

SyncObserver.States インターフェイス

observer に通知できる同期ステータスを定義します。

構文

```
public interface SyncObserver.States
```

メンバー

継承されたメンバーを含む SyncObserver.States インターフェイスのすべてのメンバー。

名前	説明
CHECKING_LAST_UPLOAD 変数	前のアップロードのステータスをチェックしています。
COMMITTING_DOWNLOAD 変数	ダウンロードされたローがデータベースにコミットされていることを示します。
CONNECTING 変数	同期を開始していることを示します。
DISCONNECTING 変数	同期ストリームを切断していることを示します。
DONE 変数	同期を終了していることを示します。
ERROR 変数	同期は完了しましたが、エラーが発生したことを示します。
FINISHING_UPLOAD 変数	アップロードの完了処理中であることを示します。
RECEIVING_DATA 変数	スキーマ情報またはローデータが受信されていることを示します。
RECEIVING_TABLE 変数	新しいテーブルがダウンロードされていることを示します。
RECEIVING_UPLOAD_ACK 変数	アップロードの確認がダウンロードされていることを示します。
ROLLING_BACK_DOWNLOAD 変数	ダウンロードされたローがデータベースにコミットされていることを示します。
SENDING_DATA 変数	スキーマ情報またはローデータが送信されていることを示します。
SENDING_DOWNLOAD_ACK 変数	ダウンロード完了の確認が送信されていることを示します。
SENDING_HEADER 変数	同期ストリームが開かれ、ヘッダーが送信されようとしています。
SENDING_SCHEMA 変数	スキーマを送信しています。
SENDING_TABLE 変数	新しいテーブルがアップロードされていることを示します。

名前	説明
STARTING 変数	同期を開始していることを示します。

参照

- SyncParms.setSyncObserver メソッド [Ultra Light J]255 ページ
- SyncObserver インターフェイス [Ultra Light J]235 ページ

CHECKING_LAST_UPLOAD 変数

前のアップロードのステータスをチェックしています。

構文

```
final int SyncObserver.States.CHECKING_LAST_UPLOAD
```

備考

前のアップロードのステータスをチェックしていることを示します。

COMMITTING_DOWNLOAD 変数

ダウンロードされたローがデータベースにコミットされていることを示します。

構文

```
final int SyncObserver.States.COMMITTING_DOWNLOAD
```

CONNECTING 変数

同期を開始していることを示します。

構文

```
final int SyncObserver.States.CONNECTING
```

備考

処理はまだ行われていません。

DISCONNECTING 変数

同期ストリームを切断していることを示します。

構文

```
final int SyncObserver.States.DISCONNECTING
```

DONE 変数

同期を終了していることを示します。

構文

```
final int SyncObserver.States.DONE
```

備考

その他のステータスはレポートされません。

ERROR 変数

同期は完了しましたが、エラーが発生したことを示します。

構文

```
final int SyncObserver.States.ERROR
```

FINISHING_UPLOAD 変数

アップロードの完了処理中であることを示します。

構文

```
final int SyncObserver.States.FINISHING_UPLOAD
```

RECEIVING_DATA 変数

スキーマ情報またはローデータが受信されていることを示します。

構文

```
final int SyncObserver.States.RECEIVING_DATA
```

RECEIVING_TABLE 変数

新しいテーブルがダウンロードされていることを示します。

構文

```
final int SyncObserver.States.RECEIVING_TABLE
```

RECEIVING_UPLOAD_ACK 変数

アップロードの確認がダウンロードされていることを示します。

構文

```
final int SyncObserver.States.RECEIVING_UPLOAD_ACK
```

ROLLING_BACK_DOWNLOAD 変数

ダウンロードされたローがデータベースにコミットされていることを示します。

構文

```
final int SyncObserver.States.ROLLING_BACK_DOWNLOAD
```

備考

ダウンロード中にエラーが発生したため、同期によってダウンロードがロールバックされていることを示します。

SENDING_DATA 変数

スキーマ情報またはローデータが送信されていることを示します。

構文

```
final int SyncObserver.States.SENDING_DATA
```

SENDING_DOWNLOAD_ACK 変数

ダウンロード完了の確認が送信されていることを示します。

構文

```
final int SyncObserver.States.SENDING_DOWNLOAD_ACK
```

SENDING_HEADER 変数

同期ストリームが開かれ、ヘッダーが送信されようとしています。

構文

```
final int SyncObserver.States.SENDING_HEADER
```

備考

同期ストリームが開かれ、ヘッダーが送信されようとしていることを示します。

SENDING_SCHEMA 変数

スキーマを送信しています。

構文

```
final int SyncObserver.States.SENDING_SCHEMA
```

備考

スキーマを送信していることを示します。

SENDING_TABLE 変数

新しいテーブルがアップロードされていることを示します。

構文

```
final int SyncObserver.States.SENDING_TABLE
```

STARTING 変数

同期を開始していることを示します。

構文

```
final int SyncObserver.States.STARTING
```

備考

処理はまだ行われていません。

SyncParms クラス

データベース同期処理中に使用されたパラメーターを保持します。

構文

```
public class SyncParms
```

メンバー

継承されたメンバーを含む SyncParms クラスのすべてのメンバー。

名前	説明
getAcknowledgeDownload メソッド	クライアントがダウンロードの確認を送信しているかどうかを判断します。
getAuthenticationParms メソッド	カスタムのユーザー認証スクリプトに渡されるパラメーターを返します。
getLivenessTimeout メソッド	活性タイムアウトの長さを秒単位で返します。
getNewPassword メソッド	setUserName メソッドで指定されたユーザーの新しい Mobile Link パスワードを返します。
getPassword メソッド	setUserName メソッドで指定されたユーザーの Mobile Link パスワードを返します。
getPublications メソッド	同期させるパブリケーションを返します。
getSendColumnNames メソッド	カラム名が Mobile Link サーバーに送信されている場合は true を返します。
getStreamParms メソッド	同期ストリームを設定するパラメーターを返します。
getSyncObserver メソッド	現在指定されている SyncObserver オブジェクトを返します。
getSyncResult メソッド	同期のステータスを含む SyncResult オブジェクトを返します。
getTableOrder メソッド	統合データベースにテーブルがアップロードされる順序を返します。
getUserName メソッド	Mobile Link サーバーがクライアントをユニークに識別する Mobile Link ユーザー名を返します。
getVersion メソッド	使用するスクリプトバージョンを返します。
isDownloadOnly メソッド	同期がダウンロード専用かどうかを確認します。
isPingOnly メソッド	クライアントが Mobile Link サーバーに対して ping または同期を実行しているかどうかを確認します。

名前	説明
isUploadOnly メソッド	同期がアップロード専用かどうかを確認します。
setAcknowledgeDownload メソッド	クライアントがダウンロードの確認を送信すべきかどうかを指定します。
setAuthenticationParms メソッド	カスタムユーザー認証スクリプト (Mobile Link authenticate_parameters 接続イベント) のパラメーターを指定します。
setDownloadOnly メソッド	同期をダウンロード専用に設定します。
setLivenessTimeout メソッド	活性タイムアウトの長さを秒単位で設定します。
setNewPassword メソッド	setUserName メソッドで指定されたユーザーの新しい Mobile Link パスワードを設定します。
setPassword メソッド	setUserName メソッドで指定されたユーザーの Mobile Link パスワードを設定します。
setPingOnly メソッド	クライアントが Mobile Link サーバーに対して、同期を実行しないで ping を実行するように設定します。
setPublications メソッド	同期させるパブリケーションを設定します。
setSendColumnNames メソッド	同期中にカラム名を Mobile Link サーバーに送信するかどうかを指定します。
setSyncObserver メソッド	同期の進行状況をモニターする SyncObserver オブジェクトを設定します。
setTableOrder メソッド	統合データベースにテーブルがアップロードされる順序を設定します。
setUploadOnly メソッド	同期をアップロード専用に設定します。
setUserName メソッド	Mobile Link サーバーがクライアントをユニークに識別する Mobile Link ユーザー名を設定します。
setVersion メソッド	使用する同期スクリプトを設定します。
HTTP_STREAM 変数	HTTP 同期用の SyncParms オブジェクトを作成します。

名前	説明
HTTPS_STREAM 変数	セキュア HTTPS 同期用の SyncParms オブジェクトを作成します。

備考

このインターフェイスは、Connection.createSyncParms メソッドによって呼び出されます。

同期コマンドは一度に 1 つだけ設定できます。コマンドは、setDownloadOnly、setPingOnly、setUploadOnly の各メソッドを使用して指定します。このいずれかのメソッドを true に設定すると、他のメソッドが false に設定されます。

UserName パラメーターと Version パラメーターは設定する必要があります。UserName はクライアントデータベースごとにユニークである必要があります。

通信ストリームは、getStreamParms メソッドを使用して、SyncParms オブジェクトのタイプに基づいて設定します。たとえば、次のコードでは、HTTP 同期の準備と実行を行っています。

```
SyncParms syncParms = myConnection.createSyncParms(
      SyncParms.HTTP_STREAM,
      "MyUniqueMLUserID",
      "MyMLScriptVersion"
   );
syncParms.setPassword("ThePWDforMyUniqueMLUserID");
syncParms.getStreamParms().setHost("MyMLHost");
myConnection.synchronize(syncParms);
```

「カンマ区切りのリスト」

AuthenticationParms、Publications、TableOrder の各パラメーターはすべて、値のカンマ区切りのリストを含む文字列値を使用して指定します。リスト内の値は一重引用符または二重引用符で囲むことができますが、エスケープ文字はありません。引用符がなかった場合、値の前後のスペースは無視されます。たとえば、次のコードは **Table A**、**Table B,D**、**Table C"** を順に指定します。

```
syncParms.setTableOrder("'Table A',¥"Table B,D",Table C" );
```

参照

- SyncParms.getStreamParms メソッド [Ultra Light J]247 ページ
- SyncParms.setUserName メソッド [Ultra Light J]256 ページ
- Connection.createSyncParms メソッド [Ultra Light J]111 ページ
- StreamHTTPParms インターフェイス [Ultra Light J]220 ページ
- StreamHTTPSParms インターフェイス [Ultra Light J]229 ページ

getAcknowledgeDownload メソッド

クライアントがダウンロードの確認を送信しているかどうかを判断します。

構文

```
abstract boolean SyncParms.getAcknowledgeDownload()
```

戻り値

クライアントがダウンロードの確認を送信している場合は true、それ以外の場合は false。

参照

- SyncParms.setAcknowledgeDownload メソッド [Ultra Light J]250 ページ

getAuthenticationParms メソッド

カスタムのユーザー認証スクリプトに渡されるパラメーターを返します。

構文

```
abstract String SyncParms.getAuthenticationParms()
```

戻り値

認証パラメーターのリスト、またはパラメーターが指定されていない場合は NULL。

参照

- SyncParms.setAuthenticationParms メソッド [Ultra Light J]251 ページ

getLivenessTimeout メソッド

活性タイムアウトの長さを秒単位で返します。

構文

```
abstract int SyncParms.getLivenessTimeout()
```

戻り値

タイムアウト。

参照

- SyncParms.setLivenessTimeout メソッド [Ultra Light J]252 ページ

getNewPassword メソッド

setUserName メソッドで指定されたユーザーの新しい Mobile Link パスワードを返します。

構文

```
abstract String SyncParms.getNewPassword()
```

戻り値

次の同期後に設定される新しいパスワード。

参照

- SyncParms.setUserName メソッド [Ultra Light J]256 ページ
- SyncParms.setNewPassword メソッド [Ultra Light J]252 ページ

getPassword メソッド

setUserName メソッドで指定されたユーザーの Mobile Link パスワードを返します。

構文

```
abstract String SyncParms.getPassword()
```

戻り値

Mobile Link ユーザーのパスワード。

参照

- SyncParms.setPassword メソッド [Ultra Light J]253 ページ

getPublications メソッド

同期させるパブリケーションを返します。

構文

```
abstract String SyncParms.getPublications()
```

戻り値

同期するパブリケーションのセット。

参照

- SyncParms.setPublications メソッド [Ultra Light J]254 ページ

getSendColumnNames メソッド

カラム名が Mobile Link サーバーに送信されている場合は true を返します。

構文

```
abstract boolean SyncParms.getSendColumnNames()
```


戻り値

カラム名が送信されている場合は true。

参照

- SyncParms.setSendColumnNames メソッド [Ultra Light J]254 ページ

getStreamParms メソッド

同期ストリームを設定するパラメーターを返します。

構文

```
abstract StreamHTTPParms SyncParms.getStreamParms()
```

戻り値

HTTP または HTTPS の同期ストリームのパラメーターを指定する StreamHTTPParms または StreamHTTPSParms オブジェクト。オブジェクトは参照で返されます。

備考

同期ストリームのタイプは、SyncParms オブジェクトの作成時に指定します。

参照

- Connection.createSyncParms メソッド [Ultra Light J]111 ページ
- StreamHTTPParms インターフェイス [Ultra Light J]220 ページ
- StreamHTTPSParms インターフェイス [Ultra Light J]229 ページ

getSyncObserver メソッド

現在指定されている SyncObserver オブジェクトを返します。

構文

```
abstract SyncObserver SyncParms.getSyncObserver()
```

戻り値

SyncObserver オブジェクト、または observer が指定されていない場合は null。

参照

- SyncParms.setSyncObserver メソッド [Ultra Light J]255 ページ

getSyncResult メソッド

同期のステータスを含む SyncResult オブジェクトを返します。

構文

```
abstract SyncResult SyncParms.getSyncResult()
```

戻り値

前回の Connection.synchronize メソッドの呼び出しの結果を表す SyncResult オブジェクト。

備考

次の例は、前回の Connection.synchronize メソッド呼び出しの結果セットを取得する方法を示しています。

```
conn.synchronize( mySyncParms );
SyncResult result = mySyncParms.getSyncResult();
display(
    "*** Synchronized *** sent=" + result.getSentRowCount()
    + ", received=" + result.getReceivedRowCount()
);
```

注意
このメソッドは、前回の SYNCHRONIZE SQL 文の結果を返しません。前回の SYNCHRONIZE SQL 文の SyncResult オブジェクトを取得するには、渡された Connection オブジェクトで getSyncResult メソッドを使用します。

参照

- SyncResult クラス [Ultra Light J]258 ページ
- Connection.getSyncResult メソッド [Ultra Light J]116 ページ

getTableOrder メソッド

統合データベースにテーブルがアップロードされる順序を返します。

構文

```
abstract String SyncParms.getTableOrder()
```

戻り値

テーブル名のカンマ区切りのリスト、またはテーブルの順序が指定されていない場合は null。カンマ区切りのリストの詳細については、クラスの説明を参照してください。

参照

- SyncParms.setTableOrder メソッド [Ultra Light J]255 ページ

getUserName メソッド

Mobile Link サーバーがクライアントをユニークに識別する Mobile Link ユーザー名を返します。

構文

```
abstract String SyncParms.getUserName()
```

戻り値

Mobile Link ユーザー名。

参照

- SyncParms.setUserName メソッド [Ultra Light J]256 ページ

getVersion メソッド

使用するスクリプトバージョンを返します。

構文

```
abstract String SyncParms.getVersion()
```

戻り値

スクリプトバージョン。

参照

- SyncParms.setVersion メソッド [Ultra Light J]257 ページ

isDownloadOnly メソッド

同期がダウンロード専用かどうかを確認します。

構文

```
abstract boolean SyncParms.isDownloadOnly()
```

戻り値

アップロードが無効になっている場合は true、それ以外の場合は false。

参照

- SyncParms.setDownloadOnly メソッド [Ultra Light J]251 ページ

isPingOnly メソッド

クライアントが Mobile Link サーバーに対して ping または同期を実行しているかどうかを確認します。

構文

```
abstract boolean SyncParms.isPingOnly()
```

戻り値

クライアントがサーバーに対して ping だけを実行している場合は true、それ以外の場合は false。

参照

- SyncParms.setPingOnly メソッド [Ultra Light J]253 ページ

isUploadOnly メソッド

同期がアップロード専用かどうかを確認します。

構文

```
abstract boolean SyncParms.isUploadOnly()
```

戻り値

ダウンロードが無効になっている場合は true、それ以外の場合は false。

参照

- SyncParms.setUploadOnly メソッド [Ultra Light J]256 ページ

setAcknowledgeDownload メソッド

クライアントがダウンロードの確認を送信すべきかどうかを指定します。

構文

```
abstract void SyncParms.setAcknowledgeDownload(boolean ack)
```

パラメーター

- **ack**　クライアントからダウンロード確認を送信する場合は true、それ以外の場合は false に設定します。

備考

デフォルトは false です。

参照

- SyncParms.getAcknowledgeDownload メソッド [Ultra Light J]244 ページ

setAuthenticationParms メソッド

カスタムユーザー認証スクリプト (Mobile Link authenticate_parameters 接続イベント) のパラメーターを指定します。

構文

```
abstract void SyncParms.setAuthenticationParms(
    String v
) throws ULjException
```

パラメーター

- **v** 認証パラメーターのカンマ区切りのリスト、または NULL 参照。カンマ区切りのリストの詳細については、クラスの説明を参照してください。

備考

最初の 255 文字列のみが使用されます。それぞれの文字列は認証パラメーターの Mobile Link サーバーの制限よりも長くはなりません (現在は 4000 UTF8 バイト)。

21K 文字よりも長い文字列は、Mobile Link に送信されたときにトランケートされ、認証パラメーター用のサーバー制限を超過する文字列はサーバー側の同期エラーを引き起こします。

参照

- SyncParms.getAuthenticationParms メソッド [Ultra Light J]245 ページ

setDownloadOnly メソッド

同期をダウンロード専用に設定します。

構文

```
abstract void SyncParms.setDownloadOnly(boolean v)
```

パラメーター

- **v** 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。

備考

デフォルトは false です。true を指定すると、setPingOnly メソッドと setUploadOnly メソッドが自動的に呼び出され、これらが false に設定されます。

参照

- SyncParms.isDownloadOnly メソッド [Ultra Light J]249 ページ
- SyncParms.setPingOnly メソッド [Ultra Light J]253 ページ
- SyncParms.setUploadOnly メソッド [Ultra Light J]256 ページ

setLivenessTimeout メソッド

活性タイムアウトの長さを秒単位で設定します。

構文

```
abstract void SyncParms.setLivenessTimeout(
    int seconds
) throws ULjException
```

パラメーター

- **seconds** 新しい活性タイムアウト値。

備考

活性タイムアウトは、サーバーで許容される、リモートのアイドル時間の長さです。リモートが l 秒間サーバーと通信しなかった場合、サーバーはリモートとの接続が失われたとみなし、同期を終了します。リモートは、接続を継続するために、自動的に定期メッセージをサーバーに送信します。

負の値を設定すると、例外がスローされます。値は Mobile Link サーバーによって予告なく変更される場合があります。変更は、値が低すぎるか高すぎる場合に行われます。

デフォルトの値は、BlackBerry/J2SE/J2ME プラットフォームでは 100 秒、Android プラットフォームでは 240 秒です。

参照

- SyncParms.getLivenessTimeout メソッド [Ultra Light J]245 ページ

setNewPassword メソッド

setUserName メソッドで指定されたユーザーの新しい Mobile Link パスワードを設定します。

構文

```
abstract void SyncParms.setNewPassword(String v)
```

パラメーター

- **v** Mobile Link ユーザー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。

デフォルトは NULL で、この場合、パスワードは置換されません。

参照

- SyncParms.getNewPassword メソッド [Ultra Light J]245 ページ
- SyncParms.setPassword メソッド [Ultra Light J]253 ページ
- SyncParms.setUserName メソッド [Ultra Light J]256 ページ

setPassword メソッド

setUserName メソッドで指定されたユーザーの Mobile Link パスワードを設定します。

構文

```
abstract void SyncParms.setPassword(String v) throws ULjException
```

パラメーター

- **v** Mobile Link ユーザーのパスワード。

備考

このユーザー名とパスワードは、データベースのユーザー ID とパスワードとは異なります。このメソッドは、Mobile Link サーバーに対してアプリケーションを認証するために使用されます。

デフォルトは空の文字列で、これはパスワードなしを示します。

参照

- SyncParms.getPassword メソッド [Ultra Light J]246 ページ
- SyncParms.setNewPassword メソッド [Ultra Light J]252 ページ
- SyncParms.setUserName メソッド [Ultra Light J]256 ページ

setPingOnly メソッド

クライアントが Mobile Link サーバーに対して、同期を実行しないで ping を実行するように設定します。

構文

```
abstract void SyncParms.setPingOnly(boolean v)
```

パラメーター

- **v** サーバーに ping だけを実行する場合は true、同期を実行する場合は false に設定します。

備考

デフォルトは false です。true を指定すると、setDownloadOnly メソッドと setUploadOnly メソッドが自動的に呼び出され、これらが false に設定されます。

参照

- SyncParms.isPingOnly メソッド [Ultra Light J]249 ページ
- SyncParms.setDownloadOnly メソッド [Ultra Light J]251 ページ
- SyncParms.setUploadOnly メソッド [Ultra Light J]256 ページ

setPublications メソッド

同期させるパブリケーションを設定します。

構文

```
abstract void SyncParms.setPublications(String pubs) throws ULjException
```

パラメーター

- **pubs** パブリケーション名のカンマ区切りのリスト。カンマ区切りのリストの詳細については、クラスの説明を参照してください。

備考

デフォルトでは、データベース内のすべてのテーブルの同期を指定する Connection.SYNC_ALL 定数が設定されます。すべてのパブリケーションを同期するには、このメソッドに Connection.SYNC_ALL_PUBS 定数を設定します。

参照

- SyncParms.getPublications メソッド [Ultra Light J]246 ページ
- Connection.SYNC_ALL 変数 [Ultra Light J]126 ページ
- Connection.SYNC_ALL_PUBS 変数 [Ultra Light J]126 ページ

setSendColumnNames メソッド

同期中にカラム名を Mobile Link サーバーに送信するかどうかを指定します。

構文

```
abstract void SyncParms.setSendColumnNames(boolean c)
```

パラメーター

- **c** カラム名を送信する場合は true に設定します。

備考

カラム名は、Mobile Link サーバー API を使用している場合にのみサーバーで使用されます。

デフォルト値は false です。

参照

- SyncParms.getSendColumnNames メソッド [Ultra Light J]246 ページ

setSyncObserver メソッド

同期の進行状況をモニターする SyncObserver オブジェクトを設定します。

構文

```
abstract void SyncParms.setSyncObserver(SyncObserver so)
```

パラメーター

- **so** SyncObserver オブジェクト。

備考

デフォルトは NULL で、これは observer なしを示します。

参照

- SyncObserver インターフェイス [Ultra Light J]235 ページ

setTableOrder メソッド

統合データベースにテーブルがアップロードされる順序を設定します。

構文

```
abstract void SyncParms.setTableOrder(String v) throws ULjException
```

パラメーター

- **v** 同期する順序でのテーブル名のカンマ区切りのリスト、またはテーブル順序を指定しない場合は NULL。カンマ区切りのリストの詳細については、クラスの説明を参照してください。

備考

プライマリテーブルをリストの先頭に指定し、統合データベースで外部キー関係を持つすべてのテーブルをリストに含めます。

Publications パラメーターによって同期対象として選択されているテーブルはすべて、TableOrder パラメーターで指定されているかどうかに関係なく同期されます。指定されていないテーブルは、クライアントデータベースでの外部キー関係の順序で同期されます。これらは、指定したテーブルの後に同期されます。

デフォルト値は NULL 参照で、テーブルのデフォルトの順序は上書きされません。

参照

- SyncParms.getTableOrder メソッド [Ultra Light J]248 ページ
- SyncParms.setPublications メソッド [Ultra Light J]254 ページ

setUploadOnly メソッド

同期をアップロード専用に設定します。

構文

```
abstract void SyncParms.setUploadOnly(boolean v)
```

パラメーター

- **v** 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。

備考

デフォルトは false です。true を指定すると、setDownloadOnly メソッドと setPingOnly メソッドが自動的に呼び出され、これらが false に設定されます。

参照

- SyncParms.isUploadOnly メソッド [Ultra Light J]250 ページ
- SyncParms.setDownloadOnly メソッド [Ultra Light J]251 ページ
- SyncParms.setPingOnly メソッド [Ultra Light J]253 ページ

setUserName メソッド

Mobile Link サーバーがクライアントをユニークに識別する Mobile Link ユーザー名を設定します。

構文

```
abstract void SyncParms.setUserName(String v) throws ULjException
```

パラメーター

- **v** Mobile Link ユーザー名。

備考

この値は、以下を判別するために使用されます。

●ダウンロードの内容

●同期ステータスを記録するかどうか

●同期中に中断された場合、回復するかどうか

このユーザー名とパスワードは、データベースのユーザー ID とパスワードとは異なります。このメソッドは、Mobile Link サーバーに対してアプリケーションを認証するために使用されます。

このパラメーターは、SyncParms オブジェクトの作成時に初期化されます。

参照

- SyncParms.getUserName メソッド [Ultra Light J]248 ページ
- SyncParms.setPassword メソッド [Ultra Light J]253 ページ
- SyncParms.setNewPassword メソッド [Ultra Light J]252 ページ
- Connection.createSyncParms メソッド [Ultra Light J]111 ページ

setVersion メソッド

使用する同期スクリプトを設定します。

構文

```
abstract void SyncParms.setVersion(String v) throws ULjException
```

パラメーター

- **v** スクリプトバージョン。

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区別されます。たとえば、異なるバージョン文字列によって特定される 2 つの download_cursor スクリプトが存在する場合があります。バージョン文字列によって、アプリケーションが同期スクリプトのセットから適切に選択できます。

このパラメーターは、SyncParms オブジェクトの作成時に初期化されます。

参照

- SyncParms.getVersion メソッド [Ultra Light J]249 ページ
- Connection.createSyncParms メソッド [Ultra Light J]111 ページ

HTTP_STREAM 変数

HTTP 同期用の SyncParms オブジェクトを作成します。

構文

```
final int SyncParms.HTTP_STREAM
```

参照

- Connection.createSyncParms メソッド [Ultra Light J]111 ページ

HTTPS_STREAM 変数

セキュア HTTPS 同期用の SyncParms オブジェクトを作成します。

構文

```
final int SyncParms.HTTPS_STREAM
```

参照

- Connection.createSyncParms メソッド [Ultra Light J]111 ページ

SyncResult クラス

指定されたデータベース同期のステータス関連の情報をレポートします。

構文

```
public class SyncResult
```

メンバー

継承されたメンバーを含む SyncResult クラスのすべてのメンバー。

名前	説明
getAuthStatus メソッド	前回試行された同期の認証ステータスコードを返します。
getAuthValue メソッド	カスタムユーザー認証同期スクリプトで指定されている値を返します。
getCurrentTableName メソッド	現在同期中のテーブルの名前を返します。
getIgnoredRows メソッド	前回行われた同期で、アップロードされたローが無視されたかどうかを確認します。
getReceivedByteCount メソッド	データ同期中に受信したバイト数を返します。
getReceivedRowCount メソッド	受信したローの数を返します。
getSentByteCount メソッド	データ同期中に送信されたバイト数を返します。
getSentRowCount メソッド	送信されたローの数を返します。
getStreamErrorCode メソッド	ストリーム自体によってレポートされるエラーコードを返します。
getStreamErrorMessage メソッド	ストリーム自体によってレポートされるエラーメッセージを返します。

名前	説明
getSyncedTableCount メソッド	現在までに同期されたテーブル数を返します。
getTotalTableCount メソッド	同期されるテーブル数を返します。
isUploadOK メソッド	前回のアップロード同期が成功したかどうかを確認します。

参照

- SyncParms.getSyncResult メソッド [Ultra Light J]247 ページ

getAuthStatus メソッド

前回試行された同期の認証ステータスコードを返します。

構文

```
abstract int SyncResult.getAuthStatus()
```

戻り値

AuthStatusCode の値。

getAuthValue メソッド

カスタムユーザー認証同期スクリプトで指定されている値を返します。

構文

```
abstract int SyncResult.getAuthValue()
```

戻り値

カスタムユーザー認証同期スクリプトから返された整数。

getCurrentTableName メソッド

現在同期中のテーブルの名前を返します。

構文

```
abstract String SyncResult.getCurrentTableName()
```

戻り値

テーブル名。

getIgnoredRows メソッド

前回行われた同期で、アップロードされたローが無視されたかどうかを確認します。

構文

```
abstract boolean SyncResult.getIgnoredRows()
```

戻り値

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、ローが無視されなかった場合は false。

getReceivedByteCount メソッド

データ同期中に受信したバイト数を返します。

構文

```
abstract long SyncResult.getReceivedByteCount()
```

戻り値

バイト数。

getReceivedRowCount メソッド

受信したローの数を返します。

構文

```
abstract int SyncResult.getReceivedRowCount()
```

戻り値

受け取ったローの数です。

getSentByteCount メソッド

データ同期中に送信されたバイト数を返します。

構文

```
abstract long SyncResult.getSentByteCount()
```


戻り値

送信されたバイト数。

getSentRowCount メソッド

送信されたローの数を返します。

構文

```
abstract int SyncResult.getSentRowCount()
```

戻り値

送信されたローの数です。

getStreamErrorCode メソッド

ストリーム自体によってレポートされるエラーコードを返します。

構文

```
abstract int SyncResult.getStreamErrorCode()
```

戻り値

通信ストリームエラーがなかった場合は 0、それ以外の場合はサーバーからの応答コードを返します。

備考

このメソッドは、HTTP 応答コードを返します。

getStreamErrorMessage メソッド

ストリーム自体によってレポートされるエラーメッセージを返します。

構文

```
abstract String SyncResult.getStreamErrorMessage()
```

戻り値

メッセージがない場合は null、それ以外の場合は応答メッセージを返します。

備考

このメソッドは、HTTP 応答メッセージを返します。

getSyncedTableCount メソッド

現在までに同期されたテーブル数を返します。

構文

```
abstract int SyncResult.getSyncedTableCount()
```

戻り値

同期されたテーブル数。

getTotalTableCount メソッド

同期されるテーブル数を返します。

構文

```
abstract int SyncResult.getTotalTableCount()
```

戻り値

同期するテーブル数。

isUploadOK メソッド

前回のアップロード同期が成功したかどうかを確認します。

構文

```
abstract boolean SyncResult.isUploadOK()
```

戻り値

前回のアップロード同期が成功した場合は true、それ以外の場合は false。

SyncResult.AuthStatusCode インターフェイス

Mobile Link サーバーから返された認証コードを列挙します。

構文

```
public interface SyncResult.AuthStatusCode
```

メンバー

継承されたメンバーを含む SyncResult.AuthStatusCode インターフェイスのすべてのメンバー。

名前	説明
EXPIRED 変数	ユーザー ID またはパスワードの有効期限が切れています。
IN_USE 変数	ユーザー ID がすでに使用されています。
INVALID 変数	ユーザー ID またはパスワードが不正です。
UNKNOWN 変数	認証ステータスが不明です。
VALID 変数	ユーザー ID とパスワードは、同期時には有効でした。
VALID_BUT_EXPIRES_SOON 変数	ユーザー 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。

参照

- SyncResult.getAuthStatus メソッド [Ultra Light J]259 ページ

EXPIRED 変数

ユーザー ID またはパスワードの有効期限が切れています。

構文

```
final int SyncResult.AuthStatusCode.EXPIRED
```

備考

認証に失敗しました。

IN_USE 変数

ユーザー ID がすでに使用されています。

構文

```
final int SyncResult.AuthStatusCode.IN_USE
```

備考

認証に失敗しました。

INVALID 変数

ユーザー ID またはパスワードが不正です。

構文

```
final int SyncResult.AuthStatusCode.INVALID
```

備考

認証に失敗しました。

UNKNOWN 変数

認証ステータスが不明です。

構文

```
final int SyncResult.AuthStatusCode.UNKNOWN
```

備考

このコードは、同期が実行されていないことを示します。

VALID 変数

ユーザー ID とパスワードは、同期時には有効でした。

構文

```
final int SyncResult.AuthStatusCode.VALID
```

VALID_BUT_EXPIRES_SOON 変数

ユーザー 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。

構文

```
final int SyncResult.AuthStatusCode.VALID_BUT_EXPIRES_SOON
```

TableSchema インターフェイス

テーブルのスキーマを指定し、システムテーブルの名前を定義する定数を提供します。

構文

```
public interface TableSchema
```


メンバー

継承されたメンバーを含む TableSchema インターフェイスのすべてのメンバー。

名前	説明
SYS_ARTICLES 変数	パブリケーションのアーティクルに関する情報を含むシステムテーブルの名前です。
SYS_COLUMNS 変数	データベース内のテーブルカラムに関する情報を含むシステムテーブルの名前です。
SYS_FKEY_COLUMNS 変数	外部キーカラムに関する情報を含むシステムテーブルの名前です。
SYS_FOREIGN_KEYS 変数	データベース内の外部キーに関する情報を含むシステムテーブルの名前です。
SYS_INDEX_COLUMNS 変数	データベース内のインデックスカラムに関する情報を含むシステムテーブルの名前です。
SYS_INDEXES 変数	データベース内のテーブルインデックスに関する情報を含むシステムテーブルの名前です。
SYS_INTERNAL 変数	内部情報を含むシステムテーブルの名前です。
SYS_PRIMARY_INDEX 変数	システムテーブルのプライマリキーインデックスの名前です。
SYS_PUBLICATIONS 変数	データベースパブリケーションに関する情報を含むシステムテーブルの名前です。
SYS_TABLES 変数	データベース内のテーブルに関する情報を含むシステムテーブルの名前です。
SYS_ULDATA 変数	システムの値に関する情報を含むシステムテーブルの名前です。
SYS_ULDATA_INTERNAL 変数	内部システムデータの型です。
SYS_ULDATA_OPTION 変数	オプションのシステムデータの型です。
SYS_ULDATA_PROPERTY 変数	プロパティシステムデータの型です。
TABLE_IS_DOWNLOAD_ONLY 変数	テーブルがダウンロード専用テーブル (同期されたときにアップロードされるテーブル) であることを示します。

名前	説明
TABLE_IS_NOSYNC 変数	テーブルが非同期テーブルであることを示します。
TABLE_IS_SYSTEM 変数	テーブルがシステムテーブルであることを示します。

備考

バージョン 12 の時点で、このインターフェイスにはテーブル関連の定数のみが含まれるようになりました。これには、システムテーブル名、テーブルフラグ、**sysuldata** システムテーブルのデータ型などがあります。

SYS_ARTICLES 変数

パブリケーションのアーティクルに関する情報を含むシステムテーブルの名前です。

構文

```
final String TableSchema.SYS_ARTICLES
```

SYS_COLUMNS 変数

データベース内のテーブルカラムに関する情報を含むシステムテーブルの名前です。

構文

```
final String TableSchema.SYS_COLUMNS
```

SYS_FKEY_COLUMNS 変数

外部キーカラムに関する情報を含むシステムテーブルの名前です。

構文

```
final String TableSchema.SYS_FKEY_COLUMNS
```

SYS_FOREIGN_KEYS 変数

データベース内の外部キーに関する情報を含むシステムテーブルの名前です。

構文

```
final String TableSchema.SYS_FOREIGN_KEYS
```


SYS_INDEX_COLUMNS 変数

データベース内のインデックスカラムに関する情報を含むシステムテーブルの名前です。

構文

```
final String TableSchema.SYS_INDEX_COLUMNS
```

SYS_INDEXES 変数

データベース内のテーブルインデックスに関する情報を含むシステムテーブルの名前です。

構文

```
final String TableSchema.SYS_INDEXES
```

SYS_INTERNAL 変数

内部情報を含むシステムテーブルの名前です。

構文

```
final String TableSchema.SYS_INTERNAL
```

SYS_PRIMARY_INDEX 変数

システムテーブルのプライマリキーインデックスの名前です。

構文

```
final String TableSchema.SYS_PRIMARY_INDEX
```

SYS_PUBLICATIONS 変数

データベースパブリケーションに関する情報を含むシステムテーブルの名前です。

構文

```
final String TableSchema.SYS_PUBLICATIONS
```

SYS_TABLES 変数

データベース内のテーブルに関する情報を含むシステムテーブルの名前です。

構文

```
final String TableSchema.SYS_TABLES
```

SYS_ULDATA 変数

システムの値に関する情報を含むシステムテーブルの名前です。

構文

```
final String TableSchema.SYS_ULDATA
```

SYS_ULDATA_INTERNAL 変数

内部システムデータの型です。

構文

```
final String TableSchema.SYS_ULDATA_INTERNAL
```

SYS_ULDATA_OPTION 変数

オプションのシステムデータの型です。

構文

```
final String TableSchema.SYS_ULDATA_OPTION
```

SYS_ULDATA_PROPERTY 変数

プロパティシステムデータの型です。

構文

```
final String TableSchema.SYS_ULDATA_PROPERTY
```

TABLE_IS_DOWNLOAD_ONLY 変数

テーブルがダウンロード専用テーブル (同期されたときにアップロードされるテーブル) であることを示します。

構文

```
final short TableSchema.TABLE_IS_DOWNLOAD_ONLY
```


備考

この値は、TABLE_IS_NOSYNC フラグを除き、SYS_TABLES テーブルの table_flags カラムで他のフラグと論理的に組み合わせることができます。

TABLE_IS_NOSYNC 変数

テーブルが非同期テーブルであることを示します。

構文

```
final short TableSchema.TABLE_IS_NOSYNC
```

備考

この値は、TABLE_IS_DOWNLOAD_ONLY フラグを除き、SYS_TABLES テーブルの table_flags カラムで他のフラグと論理的に組み合わせることができます。

TABLE_IS_SYSTEM 変数

テーブルがシステムテーブルであることを示します。

構文

```
final short TableSchema.TABLE_IS_SYSTEM
```

備考

この値は、SYS_TABLES テーブルの table_flags カラムで他のフラグと論理的に組み合わせることができます。

ULjException クラス

データベースからスローされた例外に取って代わります。

構文

```
public class ULjException
```

メンバー

継承されたメンバーを含む ULjException クラスのすべてのメンバー。

名前	説明
getCausingException メソッド	例外の原因となっている ULjException オブジェクトを返します。

名前	説明
getErrorCode メソッド	この例外に関連付けられているエラーコードを返します。
getSqlOffset メソッド	SQL 文字列内のエラーオフセットを返します。

getCausingException メソッド

例外の原因となっている ULjException オブジェクトを返します。

構文

```
abstract ULjException ULjException.getCausingException()
```

戻り値

原因となっている例外がない場合は NULL、それ以外の場合は ULjException オブジェクトを返します。

getErrorCode メソッド

この例外に関連付けられているエラーコードを返します。

構文

```
abstract int ULjException.getErrorCode()
```

戻り値

エラーコード。

getSqlOffset メソッド

SQL 文字列内のエラーオフセットを返します。

構文

```
abstract int ULjException.getSqlOffset()
```

戻り値

エラーメッセージに関連付けられている SQL 文字列がない場合は -1、それ以外の場合は、文字列内でエラーが発生した箇所の 0 から始まるオフセットを返します。

Unsigned64 クラス

符号なし 64 ビットのバイナリ値を実装します。

構文

```
public class Unsigned64
```

メンバー

継承されたメンバーを含む Unsigned64 クラスのすべてのメンバー。

名前	説明
add メソッド	2 つの値を加算し、結果をそれ自体に配置します。
compare メソッド	2 つの long 値を比較します。
divide メソッド	2 つの値を除算し、結果をそれ自体に配置します。
multiply メソッド	2 つの値を乗算し、結果をそれ自体に配置します。
remainder メソッド	1 つの値を別の値で除算したときの余りを返します。
subtract メソッド	2 つの値を減算し、結果をそれ自体に配置します。

備考

このクラスの目的は、値を long integer として保持し、これらの値をこのクラスの静的メソッドを使用して解釈することです。

このクラスはインスタンスを作成できません。

add メソッド

2 つの値を加算し、結果をそれ自体に配置します。

構文

```
final long Unsigned64.add(long v1, long v2)
```

パラメーター

- **v1** 最初のオペランド。

- **v2** 2 番目のオペランド。

戻り値

2 つのオペランドの和。

compare メソッド

2 つの long 値を比較します。

オーバーロードリスト

名前	説明
compare(int, int) メソッド	2 つの integer 値を比較します。
compare(long, long) メソッド	2 つの long 値を比較します。

compare(int, int) メソッド

2 つの integer 値を比較します。

構文

```
final byte Unsigned64.compare(int v1, int v2)
```

パラメーター

- **v1** 比較する最初の値。
- **v2** 比較する 2 番目の値。

戻り値

v2 が v1 より大きい場合は -1、v1 と v2 が等しい場合は 0、v2 が v1 より小さい場合は 1。

compare(long, long) メソッド

2 つの long 値を比較します。

構文

```
final byte Unsigned64.compare(long v1, long v2)
```

パラメーター

- **v1** 比較する最初の値。
- **v2** 比較する 2 番目の値。

戻り値

v2 が v1 より大きい場合は -1、v1 と v2 が等しい場合は 0、v2 が v1 より小さい場合は 1。

divide メソッド

2 つの値を除算し、結果をそれ自体に配置します。

構文

```
final long Unsigned64.divide(long v1, long v2)
```

パラメーター

- **v1** 最初のオペランド。
- **v2** 2 番目のオペランド。

戻り値

最初のオペランドを 2 番目のオペランドで割った値。

multiply メソッド

2 つの値を乗算し、結果をそれ自体に配置します。

構文

```
final long Unsigned64.multiply(long v1, long v2)
```

パラメーター

- **v1** 最初のオペランド。
- **v2** 2 番目のオペランド。

戻り値

v1 と v2 の積。

remainder メソッド

1 つの値を別の値で除算したときの余りを返します。

オーバーロードリスト

名前	説明
remainder(long, long) メソッド	1 つの値を別の値で除算したときの余りを返します。
remainder(long, long, long) メソッド	指定の指数で乗算した値を別の値から差し引いた余り (v1 - quot * v2) を返します。

remainder(long, long) メソッド

1 つの値を別の値で除算したときの余りを返します。

構文

```
final long Unsigned64.remainder(long v1, long v2)
```

パラメーター

- **v1** 除算する値。
- **v2** 除算される値。

戻り値

long integer として表される余り。

remainder(long, long, long) メソッド

指定の指数で乗算した値を別の値から差し引いた余り (v1 - quot * v2) を返します。

構文

```
final long Unsigned64.remainder(long v1, long v2, long quot)
```

パラメーター

- **v1** 除算する値。
- **v2** 除算される値。
- **quot** 商の値。

戻り値

long integer として表される余り。

subtract メソッド

2 つの値を減算し、結果をそれ自体に配置します。

構文

```
final long Unsigned64.subtract(long v1, long v2)
```

パラメーター

- **v1** 最初のオペランド。
- **v2** 2 番目のオペランド。

戻り値

v1 から v2 を差し引いた値。

UUIDValue インターフェイス

ユニーク識別子 (UUID またはユニバーサルユニーク識別子) オブジェクトを記述します。

構文

```
public interface UUIDValue
```

メンバー

継承されたメンバーを含む UUIDValue インターフェイスのすべてのメンバー。

名前	説明
getString メソッド	UUIDValue オブジェクトの String 表現を返します。
isNull メソッド	UUIDValue オブジェクトが NULL かどうかを確認します。
set メソッド	UUIDValue オブジェクトに String 値を設定します。
setNull メソッド	UUIDValue オブジェクトを NULL に設定します。

備考

このようなエンティティは、ユニーク識別子が必要であり任意の値を指定できる場合に便利です。

テーブルのカラムに値が指定されておらず、DEFAULT NEWID() 句を使用してカラムが作成された場合、UUIDValue は SQL INSERT 文でも作成できます。

createUUIDValue メソッドを使用して UUIDValue を作成する場合、Connection オブジェクトを使用できます。

参照

- Connection.createUUIDValue メソッド [Ultra Light J]112 ページ

getString メソッド

UUIDValue オブジェクトの String 表現を返します。

構文

```
String UUIDValue.getString() throws ULjException
```

戻り値

String 値。

isNull メソッド

UUIDValue オブジェクトが NULL かどうかを確認します。

構文

```
boolean UUIDValue.isNull()
```

戻り値

オブジェクトが NULL の場合は true、NULL 以外の場合は false。

set メソッド

UUIDValue オブジェクトに String 値を設定します。

構文

```
void UUIDValue.set(String value) throws ULjException
```

パラメーター

- **value** String として表した数値。

setNull メソッド

UUIDValue オブジェクトを NULL に設定します。

構文

```
void UUIDValue.setNull() throws ULjException
```


索引

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Z

あ

