



UltraLite® – Java-Programmierung

Version 16.0

Februar 2013

Version 16.0
Februar 2013

© 2013 SAP AG oder ein SAP-Konzernunternehmen. Alle Rechte vorbehalten.

Sie können diese Dokumentation (ganz oder teilweise) unter folgenden Bedingungen benutzen, reproduzieren und verteilen: 1) Sie müssen diese und alle anderen Urheberrechtsvermerke auf allen Kopien oder Auszügen der Dokumentation wiedergeben. 2) Sie dürfen die Dokumentation nicht verändern. 3) Sie dürfen nichts tun, aus dem abgeleitet werden könnte, dass Sie oder jemand anderer als SAP Verfasser oder Quelle der Dokumentation ist. Die hier enthaltenen Informationen können jederzeit ohne vorherigen Hinweis geändert werden.

Einige Softwareprodukte, die von der SAP AG oder einem ihrer Vertriebspartner vermarktet werden, enthalten Softwarekomponenten anderer Softwareanbieter. Die nationalen Produktspezifikationen können unterschiedlich sein.

Diese Dokumentationen werden von der SAP AG und ihren Tochtergesellschaften ("SAP Group") lediglich zu Informationszwecken bereitgestellt, ohne dass eine Gewährleistung oder eine Garantie irgendeiner Art gegeben wird. Die SAP Group übernimmt keine Verantwortung im Hinblick auf Fehler oder Auslassungen in den Dokumentationen. Die einzigen Garantien für Produkte und Dienstleistungen der SAP Group sind diejenigen, die in den mit den Produkten und Dienstleistungen eventuell gelieferten ausdrücklichen Garantieerklärungen enthalten sind. Keine der hier enthaltenen Informationen kann als Gewährung einer weitergehenden Garantie betrachtet werden.

SAP und weitere erwähnte SAP-Produkte und -Dienstleistungen sowie die entsprechenden Logos sind Marken oder eingetragene Marken der SAP AG in Deutschland und anderen Ländern. Weitere Hinweise finden Sie unter <http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark>.

Inhalt

Über diese Dokumentation	vii
Erforderliche Systemausstattung und unterstützte Plattformen	1
UltraLiteJ-Anwendungsentwicklung	3
Kurzanleitung zur UltraLiteJ-Anwendungsentwicklung	3
Hinweise zum Android- und BlackBerry-Setup	4
UltraLite- und UltraLite Java Edition-Datenbank, Methoden zum Erstellen und Verbinden	5
Kurzeinführung in Schemavorgänge und Datenverwaltung	9
Zugriff auf Schemainformationen	19
Fehlerbehandlung	20
MobiLink-Datensynchronisation	21
So schließen Sie eine UltraLite Java Edition-Datenbankverbindung	26
So erstellen Sie UltraLiteJ-Anwendungen und stellen sie per Deployment bereit	27
Codebeispiele	30
Praktische Einführung: Eine Android-Anwendung erstellen	41
Lektion 1: Einrichten eines neuen Android-Projekts	42
Lektion 2: Starten des MobiLink-Servers	43
Lektion 3: Ausführen Ihrer Android-Anwendung	44
Lektion 4: Testen Ihrer Android-Anwendung und Synchronisieren	45
Aufräumen	46
Praktische Einführung: Eine BlackBerry-Anwendung erstellen	49
Teil 1: Erstellen einer neuen BlackBerry-Anwendung	49
Teil 2: MobiLink zum Synchronisieren der BlackBerry-Anwendung verwenden	65
Aufräumen	73
Programmcode der praktischen Einführung	74

UltraLiteJ-API-Referenz	81
ColumnSchema-Schnittstelle	81
ConfigFile-Schnittstelle	87
ConfigFileAndroid-Schnittstelle [Android]	89
ConfigNonPersistent-Schnittstelle [BlackBerry]	90
ConfigObjectStore-Schnittstelle [BlackBerry]	91
ConfigPersistent-Schnittstelle	93
Configuration-Schnittstelle	102
Connection-Schnittstelle	105
DatabaseInfo-Schnittstelle	133
DatabaseManager-Klasse	138
DecimalNumber-Schnittstelle	146
Domain-Schnittstelle	149
FileTransfer-Schnittstelle	159
FileTransferProgressData-Schnittstelle	175
FileTransferProgressListener-Schnittstelle	176
IndexSchema-Schnittstelle	178
PreparedStatement-Schnittstelle	180
ResultSet-Schnittstelle	197
ResultSetMetadata-Schnittstelle	216
SISListener-Schnittstelle [BlackBerry]	221
SISRequestHandler-Schnittstelle [BlackBerry]	222
SQLInfo-Schnittstelle [Android]	223
StreamHTTPParms-Schnittstelle	225
StreamHTTPSParms-Schnittstelle	238
SyncObserver-Schnittstelle	244
SyncObserver.States-Schnittstelle	246
SyncParms-Klasse	250
SyncResult-Klasse	268
SyncResult.AuthStatusCode-Schnittstelle	278
TableSchema-Schnittstelle	280
ULjEvent-Schnittstelle [Android]	285
ULjException-Klasse	286
Unsigned64-Klasse	288
UUIDValue-Schnittstelle	292

ValidateDatabaseProgressData-Schnittstelle [Android]	294
ValidateDatabaseProgressData.StatusId-Schnittstelle [Android]	295
ValidateDatabaseProgressListener-Schnittstelle [Android]	300
 Index	 303

Über diese Dokumentation

In diesem Handbuch wird die UltraLiteJ-Programmierschnittstelle beschrieben. Mit UltraLiteJ können Sie Datenbank Anwendungen entwickeln und das Deployment auf Android- und BlackBerry-Smartphones durchführen.

Erforderliche Systemausstattung und unterstützte Plattformen

Entwicklungsplattformen

Zum Entwickeln von UltraLiteJ-Anwendungen benötigen Sie Folgendes:

- Eine Java-IDE, z.B. Eclipse
- Java SE 1.6 oder höher

Zielpattformen

UltraLiteJ unterstützt die folgenden Zielpattformen:

- **Android-Smartphones**
- **BlackBerry-Smartphones**, die OS 4.2 und später ausführen
- Java SE 1.6 oder höher auf einem Computer oder Gerät

Weitere Hinweise zu den von UltraLite unterstützten Plattformen finden Sie unter <http://www.sybase.com/detail?id=1002288>.

UltraLiteJ-Anwendungsentwicklung

Die UltraLiteJ-API stellt Datenbankfunktionen und Synchronisation für Ihre Java-Anwendungen bereit. Sie wurde speziell für die Verwendung mit Android- und BlackBerry-Smartphones entwickelt, ist aber auch mit Java SE-Plattformen kompatibel. Die API enthält alle erforderlichen Methoden, um eine Verbindung mit einer UltraLite Java Edition-Datenbank oder einer UltraLite-Datenbank für Android herzustellen, Schemavorgänge auszuführen und Daten mithilfe von SQL-Anweisungen zu verwalten. Erweiterte Vorgänge, wie die Datenverschlüsselung und -synchronisation, werden ebenfalls unterstützt.

Hinweis

Die UltraLiteJ-API für Android-Smartphones verfügt über eine gemeinsame C++-Codebasis mit UltraLite für andere Plattformen und ihr Verhalten ähnelt dem anderer Plattformen. Es gibt einige Funktionen für den Zugriff auf Tabellen und Zeilen ohne SQL-Anweisungen, für die keine API auf Android bereitgestellt wird.

Siehe auch

- „UltraLite-Überblick“ [[UltraLite - Datenbankverwaltung](#)]
- „Vorteile der UltraLite APIs für Windows Mobile“ [[UltraLite - Datenbankverwaltung](#)]

Kurzanleitung zur UltraLiteJ-Anwendungsentwicklung

Wenn Sie eine UltraLiteJ-Anwendung erstellen, führen Sie üblicherweise die folgenden Datenverwaltungsaufgaben in Ihrem Anwendungscode aus:

1. Importieren des UltraLiteJ-API-Pakets in Ihre Java-Datei(en).

Name und Speicherort des UltraLiteJ-Pakets hängen davon ab, für welches Gerät Sie Anwendungen entwickeln.

2. Erstellen eines neuen Configuration-Objekts, um eine Datenbank zu erstellen oder sich mit dieser zu verbinden.

Configuration-Objekte definieren, wo sich die Client-Datenbank befindet oder wo sie erstellt werden soll. Außerdem geben sie den Benutzernamen und das Kennwort an, die beim Verbinden mit der Datenbank verlangt werden. Variationen von Configuration-Objekten stehen für verschiedene Geräte und für nicht-beständige Datenbankspeicher zur Verfügung.

3. Erstellen eines neuen Connection-Objekts.

Connection-Objekte stellen eine Verbindung zu einer Client-Datenbank her, wobei die im Configuration-Objekt festgelegten Spezifikationen verwendet werden.

4. Das Datenbankschema mithilfe von SQL-Anweisungen erstellen und ändern und die PreparedStatement-Schnittstelle zum Abfragen der Datenbank verwenden.

Sie können SQL-Anweisungen verwenden, Tabellen, Indizes, Fremdschlüssel und Publikationen für Ihre Datenbank zu erstellen oder zu aktualisieren.

PreparedStatement-Objekte fragen die Datenbank ab, die dem Connection-Objekt zugeordnet ist. Sie akzeptieren unterstützte SQL-Anweisungen, die als Zeichenfolgen übergeben werden. Sie können PreparedStatement-Objekte dazu verwenden, den Inhalt der Datenbank zu aktualisieren.

5. Generieren von ResultSet-Objekten.

ResultSet-Objekte werden erstellt, wenn das Connection-Objekt eine PreparedStatement-Anweisung ausführt, das eine SQL SELECT-Anweisung enthält. Sie können ResultSet-Objekte verwenden, um Zeilen von Abfrageergebnissen abzurufen und darin den Tabelleninhalt der Datenbank anzuzeigen.

Siehe auch

- „Hinweise zum Android- und BlackBerry-Setup“ auf Seite 4
- Configuration-Schnittstelle [UltraLiteJ] auf Seite 102
- Connection-Schnittstelle [UltraLiteJ] auf Seite 105
- „UltraLite-SQL-Anweisungen“ [UltraLite - Datenbankverwaltung]
- PreparedStatement-Schnittstelle [UltraLiteJ] auf Seite 180
- ResultSet-Schnittstelle [UltraLiteJ] auf Seite 197

Hinweise zum Android- und BlackBerry-Setup

Im Folgenden finden Sie einige Hinweise zur UltraLiteJ-API, die Sie vor der Entwicklung von Anwendungen für Android- und BlackBerry-Smartphones berücksichtigen müssen.

JAR-Ressourcendateien

Achten Sie beim Einrichten einer Anwendung für die UltraLiteJ-API darauf, dass Ihr Projekt korrekt für die Verwendung der geeigneten Datei *UltraLiteJ16.jar* bzw. *UltraLiteJNI16.jar* konfiguriert ist.

Die UltraLiteJ-API für Android befindet sich im Verzeichnis *UltraLite\UltraLiteJ\Android\UltraLiteJNI16.jar* der SQL Anywhere-Installation. Sie müssen Ihr Android-Entwicklungsprojekt so konfigurieren, dass die Datei *UltraLiteJNI16.jar* in den Classpath einbezogen wird. Weitere Hinweise finden Sie unter „[Praktische Einführung: Eine Android-Anwendung erstellen](#)“ auf Seite 41.

Verwenden Sie bei der Android-Entwicklung die folgende Anweisung, um das UltraLiteJ-Paket in Ihre Java-Datei zu importieren:

```
import com.ianywhere.ultralitejni16.*;
```

Die UltraLiteJ-API für BlackBerry und Java SE befindet sich im Verzeichnis *UltraLite\UltraLiteJ* Ihrer SQL Anywhere-Installation. Für jede Zielpattform gibt es ein entsprechendes Unterverzeichnis und eine Datei *UltraLiteJ16.jar*. Sie müssen Ihr BlackBerry-Entwicklungsprojekt so konfigurieren, dass die Datei *UltraLiteJ16.jar* in den Classpath einbezogen wird. Weitere Hinweise finden Sie unter „[Praktische Einführung: Eine BlackBerry-Anwendung erstellen](#)“ auf Seite 49.

Für die BlackBerry-Entwicklung verwenden Sie die folgende Anweisung, um das UltraLiteJ-Paket in Ihre Java-Datei zu importieren:

```
import com.ianywhere.ultralitej16.*;
```

Bei allen Kodierungsbeispielen und praktischen Einführungen in dieser Dokumentation wird vorausgesetzt, dass die obige Anweisung ausgeführt wurde und dass Sie mit der Entwicklung von Java-Anwendungen in Eclipse vertraut sind.

Java SE-Anwendungen

Java SE-Anwendungen werden von der Datei *UltraLiteJNI.jar* nicht unterstützt. Sie müssen die Datei *UltraLite\UltraLiteJ\J2SE\UltraLiteJ16.jar* verwenden.

UltraLite- und UltraLite Java Edition-Datenbanken

Auf Android-Smartphones bietet UltraLiteJ eine Schnittstelle zum gleichen UltraLite-Datenbank-Managementsystem, das für Windows Mobile, iPhone und Windows bereitgestellt wird. Auf BlackBerry-Smartphones stellt UltraLiteJ eine Schnittstelle zum Datenbank-Managementsystem von UltraLite Java Edition bereit. UltraLite Java Edition besitzt zwar ähnliche Funktionen wie UltraLite, aber nicht identische. UltraLite Java Edition-Datenbanken und UltraLite-Datenbanken sind nicht austauschbar.

Android-Smartphones unterstützen nur UltraLite-Datenbanken. Diese können in Sybase Central oder mithilfe von UltraLite-Befehlszeilen-Dienstprogrammen erstellt werden. Weitere Hinweise zum Erstellen einer UltraLite-Datenbank finden Sie unter „[UltraLite-Datenbank erstellen](#)“ [[UltraLite - Datenbankverwaltung](#)].

BlackBerry-Smartphones unterstützen nur UltraLite Java Edition-Datenbanken. Weitere Hinweise zu Erstellung und Verwendung einer UltraLite Java Edition-Datenbank finden Sie unter „[UltraLite- und UltraLite Java Edition-Datenbank, Methoden zum Erstellen und Verbinden](#)“ auf Seite 5.

Siehe auch

- „Vorteile der UltraLite APIs für Windows Mobile“ [[UltraLite - Datenbankverwaltung](#)]
- „Funktionsvergleiche von UltraLite, UltraLite Java Edition und SQL Anywhere“ [[UltraLite - Datenbankverwaltung](#)]
- „Einschränkungen für UltraLite- und UltraLite Java Edition-Datenbanken“ [[UltraLite - Datenbankverwaltung](#)]

UltraLite- und UltraLite Java Edition-Datenbank, Methoden zum Erstellen und Verbinden

Java-Anwendungen müssen eine Verbindung mit einer Datenbank herstellen, bevor Datenoperationen durchgeführt werden können. In diesem Abschnitt wird erklärt, wie Sie mithilfe der UltraLiteJ-API eine UltraLite- oder UltraLite Java Edition-Datenbank erstellen bzw. mithilfe eines angegebenen Kennworts eine Verbindung mit der Datenbank herstellen.

Hinweis

Sie können entweder Sybase Central oder UltraLite-Befehlszeilen-Dienstprogramme verwenden, um eine UltraLite-Datenbank ohne die UltraLiteJ-API zu erstellen. Siehe „[UltraLite-Datenbank erstellen](#)“ [[UltraLite - Datenbankverwaltung](#)].

Um eine UltraLite Java Edition-Datenbank ohne die UltraLiteJ-API zu erstellen, können Sie eine der folgenden Aufgaben ausführen:

- Erstellen Sie die Datenbank mit dem Dienstprogramm uljload. Siehe „[Lade-Dienstprogramm für UltraLite Java Edition-Datenbank \(uljload\)](#)“ [[UltraLite - Datenbankverwaltung](#)].
- Verwenden Sie die Dienstprogramme ulunload und uljload, um eine UltraLite-Datenbank zu konvertieren. Siehe „[UltraLite-Dienstprogramm zum Entladen von Datenbanken \(ulunload\)](#)“ [[UltraLite - Datenbankverwaltung](#)].
- Stellen Sie eine Java SE-Anwendung auf einem BlackBerry-Smartphone bereit, indem Sie die UltraLite Java-Datenbank auf eine SD-Karte kopieren oder mithilfe des Dateiübertragungsverfahrens via MobiLink übertragen. Siehe „[MobiLink-Dateiübertragungen](#)“ [[UltraLite - Datenbankverwaltung](#)].

Weitere Hinweise zu den Unterschieden zwischen einer UltraLite-Datenbank und einer UltraLite Java Edition-Datenbank finden Sie unter [UltraLite- und UltraLite Java Edition-Datenbanken auf Seite 5](#).

Ein Configuration-Objekt wird verwendet, um einen Datenbankspeicher zu konfigurieren. Es stehen mehrere Implementierungen eines Configuration-Objekts zur Verfügung. Es gibt eine spezielle Implementierung für jeden Datenbankspeicher-Typ, der von der UltraLiteJ-API unterstützt wird. Jede Implementierung stellt einen Satz von Methoden bereit, die zum Konfigurieren des Datenbankspeichers verwendet werden.

Die folgende Tabelle enthält die verfügbaren Configuration-Objekt-Implementierungen für die unterstützten Datenbankspeicher:

Speichertyp	UltraLiteJ-API-Unterstützung
Android-Dateisystem	Siehe ConfigFileAndroid-Schnittstelle [Android] [UltraLiteJ] .
RIM-Objekt (BlackBerry)	Siehe ConfigObjectStore-Schnittstelle [BlackBerry] [UltraLiteJ] .
Java SE-Dateisystem	Siehe ConfigFile-Schnittstelle [UltraLiteJ] .
Nicht-beständig (speicherresident)	Siehe ConfigNonPersistent-Schnittstelle [BlackBerry] [UltraLiteJ] .

Nach dem Erstellen und Konfigurieren eines Configuration-Objekts verwenden Sie ein Connection-Objekt, um die Datenbank zu erstellen oder eine Verbindung mit ihr herzustellen. Connection-Objekte können auch verwendet werden, um die folgenden Vorgänge auszuführen:

- **Transaktionen** Eine Transaktion ist ein Satz von Vorgängen zwischen Festschreibungen oder Rollbacks. Bei beständigen Datenbankspeichern werden durch das Festschreiben alle Änderungen seit

dem letzten Festschreiben oder Rollback permanent gespeichert. Ein Rollback setzt die Datenbank in den Zustand zurück, der beim Aufruf des letzten Festschreibens aktuell war.

Jede Transaktion und jeder Vorgang auf Zeilenebene in UltraLiteJ ist atomar (nicht teilbar). Beim Einfügen von Daten in mehrere Spalten werden Daten entweder in alle betreffenden Spalten oder in keine der betreffenden Spalten eingefügt.

Transaktionen müssen mit der commit-Methode des Connection-Objekts in der Datenbank festgeschrieben werden.

- **Vorbereitete SQL-Anweisungen** Die Methoden werden durch die PreparedStatement-Schnittstelle für die Verarbeitung von SQL-Anweisungen bereitgestellt. Ein PreparedStatement-Objekt kann mit der preparedStatement-Methode des Connection-Objekts erstellt werden.
- **Synchronisationen** Eine Menge von Objekten, die die Synchronisation steuern und auf die vom Connection-Objekt aus zugegriffen wird.

Siehe auch

- [„Java SE-Beispiel: Erstellen einer Datenbank“ auf Seite 32](#)
- [Connection-Schnittstelle \[UltraLiteJ\] auf Seite 105](#)
- [DatabaseManager-Klasse \[UltraLiteJ\] auf Seite 138](#)

Datenbank erstellen oder damit verbinden

Verwenden Sie die UltraLiteJ-API mit Ihrer Java-Anwendung zum Erstellen einer Datenbank oder zum Verbinden mit einer Datenbank.

Voraussetzungen

Eine vorhandene Java-Anwendung für ein Android- oder ein BlackBerry-Smartphone, das die UltraLiteJ-API implementiert.

Aufgabe

1. Erstellen Sie ein neues Configuration-Objekt, das den Datenbanknamen referenziert und für Ihre Plattform geeignet ist.

In den folgenden Beispielen ist **config** der Name des Configuration-Objekts und **DBname** der Datenbankname.

Für Android-Smartphones:

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

Für BlackBerry-Smartphones:

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

Für Java SE-Plattformen:

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

Für jede Plattform können Sie ein Configuration-Objekt für eine nicht-beständige Datenbank erstellen:

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

2. Sie können Datenbankeigenschaften mithilfe von Methoden des Configuration-Objekts festlegen.

Beispielsweise können Sie mit der setPassword-Methode ein neues Datenbankkennwort festlegen:

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

Bei Android-Smartphones können Sie mithilfe der Methoden setCreationString und setConnectionString zusätzliche Erstellungs- bzw. Verbindungsparameter festlegen.

Weitere Hinweise zu Erstellungs- und Verbindungsparametern finden Sie unter:

- „UltraLite-Erstellungsparameter“ [[UltraLite - Datenbankverwaltung](#)]
- ConfigPersistent.setCreationString-Methode [Android] [UltraLiteJ] auf Seite 99
- „UltraLite-Verbindungsparameter“ [[UltraLite - Datenbankverwaltung](#)]
- ConfigPersistent.setConnectionString-Methode [Android] [UltraLiteJ] auf Seite 99

3. Erstellen Sie ein Connection-Objekt, um eine Datenbank zu erstellen oder sich mit dieser zu verbinden:

Der folgende Code erstellt beispielsweise eine neue Datenbank:

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

Die DatabaseManager.createDatabase-Methode erstellt die Datenbank und gibt eine Verbindung zu ihr zurück.

Im oben gezeigten Beispiel wird der folgende Code verwendet, um eine Verbindung zu einer vorhandenen Datenbank herzustellen:

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

Die connect-Methode schließt den Prozess der Datenbank-Verbindungsherstellung ab. Wenn die Datenbank nicht existiert, wird ein Fehler ausgegeben.

Ergebnisse

Sie können zwar SQL-Anweisungen über Ihre Java-Anwendung ausführen, um die Tabellen und Indizes in Ihrer Datenbank zu erstellen, aber nicht bestimmte Datenbankerstellungparameter ändern, z.B. den Namen der Datenbank, das Kennwort oder die Seitengröße.

Siehe auch

- [DatabaseManager-Klasse \[UltraLiteJ\] auf Seite 138](#)
- [ConfigFileAndroid-Schnittstelle \[Android\] \[UltraLiteJ\]](#)
- [ConfigObjectStore-Schnittstelle \[BlackBerry\] \[UltraLiteJ\]](#)
- [ConfigFile-Schnittstelle \[UltraLiteJ\]](#)
- [ConfigNonPersistent-Schnittstelle \[BlackBerry\] \[UltraLiteJ\]](#)

Kurzeinführung in Schemavorgänge und Datenverwaltung

Erstellen, Aktualisieren oder Abrufen von Tabellen, Indizes, Fremdschlüsseln, Publikationen und Zeilen in der Datenbank mithilfe von SQL-Anweisungen und Abfragen.

Hinweis

Einige SQL-Anweisungen werden für BlackBerry-Smartphones nicht unterstützt.

Wenn Sie Schemavorgänge zur Datenverwaltung ausführen, führen Sie in der Regel die folgenden Aufgaben in Ihrem Anwendungscode durch:

1. Schemavorgänge durchführen.

Verwalten und Ändern des Schemas mit SQL-Anweisungen wie z.B. CREATE TABLE oder CREATE INDEX auf der Datenbankverbindung.

2. Zeilenvorgänge verwalten

Daten in Tabellen mit SQL-Anweisungen wie INSERT, UPDATE oder DELETE auf der Datenbankverbindung verwalten

3. Zeilendaten in einer Ergebnismenge abrufen

Eine Ergebnismenge mit der SELECT-Anweisung abrufen und die Zeilendaten mit Navigationsmethoden für die Ergebnismenge durchsuchen, wie z.B. **previous** und **next**.

Beispiel: Datenbankvorgänge auf einem Android-Smartphone verwalten

Das folgende Beispiel zeigt eine Klasse in einer Android-Anwendung, die die UltraLiteJ-API verwendet, um die folgenden Vorgänge auszuführen:

- Eine Tabelle in einer Datenbank erstellen
- Neue Zeilen in die Tabelle einfügen
- Eine Zeile in einer Tabelle aktualisieren

- Eine Zeile aus einer Tabelle löschen
- Änderungen in der Datenbank festschreiben
- Alle Zeilen in der Tabelle durch eine Ergebnismenge auswählen
- Ergebnismenge durchlaufen, um die Zeilen in der Datenbank anzuzeigen

Zusätzlich zu diesen Vorgängen enthält die Klasse eine Methode namens `PrintText`, die verwendet wird, um die erfolgreichen Vorgänge in das Log zu schreiben (siehe Registerkarte **LogCat** in Eclipse), sowie eine `HandleError`-Methode, die benutzt wird, um Fehler zu melden, die während der Ausführung von UltraLiteJ-API-Vorgängen auftreten.

```
package com.sampleapp;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import com.ianywhere.ultralitejni16.*;

public class NewUltraLiteJAppActivity extends Activity {
    Connection _conn = null;
    ResultSet _departments = null;
    PreparedStatement _inserter = null;
    PreparedStatement _updater = null;
    PreparedStatement _deleter = null;
    PreparedStatement _preparer = null;

    public void PrintText(String strText) {
        Log.i("NewUltraLiteJAppActivity", strText);
    }

    public void HandleError(ULjException err) {
        Log.w("NewUltraLiteJAppActivity", "Exception: " + err.toString());
    }

    public Connection GetDatabase(String strFilename) {
        ConfigFileAndroid config = null;
        Connection dbConnection = null;

        try {
            config = DatabaseManager.createConfigurationFileAndroid(
                strFilename, getApplicationContext()
            );
            dbConnection = DatabaseManager.connect(config);
            PrintText("Successfully connected to the database at: "
                + strFilename);
        } catch(ULjException ex) {
            if (config != null) {
                try {
                    dbConnection = DatabaseManager.createDatabase(config);
                    PrintText("Successfully created a new database at: "
                        + strFilename);
                } catch(ULjException exception) {
                    HandleError(exception);
                }
            }
            HandleError(ex);
        }
        return dbConnection;
    }
}
```

```

public void Commit() {
    try {
        _conn.commit();
    } catch (ULjException e1) {
        HandleError(e1);
    }
}

public void CloseDatabase() {
    try {
        _conn.release();
    } catch (ULjException e) {
        HandleError(e);
    }
}

public void ExecuteSQLStatement(String strSQLstmt) {
    PreparedStatement ps;
    try {
        ps = _conn.prepareStatement(strSQLstmt);
        ps.execute();
        ps.close();
        PrintText("Successfully executed: " + strSQLstmt);
    } catch (ULjException e) {
        HandleError(e);
    }
}

public void InitStatements() {
    String stmt;
    try {
        stmt = "INSERT INTO Department(dept_no, name) VALUES (?,?)";
        _inserter = _conn.prepareStatement(stmt);
        stmt = "UPDATE Department SET dept_no = ? WHERE dept_no = ?";
        _updater = _conn.prepareStatement(stmt);
        stmt = "DELETE FROM Department WHERE dept_no = ?";
        _deleter = _conn.prepareStatement(stmt);
    } catch (ULjException e) {
        HandleError(e);
    }
}

public void FiniStatements() {
    try {
        _departments.close();
        _inserter.close();
        _updater.close();
        _deleter.close();
        _preparer.close();
    } catch (ULjException e) {
        HandleError(e);
    }
}

public void AddDepartment(int deptID, String deptName) {
    try {
        _inserter.set(1, deptID);
        _inserter.set(2, deptName);
        _inserter.execute();
        PrintText("Successfully executed:"
            + " INSERT INTO Department(dept_no, name)"
            + " VALUES (" + deptID + "," + deptName + ")");
    } catch (ULjException e) {

```

```
        HandleError(e);
    }
}

public void UpdateDepartment(int deptIDold, int deptIDnew) {
    try {
        _updater.set(1, deptIDnew);
        _updater.set(2, deptIDold);
        _updater.execute();
        PrintText("Successfully executed:"
            + " UPDATE Department SET dept_no = " + deptIDnew
            + " WHERE dept_no = " + deptIDold);
    } catch (ULjException e) {
        HandleError(e);
    }
}

public void DeleteDepartment(int deptID) {
    try {
        _deleter.set(1, deptID);
        _deleter.execute();
        PrintText("Successfully executed:"
            + " DELETE FROM Department WHERE dept_no = " + deptID);
    } catch (ULjException e) {
        HandleError(e);
    }
}

public ResultSet SelectDepartmentRows() {
    String stmt = "SELECT * FROM Department ORDER BY dept_no";
    _preparer = null;
    _departments = null;
    try {
        _preparer = _conn.prepareStatement(stmt);
        _departments = _preparer.executeQuery();
        PrintText("Successfully executed: " + stmt);
    } catch (ULjException e) {
        HandleError(e);
    }
    return _departments;
}

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    PrintText("Starting application...");

    _conn = GetDatabase("test.udb");
    if (_conn == null) {
        return;
    }

    String[] stmt = new String[3];

    stmt[0] = "CREATE TABLE Department("
        + "dept_no INT PRIMARY KEY, "
        + "name CHAR(50) NOT NULL)";

    stmt[1] = "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(dept_no))";

stmt[2] = "CREATE INDEX ON Employee(last_name, first_name)";

for(int i = 0; i< stmt.length; i++) {
    ExecutesSQLStatement(stmt[i]);
}

InitStatements();

AddDepartment(101, "Electronics");
AddDepartment(105, "Sales");
AddDepartment(109, "Accounting");

UpdateDepartment(101, 102);

DeleteDepartment(102);

Commit();

_departments = SelectDepartmentRows();
if (_departments != null) {
    try {
        while(_departments.next()) {
            int dept_no = _departments.getInt(1);
            String dept_name = _departments.getString(2);
            PrintText("Department no.:" + dept_no
                    + " Department name: " + dept_name);
        }
    } catch (ULjException e) {
        HandleError(e);
    }
}

FiniStatements();

CloseDatabase();

PrintText("Closing application...");
finish();
    }
}

```

Siehe auch

- [DatabaseManager-Klasse \[UltraLiteJ\] auf Seite 138](#)
- [DatabaseManager.createConfigurationFileAndroid-Methode \[UltraLiteJ\] auf Seite 140](#)
- [Connection.prepareStatement-Methode \[UltraLiteJ\] auf Seite 118](#)
- [PreparedStatement-Schnittstelle \[UltraLiteJ\] auf Seite 180](#)
- [PreparedStatement.set-Methode \[UltraLiteJ\] auf Seite 189](#)
- [PreparedStatement.execute-Methode \[UltraLiteJ\] auf Seite 182](#)
- [PreparedStatement.executeQuery-Methode \[UltraLiteJ\] auf Seite 183](#)
- [Connection.commit-Methode \[UltraLiteJ\] auf Seite 110](#)
- [Connection.rollback-Methode \[UltraLiteJ\] auf Seite 120](#)
- [ResultSet-Schnittstelle \[UltraLiteJ\] auf Seite 197](#)
- [ResultSet.afterLast-Methode \[Android\] \[UltraLiteJ\] auf Seite 200](#)
- [ResultSet.beforeFirst-Methode \[Android\] \[UltraLiteJ\] auf Seite 200](#)
- [ResultSet.first-Methode \[Android\] \[UltraLiteJ\] auf Seite 200](#)
- [ResultSet.last-Methode \[Android\] \[UltraLiteJ\] auf Seite 215](#)
- [ResultSet.next-Methode \[UltraLiteJ\] auf Seite 215](#)
- [ResultSet.previous-Methode \[UltraLiteJ\] auf Seite 215](#)
- [ResultSet.relative-Methode \[Android\] \[UltraLiteJ\] auf Seite 215](#)
- [„UltraLite-SQL-Anweisungen“ \[UltraLite - Datenbankverwaltung\]](#)

Schemavorgänge

Führen Sie Schemavorgänge mit diesen allgemeinen Aufgaben durch:

1. Eine SQL-Anweisung in einer Zeichenfolge aufbauen
2. Ein PreparedStatement-Objekt durch Übergabe der Zeichenfolgenvariablen an die Connection.prepareStatement-Methode erstellen.
3. Die PreparedStatement.execute-Methode aufrufen, um den Vorgang in der Datenbank durchzuführen.
4. Das PreparedStatement-Objekt schließen, um Ressourcen freizugeben.

Beispiel

Der Code in diesem Beispiel ist Teil eines vollständigen Beispiels, das veranschaulicht, wie grundlegende Schema- und Datenverwaltungsvorgänge mit der UltraLiteJ-API durchgeführt werden. Siehe [„Beispiel: Datenbankvorgänge auf einem Android-Smartphone verwalten“](#).

Einzelne CREATE TABLE- und CREATE INDEX-Anweisungen werden erstellt und an eine benutzerdefinierte Methode namens ExecuteSQLStatement übergeben, um alle erforderlichen Schemavorgänge durchzuführen:

```
String[] stmt = new String[3];

stmt[0] = "CREATE TABLE Department("
    + "dept_no INT PRIMARY KEY, "
    + "name CHAR(50) NOT NULL)";

stmt[1] = "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(dept_no))";

stmt[2] = "CREATE INDEX ON Employee(last_name, first_name)";

for(int i = 0; i< stmt.length; i++) {
    ExecuteSQLStatement(stmt[i]);
}
```

Die ExecuteSQLStatement-Methode besteht aus dem folgenden Code:

```
public void ExecuteSQLStatement(String strSQLstmt) {
    PreparedStatement ps;
    try {
        ps = _conn.prepareStatement(strSQLstmt);
        ps.execute();
        ps.close();
        PrintText("Successfully executed: " + strSQLstmt);
    } catch (ULjException e) {
        HandleError(e);
    }
}
```

Siehe auch

- [Connection.prepareStatement-Methode \[UltraLiteJ\] auf Seite 118](#)
- [PreparedStatement-Schnittstelle \[UltraLiteJ\] auf Seite 180](#)
- [PreparedStatement.set-Methode \[UltraLiteJ\] auf Seite 189](#)
- [PreparedStatement.execute-Methode \[UltraLiteJ\] auf Seite 182](#)

Zeilenvorgänge verwalten

Sie verwalten Zeilenvorgänge, indem Sie folgende allgemeine Aufgaben durchführen:

1. Eine SQL-Anweisung in einer Zeichenfolge aufbauen
2. Ein PreparedStatement-Objekt durch Übergabe der Zeichenfolgenvariablen an die Connection.prepareStatement-Methode erstellen
3. Legen Sie die Hostvariablen fest, die durch das ?-Zeichen markiert werden, indem Sie die PreparedStatement.set-Methode verwenden.

Jede Hostvariable kann in Übereinstimmung mit ihrer Ordinalposition in der Anweisung referenziert werden. Das erste ? wird als 1 referenziert, das zweite als 2. Die **set**-Methode im untenstehenden Beispiel ermöglicht es Ihnen, die Ordinalposition der Variablen zu referenzieren und einen neuen Wert anzugeben.

4. Die PreparedStatement.execute-Methode aufrufen, um den Vorgang in der Datenbank durchzuführen.
5. Schreiben Sie die Änderungen in der Datenbank fest, indem Sie die Connection.commit-Methode aufrufen, um die Änderungen permanent zu speichern. Andernfalls rufen Sie die Connection.rollback-Methode auf.

Transaktionen müssen mithilfe der von der Connection-Schnittstelle unterstützten Methoden explizit festgeschrieben oder zurückgesetzt werden.

6. Das PreparedStatement-Objekt schließen, um Ressourcen freizugeben.

Beispiel

Der Code in diesem Beispiel ist Teil eines vollständigen Beispiels, das veranschaulicht, wie grundlegende Schema- und Datenverwaltungsvorgänge mit der UltraLiteJ-API durchgeführt werden. Siehe „[Beispiel: Datenbankvorgänge auf einem Android-Smartphone verwalten](#)“.

Globale PreparedStatement-Objekte werden mit einem InitStatements-Methodenaufruf definiert und instanziiert. Vorgänge zum Einfügen, Aktualisieren und Löschen von Daten werden in den Erläuterungen zu den AddDepartment-, UpdateDepartment- und DeleteDepartment-Methoden erklärt. Die Commit-Methode zeigt, wie Sie die Zeilenvorgänge permanent machen. Die FiniStatements-Methode schließt die globalen PreparedStatement-Objekte und gibt Ressourcen frei.

```
Connection _conn = null;
ResultSet _departments = null;
PreparedStatement _inserter = null;
PreparedStatement _updater = null;
PreparedStatement _deleter = null;
PreparedStatement _preparer = null;

public void InitStatements() {
    String stmt;
    try {
        stmt = "INSERT INTO Department(dept_no, name) VALUES (?,?)";
        _inserter = _conn.prepareStatement(stmt);
        stmt = "UPDATE Department SET dept_no = ? WHERE dept_no = ?";
        _updater = _conn.prepareStatement(stmt);
        stmt = "DELETE FROM Department WHERE dept_no = ?";
        _deleter = _conn.prepareStatement(stmt);
    } catch (ULjException e) {
        HandleError(e);
    }
}

public void FiniStatements() {
    try {
        _departments.close();
        _inserter.close();
        _updater.close();
        _deleter.close();
        _preparer.close();
    } catch (ULjException e) {
        HandleError(e);
    }
}

public void AddDepartment(int deptID, String deptName) {
    try {
        _inserter.set(1, deptID);
        _inserter.set(2, deptName);
        _inserter.execute();
        PrintText("Successfully executed:"
            + " INSERT INTO Department(dept_no, name)"
            + " VALUES (" + deptID + ", " + deptName + ")");
    } catch (ULjException e) {
        HandleError(e);
    }
}
```

```

    }
}

public void UpdateDepartment(int deptIDold, int deptIDnew) {
    try {
        _updater.set(1, deptIDnew);
        _updater.set(2, deptIDold);
        _updater.execute();
        PrintText("Successfully executed:"
            + " UPDATE Department SET dept_no = " + deptIDnew
            + " WHERE dept_no = " + deptIDold);
    } catch (ULjException e) {
        HandleError(e);
    }
}

public void DeleteDepartment(int deptID) {
    try {
        _deleter.set(1, deptID);
        _deleter.execute();
        PrintText("Successfully executed:"
            + " DELETE FROM Department WHERE dept_no = " + deptID);
    } catch (ULjException e) {
        HandleError(e);
    }
}

public void Commit() {
    try {
        _conn.commit();
    } catch (ULjException e1) {
        HandleError(e1);
    }
}

```

Hinweis

Zeichenfolgenverkettung sollte gegenüber der Verwendung von Hostvariablen vorgezogen werden, wenn SQL-Anweisungen erstellt werden, die nur einmal ausgeführt werden müssen.

Beispiel: Der folgende Code, der Zeichenfolgenverkettung zum Aufbau von SQL-Anweisungen verwendet, kann als Ersatz der DeleteDepartment-Methode benutzt werden:

```

public void DeleteDepartment(int deptID) {
    String stmt = "DELETE FROM Department WHERE dept_no = " + deptID;
    PreparedStatement deleter;
    try {
        deleter = _conn.prepareStatement(stmt);
        deleter.execute();
        deleter.close();
        PrintText("Successfully executed: " + stmt);
    } catch (ULjException e) {
        HandleError(e);
    }
}

```

Siehe auch

- [Connection.prepareStatement-Methode \[UltraLiteJ\] auf Seite 118](#)
- [PreparedStatement-Schnittstelle \[UltraLiteJ\] auf Seite 180](#)
- [PreparedStatement.set-Methode \[UltraLiteJ\] auf Seite 189](#)
- [PreparedStatement.execute-Methode \[UltraLiteJ\] auf Seite 182](#)

Zeilendaten abrufen

Sie rufen Zeilendaten aus einer Tabelle ab, indem Sie folgende allgemeine Aufgaben ausführen:

1. Eine SELECT SQL-Anweisung in einer Zeichenfolge erstellen
2. Ein PreparedStatement-Objekt durch Übergabe der Zeichenfolgenvariablen an die Connection.prepareStatement-Methode erstellen
3. Die PreparedStatement.executeQuery-Methode aufrufen, um die Abfrageergebnisse einem ResultSet-Objekt zuzuweisen
4. Das ResultSet-Objekt mithilfe der Navigationsmethoden zum Abrufen der Zeilendaten durchsuchen

Die folgenden Navigationsmethoden können verwendet werden, um durch eine Ergebnismenge zu navigieren:

- **afterLast**¹ Bewegt den Cursor hinter die letzte Zeile.
- **beforeFirst**¹ Bewegt den Cursor vor die erste Zeile.
- **first**¹ Bewegt den Cursor in die erste Zeile.
- **last**¹ Bewegt den Cursor in die letzte Zeile.
- **next** Bewegt den Cursor in die nächste Zeile.
- **previous** Bewegt den Cursor in die vorige Zeile.
- **relative(offset)**¹ Bewegt den Cursor um eine bestimmte Anzahl von Zeilen, der im Offsetwert angegeben ist, relativ zur aktuellen Zeile. Positive Offsetwerte bewegen den Cursor in der Ergebnismenge vorwärts, relativ zur aktuellen Position des Cursors in der Ergebnismenge. Negative Offsetwerte bewegen ihn rückwärts in der Ergebnismenge. Der Offsetwert Null ändert die Cursorposition nicht, ermöglicht aber das Einlesen von Daten in den Zeilenpuffer.

¹ Diese Methode wird von UltraLite Java Edition-Datenbanken nicht unterstützt.

5. Schließen Sie das ResultSet-Objekt und das PreparedStatement-Objekt, um Ressourcen freizugeben.

Beispiel

Der Code in diesem Beispiel ist Teil eines vollständigen Beispiels, das veranschaulicht, wie grundlegende Schema- und Datenverwaltungsvorgänge mit der UltraLiteJ-API durchgeführt werden. Siehe „[Beispiel: Datenbankvorgänge auf einem Android-Smartphone verwalten](#)“.

Mit der benutzerdefinierten `SelectDepartmentRows`-Methode wird eine Ergebnismenge abgerufen:

```
public ResultSet SelectDepartmentRows() {
    String stmt = "SELECT * FROM Department ORDER BY dept_no";
    _preparer = null;
    _departments = null;
    try {
        _preparer = _conn.prepareStatement(stmt);
        _departments = _preparer.executeQuery();
        PrintText("Successfully executed: " + stmt);
    } catch (ULjException e) {
        HandleError(e);
    }
    return _departments;
}
```

Zeilendaten werden abgerufen, indem eine Ergebnismenge mit der **next**-Navigationsmethode durchsucht wird:

```
_departments = SelectDepartmentRows();
if (_departments != null) {
    try {
        while(_departments.next()) {
            int dept_no = _departments.getInt(1);
            String dept_name = _departments.getString(2);
            PrintText("Department no.:" + dept_no
                + " Department name: " + dept_name);
        }
    } catch (ULjException e) {
        HandleError(e);
    }
}
```

Siehe auch

- [PreparedStatement.executeQuery-Methode \[UltraLiteJ\] auf Seite 183](#)
- [ResultSet-Schnittstelle \[UltraLiteJ\] auf Seite 197](#)
- [ResultSet.afterLast-Methode \[Android\] \[UltraLiteJ\] auf Seite 200](#)
- [ResultSet.beforeFirst-Methode \[Android\] \[UltraLiteJ\] auf Seite 200](#)
- [ResultSet.first-Methode \[Android\] \[UltraLiteJ\] auf Seite 200](#)
- [ResultSet.last-Methode \[Android\] \[UltraLiteJ\] auf Seite 215](#)
- [ResultSet.next-Methode \[UltraLiteJ\] auf Seite 215](#)
- [ResultSet.previous-Methode \[UltraLiteJ\] auf Seite 215](#)
- [ResultSet.relative-Methode \[Android\] \[UltraLiteJ\] auf Seite 215](#)

Zugriff auf Schemainformationen

Sie können Beschreibungen von Datenbankschemata programmatisch abrufen. Diese Beschreibungen werden als **Schemainformationen** bezeichnet und Sie können entweder über Systemtabellen oder über die Schema-Schnittstellen der UltraLiteJ-API darauf zugreifen.

Zugriff auf Schemainformationen über Systemtabellen

Schemainformationen für Datenbanken werden in UltraLite- bzw. UltraLite Java Edition-Systemtabellen gespeichert. Sie können auf diese Informationen zugreifen, indem Sie eine normale SQL-Abfrage

ausführen, um die gewünschten Informationen aus der entsprechenden Systemtabelle auszuwählen, und anschließend auf die Ergebnismenge zuzugreifen.

Zugriff auf Schemainformationen über Schema-Schnittstellen

Bei einigen Schemainformationen ist der Zugriff über Schema-Schnittstellen statt über Systemtabellen möglich. Die UltraLiteJ-API enthält die folgenden Schema-Schnittstellen:

- **TableSchema** Gibt Informationen über die Spalten- und Indexkonfiguration zurück.
- **IndexSchema** Gibt Informationen über die Spalten im Index zurück. IndexSchema-Objekte können aus TableSchema-Objekten abgeleitet werden.
- **ColumnSchema** Gibt Informationen über die Spalten in der Tabelle zurück. ColumnSchema-Objekte können aus TableSchema-Objekten abgeleitet werden.

Siehe auch

- „Zeilendaten abrufen“ auf Seite 18
- „UltraLite-Systemtabellen“ [*UltraLite - Datenbankverwaltung*]
- „Systemtabellen der UltraLite Java Edition“ [*UltraLite - Datenbankverwaltung*]
- TableSchema-Schnittstelle [UltraLiteJ] auf Seite 280
- IndexSchema-Schnittstelle [UltraLiteJ] auf Seite 178
- ColumnSchema-Schnittstelle [UltraLiteJ] auf Seite 81

Fehlerbehandlung

Sie können die Klassen `ULjException` und `SQLCode` verwenden, um Fehler zu behandeln. Die meisten UltraLite-Methoden geben `ULjException`-Fehler aus. Mit der `ULjException.getErrorCode`-Methode können Sie den `SQLCode`-Wert abrufen, der dem Fehler zugeordnet ist. Mit der `ULjException.toString`-Methode können Sie einen beschreibenden Text zum Fehler abrufen. `SQLCode`-Fehler sind negative Zahlen, die den Fehlertyp angeben und mithilfe von Konstanten, z.B. `ULjException.SQLE_INDEX_NOT_FOUND`, referenziert werden können.

Beispiel

Das folgende Beispiel zeigt eine Java-Klasse, in der die `ULjException`-Klasse für die Behandlung eines Fehlers verwendet wird, der beim Herstellen einer Verbindung mit einer UltraLite Java Edition-Datenbank auftreten kann:

```
import com.ianymwhere.ultralitej16.*;
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;
}

```

Siehe auch

- [ULjException-Klasse \[UltraLiteJ\] auf Seite 286](#)
- [ULjException.getErrorCode-Methode \[UltraLiteJ\] auf Seite 287](#)
- [„SQL Anywhere-Fehlermeldungen - sortiert nach SQLCODE“ \[Fehlermeldungen\]](#)

MobiLink-Datensynchronisation

Das Datenbank-Managementsystem von UltraLite Java Edition enthält ein integriertes Protokollierungssystem, mit dessen Hilfe Datenbankänderungen synchronisiert werden können.

Die Datensynchronisation kann mithilfe von HTTP- oder HTTPS-Netzwerkprotokollen durchgeführt werden. Die HTTPS Synchronisation stellt eine sichere Verschlüsselung für den MobiLink-Server zur Verfügung.

Um Daten zu synchronisieren, muss Ihre Anwendung folgende Schritte durchführen:

1. Instanzieren Sie ein syncParms-Objekt, das Informationen über die konsolidierte Datenbank enthält (Name des Servers, Portnummer), den Namen der zu synchronisierenden Datenbank sowie die Definition der zu synchronisierenden Tabellen.
2. Aufruf der Synchronisationsmethode vom Verbindungsobjekt, wobei das syncParms-Objekt die Synchronisation durchführen soll

Die zu synchronisierenden Daten können auf der Tabellenebene definiert werden. Die Synchronisation kann nicht für Teile einer Tabelle konfiguriert werden.

Siehe auch

- [SyncParms-Klasse \[UltraLiteJ\] auf Seite 250](#)
- [SyncResult-Klasse \[UltraLiteJ\] auf Seite 268](#)
- [„Praktische Einführung: Eine Android-Anwendung erstellen“ auf Seite 41](#)

Beispiel

Das folgende Beispiel zeigt eine Datensynchronisation mit einer UltraLiteJ-Anwendung und befindet sich unter %SQLANYSAMPI6%\UltraLiteJ\J2SE\Sync.java:

```
package com.ianywhere.ultralitej.demo;
import com.ianywhere.ultralitej16.*;
/**
 * 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 16.0" );
            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 );  
    }  
}
```

Um den MobiLink-Server mit CustDB als der konsolidierten Datenbank zu starten, führen Sie *start_ml.bat* vom %SQLANYSAMPI6%\J2SE\UltraLiteJ-Verzeichnis aus.

Für Android-Smartphones ist eine praktische Einführung verfügbar, die auf der Verwendung von CustDB als konsolidierte Datenbank basiert.

Datensynchronisation auf einem BlackBerry-Smartphone

In einer BlackBerry-Umgebung werden Daten zwischen dem Gerät und dem BlackBerry Enterprise Server (BES) immer verschlüsselt. HTTPS ist nützlich, wenn zwischen dem BES und dem MobiLink-Server Verschlüsselung erforderlich ist oder wenn das Gerät nicht von einem BES verwaltet wird.

Gleichzeitige Synchronisation

Normalerweise ist in der UltraLite-Laufzeitumgebung nur ein Thread gleichzeitig erlaubt. Bei der Synchronisation gibt es jedoch eine Ausnahme zu dieser Regel. Während eine Verbindung einen Synchronisationsvorgang durchführt, können andere Verbindungen auf die UltraLite-Laufzeitumgebung zugreifen, jedoch kann kein anderer Thread die *synchronize*-Methode für die zu synchronisierende Datenbank aufrufen, während ein *synchronize*-Vorgang durchgeführt wird.

Da UltraLite Isolationsstufe Null verwendet, kann eine Verbindung während der Synchronisation auf heruntergeladene Zeilen zugreifen (d.h., bevor diese festgeschrieben werden). Diese können später verloren gehen, falls die Synchronisation fehlschlägt. Wenn eine Verbindung während einer Synchronisation eine Zeile ändert, die die Synchronisation dann versucht zu ändern, schlägt die Synchronisation fehl. Wenn während einer Synchronisation eine Verbindung versucht, eine Zeile zu ändern, die bereits von der Synchronisation geändert wurde, schlägt der Änderungsversuch fehl.

Netzwerkprotokolloptionen für UltraLiteJ-Synchronisationsdatenströme

Bei der Synchronisation mit einem MobiLink-Server müssen Sie das Netzwerkprotokoll in Ihrer Anwendung festlegen. Jede Datenbank synchronisiert über ein Netzwerkprotokoll. Für UltraLiteJ sind zwei Netzwerkprotokolle verfügbar: HTTP und HTTPS.

Für das verwendete Netzwerkprotokoll können Sie verschiedene Protokolloptionen wählen, um sicherzustellen, dass die UltraLiteJ-Anwendung den MobiLink-Server lokalisieren und eine korrekte Verbindung mit ihm aufbauen kann. Netzwerkprotokolloptionen bieten Informationen wie beispielsweise Adressinformationen (Host und Port) und protokollspezifische Informationen.

Ein HTTP-Netzwerkprotokoll einrichten

Ein HTTP-Netzwerkprotokoll wird mit der *StreamHTTPParms*-Schnittstelle in der UltraLiteJ-API festgelegt. Verwenden Sie die Schnittstellenmethoden, um die Netzwerkprotokolloptionen anzugeben, die auf dem MobiLink-Server festgelegt sind.

Ein HTTPS-Netzwerkprotokoll einrichten

Ein HTTPS-Netzwerkprotokoll wird mit der StreamHTTPSParms-Schnittstelle in der UltraLiteJ-API festgelegt. Verwenden Sie die Schnittstellenmethoden, um die Netzwerkprotokolloptionen anzugeben, die auf dem MobiLink-Server festgelegt sind.

Siehe auch

- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]
- StreamHTTPSParms-Schnittstelle [UltraLiteJ] auf Seite 225
- StreamHTTPSParms-Schnittstelle [UltraLiteJ] auf Seite 238

CustDB-Anwendungssynchronisation auf einem BlackBerry-Smartphone

CustDB (Customer Database) ist eine mit SQL Anywhere installierte Beispieldatenbank und UltraLite-Anwendung. Die CustDB-Datenbank ist eine einfache Datenbank für Verkaufsaufträge.

Die Anwendung finden und ihr Deployment vornehmen

UltraLiteJ enthält eine BlackBerry-Beispielanwendung, die auf der Datenbank CustDB basiert. Die Anwendung heißt CustDB. Der Quellcode und zugehörige Dateien befinden sich im Verzeichnis `%SQLANYSAMPI6%\UltraLiteJ\BlackBerry\CustDB\`. Das CustDB-Verzeichnis enthält die Projektdateien, die mit der BlackBerry-JDE geöffnet werden können. Weitere Hinweise zur CustDB-Anwendung finden Sie in *readme.txt*.

Führen Sie nach dem Erstellen der CustDB-Anwendung ein Deployment von *CustDB16.cod* und den erforderlichen Dateien auf einen Simulator oder ein Gerät durch.

Zur CustDB-Anwendung gehörende Dateien

Die folgenden Dateien im CustDB-Projekt enthalten den Zugriffscode für die Datenbank:

- **CustDB.java** Diese Datei enthält alle grundlegenden Datenbank-Zugriffsmethoden. Diese Methoden umfassen das Erstellen von und die Verbindungsherstellung zu Datenbanken sowie das Einfügen, Löschen und Aktualisieren von Bestellungen. Diese Datei enthält viele Datenbankaufrufe zur Kommunikation mit dem Backend-Server.
- **SchemaCreator.java** Diese Datei enthält den Code zum Erstellen von Tabellen auf dem Gerät unter Verwendung von UltraLiteJ.

Die CustDB-Anwendung verwenden

Führen Sie `%SQLANYSAMPI6%\UltraLiteJ\BlackBerry\CustDB\mobilink.bat` aus, um einen lokalen MobiLink-Server zu starten.

Wenn es erstmals gestartet wird, sammelt das CustDB-Programm Informationen, die verwendet werden, um mit dem Server zu interagieren, auf dem die CustDB-Datenbank gehostet wird. Sie geben die für Abfragen zu verwendende Employee ID (50 wird empfohlen) an sowie den Hostnamen oder die IP-Adresse des Servers, auf dem die Daten gehostet werden, und eine Portnummer für die Verbindung zum Server.

Employee ID and Sync Info
Employee ID: 50
Host Name or IP Address: 209.183.139.45
Port Number: 80

Sobald diese Werte angegeben und die Einstellungen gespeichert wurden (mit **Menü » Save (Speichern)**), synchronisiert die Anwendung mit dem angegebenen Server. Die Anwendung lädt vom Server nur Bestellungen herunter, die mit der Employee ID übereinstimmen, die der angegebenen Mitarbeiternummer (50) entspricht. Nur Bestellungen, die noch offen sind, werden ausgewählt (Bestellungen können in drei Status sein: Open/offen, Approved/genehmigt oder Denied/abgelehnt).

Jede Bestellung wird auf dem Bildschirm mit dem Kundennamen, dem bestellten Produkt, der Menge, dem Preis und dem Rabatt angezeigt. Der Bildschirm zeigt auch den aktuellen Status der Bestellung sowie etwaige Anmerkungen an, die diese Bestellung betreffen.

UltraLiteJ CustDB Demo
<div>Next</div> <div>Previous</div>
Customer: Apple St. Builders Product: 4x8 Drywall x100 Quantity: 25000 Price: 400 Discount: 20
Status: Open Notes:

Auf diesem Bildschirm können Sie Anmerkungen zur Bestellung hinzufügen oder den Status der Bestellung ändern (auf "Approved" bzw. "Denied"). Sie können durch die Bestellungen unter Verwendung der Schaltflächen **Next (Weiter)** und **Previous (Zurück)** navigieren.

Das CustDB-Programm ermöglicht es Ihnen auch, neue Bestellungen der Datenbank hinzuzufügen. Um eine neue Bestellung hinzuzufügen, klicken Sie auf **Menü » New Order (Neue Bestellung)**.

UltraLiteJ CustDB Demo	
<input type="button" value="Next"/>	
<input type="button" value="Previous"/>	
Customer:	Apple St. Builders
Product:	4x8 Drywall x100
Quantity:	25000
Price:	400
Discount:	20
Status:	Open
Notes:	

Geben Sie Werte für Menge und Rabatt ein.

Bevor Sie die Anwendung beenden, klicken Sie im Hauptmenü auf **Synchronize (Synchronisieren)**, um Ihre Änderungen und neuen Bestellungen mit der konsolidierten Datenbank zu synchronisieren.

UltraLiteJ CustDB Demo	
<input type="button" value="Next"/>	
<input type="button" value="Previous"/>	
<input type="button" value="Print Tables"/>	e St. Builders
<input type="button" value="New Order"/>	Wall x100
<input type="button" value="Delete Order"/>	
<input type="button" value="Synchronize"/>	
<input type="button" value="About CustDB"/>	
<input type="button" value="Reset Database"/>	
<input type="button" value="Close"/>	

Siehe auch

- „So erstellen Sie UltraLiteJ-Anwendungen und stellen sie per Deployment bereit“ auf Seite 27

So schließen Sie eine UltraLite Java Edition-Datenbankverbindung

Eine UltraLite Java Edition-Datenbank wird geschlossen, wenn alle gleichzeitigen Verbindungen freigegeben wurden. Mit der `DatabaseManager.release`-Methode können Sie alle Verbindungen freigeben.

Hinweis

Verwenden Sie die `Connection.release`-Methode, um die aktuelle Verbindung freizugeben.

Falls eine BlackBerry-Anwendung abstürzt, bevor sie die aktiven Datenbankverbindungen freigeben kann, wird möglicherweise ein `ULjException`-Objekt mit dem `SQLE_FILE_IN_USE`-Fehlercode generiert, wenn die Anwendung versucht, eine Verbindung mit der Datenbank erneut herzustellen. In diesem Szenario müssen Sie das Smartphone neu starten, sodass die Anwendung die Verbindung wiederherstellen kann.

Um einen Neustart des Systems zu vermeiden, sollte eine catch-all-Ausnahmeroutine verwendet werden, um die `emergencyShutdown`-Methode aufzurufen, sodass die Verbindungen freigegeben werden.

Beispiel: Die folgende catch-all-Ausnahmeroutine kann in einer BlackBerry-Anwendung verwendet werden, um die `emergencyShutdown`-Methode aufzurufen, wenn ein nicht behebbarer Fehler auftritt:

```
try {
    // top level application code
    // release all connections in a normal termination
} catch( Exception e ) {
    conn.emergencyShutdown();
    throw e;
}
```

In diesem Beispiel wird vorausgesetzt, dass das `conn`-Objekt eine aktive Verbindung mit einer UltraLite Java Edition-Datenbank darstellt.

Siehe auch

- [DatabaseManager.release-Methode \[UltraLiteJ\] auf Seite 145](#)
- [Connection.release-Methode \[UltraLiteJ\] auf Seite 119](#)

So erstellen Sie UltraLiteJ-Anwendungen und stellen sie per Deployment bereit

UltraLiteJ-Anwendungen können auf Android- und BlackBerry-Smartphones bereitgestellt werden. Damit eine UltraLiteJ-Anwendung erfolgreich ausgeführt werden kann, müssen Sie die UltraLiteJ-API zusammen mit der Anwendung per Deployment bereitstellen.

Siehe auch

- „Kompilierungs- und Deploymentspezifikationen für UltraLite-Anwendungen“ [[UltraLite - Datenbankverwaltung](#)]

Deployment einer UltraLiteJ-Anwendung für Android

Geben Sie die entsprechenden Erstellungsparameter, Verbindungsparameter, Synchronisationsparameter, Protokolloptionen, Methodenaufrufe und Deployment-Dateien an, um sicherzustellen, dass die UltraLiteJ-Anwendung auf Android-Smartphones erfolgreich ausgeführt wird.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Fügen Sie Ihrem Android-Projekt die folgenden Dateien hinzu:

- `%SQLANYI6%\UltraLite\UltraLiteJ\Android\UltraLiteJNI16.jar`
- `%SQLANYI6%\UltraLite\UltraLiteJ\Android\ARM\libultralitej16.so`

2. Geben Sie folgende Parameter an:

- Wenn Sie Verschleierung verwenden, setzen Sie beim Erstellen der Datenbank den **obfuscate=1**-Erstellungsparameter.
- Wenn Sie AES-Verschlüsselung verwenden, setzen Sie beim Erstellen der Datenbank oder beim Herstellen einer Verbindung mit der Datenbank den Verbindungsparameter **DBKEY=encryption-key**.

Um diese Parameter zu setzen, verwenden Sie die `setCreationString`- und `setConnectionString`-Methoden.

3. Setzen Sie die entsprechenden Parameter, wenn Sie die Synchronisation in der UltraLite-Anwendung verwenden:

Synchronisationstyp	Parametereinstellungen
TCP/IP	Setzen Sie den Synchronisationsparameter Stream auf "tcpip".
HTTP	Setzen Sie den Synchronisationsparameter Stream auf "http".
RSA_TLS	Setzen Sie den Synchronisationsparameter Stream auf "tls".
RSA HTTPS	Setzen Sie den Synchronisationsparameter Stream auf "https".

4. Bei der Verwendung von RSA-Ende-zu-Ende-Verschlüsselung (RSA E2EE) setzen Sie die Protokolloption **e2ee_public_key=key-file**.
5. Wenn Sie ZLIB-Kompression verwenden, setzen Sie die Protokolloption **compression=zlib**.
6. Bei der Verwendung von AES-Verschlüsselung rufen Sie die `ConfigPersistent.EnableAesDBEncryption`-Methode auf.
7. Bei der Verwendung von RSA TLS, RSA HTTPS oder RSA E2EE stellen Sie `%SQLANYI6%\UltraLite\UltraLiteJ\Android\ARM\libmlcrsa16.so` per Deployment bereit.

Ergebnisse

Die UltraLiteJ-Anwendung wird erfolgreich auf dem Android-Gerät ausgeführt, auf dem sie bereitgestellt wurde.

Nächste Schritte

Führen Sie das Deployment einer UltraLite-Datenbank auf das mobile Android-Gerät durch oder erstellen Sie eine neue Datenbank mit der per Deployment bereitgestellten Anwendung.

Siehe auch

- „Kompilierungs- und Deploymentspezifikationen für UltraLite-Anwendungen“ [[UltraLite - Datenbankverwaltung](#)]
- „Datenbank-Deploymenttechniken für UltraLite und UltraLite Java Edition“ [[UltraLite - Datenbankverwaltung](#)]
- `ConfigPersistent.setCreationString`-Methode [Android] [UltraLiteJ] auf Seite 99
- `ConfigPersistent.setConnectionString`-Methode [Android] [UltraLiteJ] auf Seite 99

Deployment einer UltraLiteJ-Anwendung für BlackBerry

Verwenden Sie Methodenaufrufe, um geeignete Erstellungsparameter, Verbindungsparameter, Synchronisationsparameter und Protokolloptionen anzugeben und ein Deployment der entsprechenden Dateien durchzuführen, um sicherzustellen, dass die UltraLiteJ-Anwendung auf BlackBerry-Smartphones erfolgreich ausgeführt wird.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Rufen Sie während der Erstellung der Datenbank und der Verbindung folgende Methoden auf:
 - `ConfigPersistent.enableObfuscation`, wenn Sie Verschleierung verwenden.
 - `ConfigPersistent.EnableAesDBEncryption` und `ConfigPersistent.setEncryptionKey`, wenn Sie AES-Verschlüsselung verwenden.
2. Wenn Sie die Synchronisation verwenden, geben Sie den Synchronisationsdatenstrom an, indem Sie die `SyncParms.HTTP_STREAM`-Konstante oder die `SyncParms.HTTPS_STREAM`-Konstanten an die `Connection.createSyncParms`-Methode übergeben, und verwenden Sie anschließend die `StreamHTTTPParms`- oder die `StreamHTTPSParms`-Schnittstelle.
3. Setzen Sie die Komprimierungs- und Verschlüsselungsoptionen über die Schnittstelle, die Sie im vorherigen Schritt gewählt haben:
 - Für die ZLIB-Komprimierung rufen Sie die `setZlibCompression`-Methode auf.
 - Für die RSA E2EE-Verschlüsselung rufen Sie die `setE2eePublicKey`-Methode auf.
4. Führen Sie ein Deployment der folgenden Dateien durch, die sich im `%SQLANY16%\UltraLite\UltraLiteJ\BlackBerry4.2`-Verzeichnis befinden:
 - `UltraLiteJ16.cod`

- *UltraLiteJ16.jad*

UltraLiteJ16.jad ist nur für OTA-Deployment (Over-The-Air) erforderlich. Als Alternative können Sie Ihre eigene *JAD*-Datei erstellen, die das Deployment von UltraLiteJ mit Ihrer Anwendung vornimmt.

5. Bei der Verwendung von RSA E2EE-Verschlüsselung übertragen Sie die DER-kodierte Datei über die FileTransfer-Schnittstelle oder speichern die Datei auf eine SD-Karte.

Ergebnisse

Die UltraLiteJ-Anwendung wird erfolgreich auf dem Blackberry-Gerät ausgeführt, auf dem sie bereitgestellt wurde.

Nächste Schritte

Führen Sie das Deployment einer UltraLite Java-Datenbank auf das Blackberry Smartphone durch, auf dem die Anwendung bereitgestellt wurde, oder erstellen Sie eine neue Datenbank mit der per Deployment bereitgestellten Anwendung.

Siehe auch

- „Kompilierungs- und Deploymentspezifikationen für UltraLite-Anwendungen“ [[UltraLite - Datenbankverwaltung](#)]
- „Datenbank-Deploymenttechniken für UltraLite und UltraLite Java Edition“ [[UltraLite - Datenbankverwaltung](#)]
- ConfigPersistent-Schnittstelle [UltraLiteJ] auf Seite 93
- StreamHTTPParams-Schnittstelle [UltraLiteJ] auf Seite 225
- StreamHTTPSPParams-Schnittstelle [UltraLiteJ] auf Seite 238
- FileTransfer-Schnittstelle [UltraLiteJ] auf Seite 159

Codebeispiele

Dieser Abschnitt enthält Java-Codebeispiele, die die UltraLiteJ-API verwenden. In den Beispielen wird eine Demoklasse verwendet, die Meldungen anzeigt und **ULjException**-Objekte für Fehlersuchzwecke behandelt.

Alle Codebeispiele befinden sich im Verzeichnis `%SQLANYSAMPI6%\UltraLiteJ\J2SE`. Es wird empfohlen, dass Sie Sicherungskopien vom ursprünglichen Quellcode erstellen, bevor Sie Änderungen am Inhalt der Datei vornehmen.

Obwohl diese Codebeispiele für die Windows-PC-Umgebung geschrieben wurden, gelten die dargestellten Konzepte für alle UltraLiteJ-Plattformen (sofern nicht anders angegeben).

Um diese Beispiele auszuführen, setzen Sie die Umgebungsvariable **JAVA_HOME** auf eine installierte Version des JDK 1.6. Anschließend können Sie *rundemo.cmd* mit dem Namen des Beispiels ausführen.

Die Demoklasse in `%SQLANYSAMPI6%\UltraLiteJ\J2SE\Demo.java` wird von allen Beispielen in diesem Abschnitt der Dokumentation verwendet.

Zum Beispiel führt der folgende Befehl das CreateDb-Beispiel aus:

```
rundemo CreateDb
```

Der Befehl bewirkt die folgende Ausgabe:

```
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
```

Die Ausgabe wird in einer Datei namens *demos.out* gespeichert.

BlackBerry-Beispiele

Die folgenden Demos für BlackBerry-Entwickler stehen im Verzeichnis *%SQLANYSAMPI6%\UltraLiteJ\BlackBerry* zur Verfügung:

- Die BinaryStoreAsFile-Demo, die die Verwendung der Möglichkeit zum Zuordnen von externen Dateien als Teil einer Datenbank veranschaulicht.
- Die CustDB-Demo, die eine mobile Anwendung für Kundenbestellungen zeigt.
- Das BlackBerryEncryption-Demopakett, das hoch entwickelte Konzepte für ultra-sichere UltraLiteJ-Datenbanken präsentiert.
- Die HelloBlackBerry-Demo.

Eine praktische Einführung auf Basis der HelloBlackBerry-Demo steht zur Verfügung.

Android-Beispiel

Ein Eclipse-Beispielprojekt, das die konsolidierte Beispieldatenbank CustDB verwendet, wird im Verzeichnis *%SQLANYSAMPI6%\UltraLiteJ\Android\CustDB* zur Verfügung gestellt. Der Quellcode befindet sich im Verzeichnis *%SQLANYSAMPI6%\UltraLiteJ\Android\CustDB\src\com\sybase\custdb*.

Eine praktische Einführung auf Basis des Beispiels steht zur Verfügung.

Siehe auch

- [„Praktische Einführung: Eine BlackBerry-Anwendung erstellen“ auf Seite 49](#)
- [„Praktische Einführung: Eine Android-Anwendung erstellen“ auf Seite 41](#)

Java SE-Beispiel: Erstellen einer Datenbank

Führen Sie das Java SE-Beispiel durch, um zu verstehen, wie Sie einen Dateisystem-Datenbankspeicher in einer Java SE-Java-Umgebung erstellen.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Kontext und Bemerkungen

Das Configuration-Objekt wird verwendet, um die Datenbank zu erstellen. Sobald sie erstellt wurde, wird das Connection-Objekt zurückgegeben. Beim Erstellen von Tabellen wird die `schemaUpdateBegin`-Methode aufgerufen, um mit den Änderungen am zugrunde liegenden Schema zu beginnen, und die `schemaUpdateComplete`-Methode schließt das Ändern des Schemas ab.

Aufgabe

1. Wechseln Sie zum folgenden Verzeichnis: `%SQLANYSAMPI6%\UltraLiteJ\J2SE`.
2. Führen Sie das CreateDb-Beispiel aus:

```
rundemo CreateDb
```

Ergebnisse

Die Anwendung wird erfolgreich ausgeführt. Ein Datenbankspeicher für das UltraLite Java Edition-Dateisystem wird erstellt.

Java SE-Beispiel: Einfügen von Zeilen

Führen Sie das Java SE-Beispiel durch, um das Einfügen von Zeilen in eine UltraLite Java Edition-Datenbank zu verstehen.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Wechseln Sie zum folgenden Verzeichnis: `%SQLANYSAMPI6%\UltraLiteJ\J2SE`.
2. Führen Sie das CreateDb-Beispiel aus:

```
rundemo CreateDb
```

3. Führen Sie den folgenden Befehl aus (mit Berücksichtigung der Groß- und Kleinschreibung):

```
rundemo LoadDb
```

Ergebnisse

Die Anwendung wird erfolgreich ausgeführt. Die Zeilen werden in die Datenbank eingefügt.

Eingefügte Daten werden in der Datenbank erst beständig, wenn vom Connection-Objekt aus die commit-Methode aufgerufen wird.

Wenn eine Zeile eingefügt, aber noch nicht festgeschrieben wurde, ist sie für andere Verbindungen sichtbar. Das kann dazu führen, dass eine Verbindung Zeilendaten abrufen, die noch nicht wirklich festgeschrieben sind.

Siehe auch

- [„Java SE-Beispiel: Erstellen einer Datenbank“ auf Seite 32](#)

Java SE-Beispiel: Lesen einer Tabelle

Führen Sie das Java SE-Beispiel durch, um das Lesen einer Tabelle zu verstehen.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Kontext und Bemerkungen

In diesem Beispiel wird ein PreparedStatement-Objekt aus einer Verbindung und ein ResultSet-Objekt aus dem PreparedStatement-Objekt abgerufen. Die next-Methode für das ResultSet gibt jedes Mal TRUE zurück, wenn eine nachfolgende Zeile abgerufen werden kann. Werte für die Spalten in der aktuellen Zeile können anschließend aus dem ResultSet-Objekt abgerufen werden.

Aufgabe

1. Wechseln Sie zum folgenden Verzeichnis: `%SQLANYSAMPI6%\UltraLite\J2SE`.
2. Führen Sie das CreateDb-Beispiel aus:

```
rundemo CreateDb
```

3. Führen Sie das LoadDb-Beispiel aus:

```
rundemo LoadDb
```

4. Führen Sie den folgenden Befehl aus (mit Berücksichtigung der Groß- und Kleinschreibung):

```
rundemo ReadSeq
```

Ergebnisse

Die Anwendung wird erfolgreich ausgeführt. Ein erstelltes ResultSet-Objekt wird vor die erste Zeile der Ergebnismenge gesetzt. Die ReadSeq-Anwendung liest und zeigt Ergebnismengen nacheinander an.

Siehe auch

- „Java SE-Beispiel: Erstellen einer Datenbank“ auf Seite 32
- „Java SE-Beispiel: Einfügen von Zeilen“ auf Seite 32

Java SE-Beispiel: Inner-Join-Vorgänge

Führen Sie das Java SE-Beispiel durch, um zu verstehen, wie ein Inner-Join-Vorgang durchgeführt wird.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Kontext und Bemerkungen

Jeder Mitarbeiter hat entsprechende Abteilungsinformationen. Der Join-Vorgang verknüpft Daten aus der Employee-Tabelle mit entsprechenden Daten aus der Department-Tabelle. Die Verknüpfung wird mit der Abteilungsnummer in der Employee-Tabelle hergestellt, um zugehörige Informationen in der Department-Tabelle zu finden.

Aufgabe

1. Wechseln Sie zum folgenden Verzeichnis: %SQLANYAMP16%\UltraLiteJ\J2SE.
2. Führen Sie das CreateDb-Beispiel aus:

```
rundemo CreateDb
```

3. Führen Sie das LoadDb-Beispiel aus:

```
rundemo LoadDb
```

4. Führen Sie den folgenden Befehl aus (mit Berücksichtigung der Groß- und Kleinschreibung):

```
rundemo ReadInnerJoin
```

Ergebnisse

Die Anwendungen werden erfolgreich ausgeführt. Die Employee-Tabelle wird gelesen und jede Zeile wird durch einen Join mit der entsprechenden Zeile in der Department-Tabelle verknüpft.

Siehe auch

- „Java SE-Beispiel: Erstellen einer Datenbank“ auf Seite 32
- „Java SE-Beispiel: Einfügen von Zeilen“ auf Seite 32

Java SE-Beispiel: Erstellen einer Verkaufsdatenbank

Führen Sie das Java SE-Beispiel durch, um zu verstehen, wie eine Datenbank erstellt wird.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Wechseln Sie zum folgenden Verzeichnis: %SQLANYSAMPI6%\UltraLiteJ\J2SE.
2. Führen Sie den folgenden Befehl aus (mit Berücksichtigung der Groß- und Kleinschreibung):

```
rundemo CreateSales
```

Ergebnisse

Die Beispielanwendung wird erfolgreich ausgeführt. Eine verkaufsbezogene Datenbank wird erstellt.

Java SE-Beispiel: Aggregation und Gruppierung

Führen Sie das Java SE-Beispiel durch, um die Aggregation und die Gruppierung von Ergebnissen zu verstehen.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Wechseln Sie zum folgenden Verzeichnis: %SQLANYSAMPI6%\UltraLiteJ\J2SE.
2. Führen Sie das CreateSales-Beispiel aus:

```
rundemo CreateSales
```

3. Führen Sie den folgenden Befehl aus (mit Berücksichtigung der Groß- und Kleinschreibung):

```
rundemo SalesReport
```

Ergebnisse

Die Anwendungen werden erfolgreich ausgeführt. Ein Verkaufsbericht wird generiert.

Siehe auch

- [„Java SE-Beispiel: Erstellen einer Verkaufsdatenbank“ auf Seite 34](#)

Java SE-Beispiel: Abrufen von Zeilen in einer alternativen Reihenfolge

Führen Sie das Java SE-Beispiel durch, um zu verstehen, wie Zeilen in einer unterschiedlichen Reihenfolge verarbeitet werden.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Wechseln Sie zum folgenden Verzeichnis: `%SQLANYSAMPI6%\UltraLiteJ\J2SE`.
2. Führen Sie das CreateSales-Beispiel aus:

```
rundemo CreateSales
```

3. Führen Sie den folgenden Befehl aus (mit Berücksichtigung der Groß- und Kleinschreibung):

```
rundemo SortTransactions
```

Ergebnisse

Die Anwendungen werden erfolgreich ausgeführt. Tabellenspalten werden nach Artikelnummer sortiert, wenn sie ausgewählt werden.

Siehe auch

- [„Java SE-Beispiel: Erstellen einer Verkaufsdatenbank“ auf Seite 34](#)

Java SE-Beispiel: Ändern von Tabellendefinitionen

Führen Sie das Java SE-Beispiel durch, um zu verstehen, wie UltraLiteJ Tabellendefinitionen ändern kann.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Kontext und Bemerkungen

Eine Invoice-Tabelle wird geändert, um eine Spaltenlänge von 50 auf 100 Zeichen zu erweitern.

Aufgabe

1. Wechseln Sie zum folgenden Verzeichnis: `%SQLANYSAMPI6%\UltraLiteJ\J2SE`.
2. Führen Sie das CreateSales-Beispiel aus:

```
rundemo CreateSales
```

3. Führen Sie den folgenden Befehl aus (mit Berücksichtigung der Groß- und Kleinschreibung):

```
rundemo Reorg
```

Ergebnisse

Die Anwendungen werden erfolgreich ausgeführt. Die Namensspalte der Invoice-Tabelle wird geändert.

Siehe auch

- „Java SE-Beispiel: Erstellen einer Verkaufsdatenbank“ auf Seite 34

Java SE-Beispiel: Verschlüsseln von Daten

Führen Sie das Java SE-Beispiel durch, um zu verstehen, wie Daten in einer UltraLite Java Edition-Datenbank verschlüsselt werden und wie die Entschlüsselung eine Performance-Verschlechterung verursacht.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Kontext und Bemerkungen

Dieses Beispiel ist für Java SE. Ein Beispiel zur vollständigen BlackBerry-Verschlüsselung finden Sie im Verzeichnis `%SQLANYSAMPI6%\UltraLiteJ\BlackBerryEncryption`.

Aufgabe

1. Wechseln Sie zum folgenden Verzeichnis: `%SQLANYSAMPI6%\UltraLiteJ\J2SE`.
2. Führen Sie den folgenden Befehl aus (mit Berücksichtigung der Groß- und Kleinschreibung):

```
rundemo Encrypted
```

Ergebnisse

Die Beispielanwendung wird erfolgreich ausgeführt. Verschlüsselte Daten werden in die UltraLite Java Edition-Datenbank eingefügt und sie werden entschlüsselt, wenn sie ausgewählt werden.

Java SE-Beispiel: Anzeigen von Schemainformationen für eine Datenbank

Führen Sie das Java SE-Beispiel durch, um zu verstehen, wie Systemtabellen der UltraLite Java Edition-Datenbank durchsucht werden, um die Schemainformationen zu überprüfen.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Wechseln Sie zum folgenden Verzeichnis: `%SQLANYSAMPI6%\UltraLiteJ\J2SE`.
2. Führen Sie das CreateSales-Beispiel aus:

```
rundemo CreateSales
```

3. Führen Sie den folgenden Befehl aus (mit Berücksichtigung der Groß- und Kleinschreibung):

```
rundemo DumpSchema
```

Ergebnisse

Die Daten für jede Zeile der Tabellen werden ebenfalls angezeigt. Die folgende Ausgabe wird angezeigt:

```
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

```

Siehe auch

- [„Java SE-Beispiel: Erstellen einer Verkaufsdatenbank“ auf Seite 34](#)

Java SE-Beispiel: Synchronisieren einer Datenbank

Führen Sie das Java SE-Beispiel durch, in dem eine Verbindung zu einem MobiLink-Server hergestellt wird, um zu verstehen, wie eine Clientdatenbank mit einer konsolidierten SQL Anywhere-Datenbank synchronisiert wird.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Wechseln Sie zum folgenden Verzeichnis: `%SQLANYSAMPI6%\UltraLiteJ\J2SE`.
2. Führen Sie das CreateSales-Beispiel aus:

```
rundemo CreateSales
```

3. Führen Sie den folgenden Befehl aus, um den MobiLink-Server zu starten:

```
start_ml
```

4. Führen Sie den folgenden Befehl aus (mit Berücksichtigung der Groß- und Kleinschreibung):

```
rundemo Sync
```

Ergebnisse

Die Beispielanwendung wird erfolgreich ausgeführt. Der Clientdatenbank synchronisiert die Daten mit der konsolidierten Datenbank.

Nächste Schritte

Fahren Sie den MobiLink-Server herunter.

Siehe auch

- „Java SE-Beispiel: Erstellen einer Verkaufsdatenbank“ auf Seite 34
- „MobiLink-Server herunterfahren“ [[MobiLink - Serveradministration](#)]

Praktische Einführung: Eine Android-Anwendung erstellen

Diese praktische Einführung führt Sie durch die Entwicklung einer Anwendung für Android-Smartphones mithilfe der UltraLiteJ-API und der Eclipse-Umgebung. In dieser praktischen Einführung führen Sie die Anwendung auf einem Windows-Simulator aus.

Die in dieser Einführung verwendete Android-Anwendung befindet sich im Verzeichnis `%SQLANYSAMPI6%\UltraLiteJ\Android\CustDB\`. Der Anwendungscode im Verzeichnis `src\com\sybase\custdb` referenziert die UltraLiteJ-API, um die folgenden Aufgaben auszuführen:

- Erstellung einer entfernten UltraLite-Datenbank.
- SQL-Vorgänge in der Datenbank.
- Datensynchronisation mit der SQL Anywhere-Beispieldatenbank CustDB mithilfe von MobiLink.

Die Verzeichnisse `res\menu` und `res\layout` veranschaulichen das Erstellen von Android-Menüelementen und -Schnittstellen. Sie können diese Dateien über Eclipse anzeigen, wenn Sie das neue Android-Projekt erstellen.

Die Projektdatei `AndroidManifest.xml` wurde geändert, sodass die Android-Anwendung auf das Netzwerk zugreifen kann, was für die Datensynchronisation erforderlich ist. Die folgende Berechtigungsanweisung wurde hinzugefügt:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Erforderliche Software

- Eclipse 3.5.2 oder später
- Android SDK Starter Package
- Android Development Tools (ADT)-Plug-In für Eclipse 1.1 oder später
- SQL Anywhere 16-Beispiele
- UltraLiteJ-API

Kenntnisse und Erfahrungen

- Kenntnisse über Java
- Kenntnisse über Eclipse

Siehe auch

- <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>
- „UltraLiteJ-API-Referenz“ auf Seite 81

Lektion 1: Einrichten eines neuen Android-Projekts

In dieser Lektion erstellen Sie ein neues Android-Projekt mit der integrierten Eclipse-Entwicklungsumgebung.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie die erforderliche Software installiert haben. Siehe [„Praktische Einführung: Eine Android-Anwendung erstellen“ auf Seite 41](#).

Kontext und Bemerkungen

In dieser praktischen Einführung wird davon ausgegangen, dass Sie mit Java und Eclipse vertraut sind.

Aufgabe

1. Kopieren Sie die Android-Bibliotheken in Ihr Android-CustDB-Beispielverzeichnis.

Öffnen Sie eine Eingabeaufforderung, wechseln Sie in das Verzeichnis `%SQLANYSDMP16%\UltraLiteJ\Android\CustDB\` und führen Sie dann den folgenden Befehl aus:

```
setup.bat
```

Die Dateien `UltraLiteJNI16.jar` und `libultralitej16.so` werden in die Verzeichnisse `Android\CustDB\libs` und `Android\CustDB\libs\armeabi` kopiert.

2. Führen Sie Eclipse aus.

Der Standardpfad der Anwendung ist `C:\Eclipse\eclipse.exe`.

3. Geben Sie im Feld **Workspace** (Arbeitsbereich) ein Arbeitsverzeichnis an, das nicht Ihr CustDB-Beispielverzeichnis ist, und klicken Sie dann auf **OK**.
4. Importieren Sie das CustDB-Projekt in Eclipse.
 - a. Klicken Sie auf **File** (Datei) » **Import** (Importieren).
 - b. Erweitern Sie das Verzeichnis **General** und klicken Sie dann auf **Existing Projects into Workspace** (Vorhandene Projekte in den Arbeitsbereich). Klicken Sie auf **Weiter**.
 - c. Geben Sie im Feld **Select Root Directory** (Stammverzeichnis auswählen) das Verzeichnis `%SQLANYSDMP16%\UltraLiteJ\Android\CustDB` an. Wählen Sie **Copy Projects Into Workspace** (Projekte in Arbeitsbereich kopieren) und klicken Sie dann auf **Finish** (Fertig stellen).
5. Stellen Sie sicher, dass in Eclipse der entsprechende Pfad zum Android-SDK angegeben ist.

Hinweis

Sie müssen ein Android-SDK installieren, bevor Sie den Pfad angeben. Weitere Hinweise zu den Voraussetzungen für diese praktische Einführung finden Sie unter [„Praktische Einführung: Eine Android-Anwendung erstellen“ auf Seite 41](#).

- a. Klicken Sie auf **Window** (Fenster) » **Preferences** (Voreinstellungen).
 - b. Klicken Sie im linken Fensterausschnitt auf **Android**.
 - c. Geben Sie im Feld **SDK Location** (SDK-Speicherort) den Speicherort des Android-SDK ein und klicken Sie auf **Apply** (Übernehmen).
Eine Liste der verfügbaren Build-Ziele wird angezeigt.
 - d. Klicken Sie auf **OK**.
6. Stellen Sie sicher, dass in Eclipse der Pfad der UltraLiteJ-Bibliothek angegeben ist.
- a. Klicken Sie auf **File (Datei)** » **Properties (Eigenschaften)**.
 - b. Klicken Sie im linken Fensterausschnitt auf **Java Build Path** (Java-Build-Pfad) » **User libraries** (Benutzerbibliotheken).
 - c. Klicken Sie auf die Registerkarte **Libraries** (Bibliotheken).
 - d. Klicken Sie auf **UltraLiteJNI16.jar** und anschließend auf **Edit** (Bearbeiten).
 - e. Öffnen Sie in Ihrem Arbeitsverzeichnis die Datei `\CustDB\libs\UltraLiteJNI16.jar`.
 - f. Klicken Sie auf **OK**.
7. Fügen Sie den Pfad der UltraLiteJNI-Javadoc-Dokumentation des Projekts hinzu.
- a. Klicken Sie im linken Fensterausschnitt auf **Javadoc Location** (Javadoc-Speicherort).
 - b. Klicken Sie auf **Browse** (Durchsuchen) und öffnen Sie das Verzeichnis `%SQLANY16%\UltraLite\UltraLiteJ\Android\html`.
 - c. Klicken Sie auf **OK**.
 - d. Klicken Sie auf **OK**, um das Fenster zu schließen.
8. Erstellen Sie das Projekt.
- Klicken Sie auf **Project** (Projekt) » **Clean** (Bereinigen) und anschließend auf **OK**.
- Das Projekt sollten ohne Fehler erstellt werden, aber möglicherweise werden auf der Registerkarte **Problems** (Probleme) Warnungen angezeigt.

Ergebnisse

Die UltraLiteJ-API ist in der neuen Android-Anwendung funktionsfähig.

Nächste Schritte

Gehen Sie weiter zu „[Lektion 2: Starten des MobiLink-Servers](#)“ auf Seite 43.

Lektion 2: Starten des MobiLink-Servers

In dieser Lektion starten Sie den MobiLink-Server, um die Synchronisation durchzuführen.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „[Lektion 1: Einrichten eines neuen Android-Projekts](#)“ auf Seite 42.

Aufgabe

- Starten Sie MobiLink, indem Sie den folgenden Befehl aus `%SQLANYSAMPI6%\MobiLink\CustDB\` ausführen:

```
mlsrv16 -v+ -zu+ -c "DSN=SQL Anywhere 16 CustDB;UID=ml_server;PWD=sql" -x  
http(port=80) -ot ml.mls
```

Die Option `-c` verbindet MobiLink mit der SQL Anywhere-Datenbank CustDB. Die Option `-v+` setzt eine hohe Ausführlichkeitsstufe, damit Sie im Meldungsfenster des MobiLink-Servers das Geschehen verfolgen können. Die Option `-x` gibt an, welche Portnummer für die Kommunikation verwendet wird. Die Option `-ot` gibt an, dass eine Logdatei (`ml.mls`) im Verzeichnis erstellt werden soll, in dem Sie den MobiLink-Server gestartet haben.

Ergebnisse

Der MobiLink-Server wurde gestartet.

Nächste Schritte

Gehen Sie weiter zu „[Lektion 3: Ausführen Ihrer Android-Anwendung](#)“ auf Seite 44.

Siehe auch

- „[MobiLink-Serveroptionen](#)“ [[MobiLink - Serveradministration](#)]

Lektion 3: Ausführen Ihrer Android-Anwendung

In dieser Lektion führen Sie Ihre Anwendung über einen Android-Simulator aus.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „[Lektion 1: Einrichten eines neuen Android-Projekts](#)“ auf Seite 42.

Aufgabe

1. Richten Sie Ihr virtuelles Android-Gerät in Eclipse ein.
 - a. Klicken Sie auf **Window** (Fenster) » **AVD Manager**.
 - b. Klicken Sie auf **Neu**.

Das Fenster **Neues virtuelles Android-Gerät (AVD) erstellen (Create New Android Virtual Device (AVD))** wird angezeigt.

- c. Im Feld **Name** geben Sie **my_avd** ein.

- d. Klicken Sie im Feld **Target** (Ziel) auf **Android 2.2 – API Level 8**.
 - e. Klicken Sie auf **Create AVD** (AVD erstellen).
 - f. Schließen Sie das Fenster **AVD Manager**.
2. Im Fenster **Package Explorer** (Paket-Explorer) wählen Sie **CustDB**.
 3. Wählen Sie im Fenster **Run** (Ausführen) die Optionen **Run As** (Ausführen als) » **Android Application** (Android-Anwendung).

Der Android-Simulator wird geladen.

4. Klicken Sie auf **Menu** (Menü).

Ihre Android-Anwendung wird geladen.

Ergebnisse

Die UltraLite-Anwendung wird in einem simulierten Android-Smartphone geladen.

Nächste Schritte

Gehen Sie weiter zu „[Lektion 4: Testen Ihrer Android-Anwendung und Synchronisieren](#)“ auf Seite 45.

Lektion 4: Testen Ihrer Android-Anwendung und Synchronisieren

In dieser Lektion aktualisieren Sie mit Ihrer Android-Anwendung die entfernte UltraLite-Datenbank und synchronisieren die konsolidierte CustDB-Datenbank.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „[Lektion 1: Einrichten eines neuen Android-Projekts](#)“ auf Seite 42.

Aufgabe

1. Vergewissern Sie sich, dass das Feld **Employee ID** (Mitarbeiter-ID) **50**, das Feld **Host** **10.0.2.2** und das Feld **Port** **80** ist. Klicken Sie anschließend auf **Save** (Speichern).

Die Anwendung wird automatisch synchronisiert und aus der konsolidierten CustDB-Datenbank wird eine Menge von Mitarbeitern, Produkten und Bestellungen in die Anwendung heruntergeladen.

2. Klicken Sie im Simulator auf **Menu** (Menü) » **New** (Neu).
3. Im Feld **Customer** (Kunde) wählen Sie **Ace Properties** (Ace-Eigenschaften).
4. Im Feld **Product** (Produkt) wählen Sie **4x8 Drywall x100**.

5. Im Feld **Quantity (Menge)** geben Sie **999** ein.
6. Im Feld **Discount (Rabatt)** geben Sie **25** ein.
7. Klicken Sie auf **OK**, um die neue Bestellung hinzuzufügen.
8. Synchronisieren Sie die Anwendung mit der konsolidierten CustDB-Datenbank.

Klicken Sie im Simulator auf **Menu (Menü)** und dann auf **Sync (Synchronisieren)**.

9. Verbinden Sie sich mit der konsolidierten CustDB-Datenbank mittels Interactive SQL.
 - a. Klicken Sie auf **Start » Programme » SQL Anywhere 16 » Administrationstools » Interactive SQL** oder führen Sie folgenden Befehl aus:

```
dbisql
```
 - b. Klicken Sie auf **ODBC Data Source Name (ODBC-Datenquellenname)** und dann auf **SQL Anywhere 16 CustDB**.
 - c. Klicken Sie auf **Verbinden**.
10. Vergewissern Sie sich, dass die Synchronisation erfolgreich war

Führen Sie die folgende SQL-Anweisung in Interactive SQL aus:

```
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'
```

Die Synchronisation war erfolgreich, wenn ein Bestellungseintrag in Interactive SQL angezeigt wird.

11. Schließen Sie das Simulator-Fenster.

Ergebnisse

Die im Simulator durchgeführten Änderungen werden mit der konsolidierten CustDB-Datenbank synchronisiert.

Nächste Schritte

Gehen Sie weiter zu „Aufräumen“ auf Seite 46.

Aufräumen

Entfernen Sie die zuletzt erstellten Daten der praktischen Einführung von Ihrem Computer.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „[Lektion 1: Einrichten eines neuen Android-Projekts](#)“ auf Seite 42.

Aufgabe

1. Schließen Sie Eclipse.

Klicken Sie auf **Datei » Beenden**.

2. Schließen Sie die Fenster von MobiLink, Interactive SQL und Synchronisationsclient, indem Sie mit der rechten Maustaste auf die einzelnen Elemente in der Taskleiste klicken und auf **Exit** (Beenden) oder **Shut Down** (Herunterfahren) klicken.
3. Setzen Sie die CustDB-Datenbank zurück.

Führen Sie über das Verzeichnis `%SQLANYSAMPI6%\UltraLite\CustDB` den folgenden Befehl aus:

```
makedbs
```

Ergebnisse

Die Daten werden von Ihrem Computer entfernt und Sie können diese praktische Einführung erneut ab der ersten Lektion starten.

Praktische Einführung: Eine BlackBerry-Anwendung erstellen

Diese praktische Einführung führt Sie durch die Entwicklung einer Anwendung für BlackBerry-Smartphones mithilfe der UltraLiteJ-API und der Eclipse-Umgebung. In dieser praktischen Einführung führen Sie die Anwendung auf einem Windows-Simulator aus und stellen sie auf einem BlackBerry-Smartphone bereit. Codebeispiele werden durchgehend in der praktischen Einführung bereitgestellt, und ein vollständiger Programmcode steht am Ende der praktischen Einführung zur Verfügung.

Erforderliche Software

- Eclipse 3.5 oder später
- BlackBerry Java Plug-In für Eclipse 1.1 oder später
- BlackBerry E-Mail- und MDS-Services-Simulator v4.1.4
- BlackBerry JDE 6.0 oder später
- UltraLiteJ-API

Kenntnisse und Erfahrungen

- Kenntnisse über Java
- Kenntnisse über Eclipse

Siehe auch

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

Teil 1: Erstellen einer neuen BlackBerry-Anwendung

In diesem Teil wird beschrieben, wie Sie eine BlackBerry-Anwendung erstellen, die eine Liste von Namen in einer UltraLite Java Edition-Datenbank verwaltet. Im zweiten Teil wird beschrieben, wie Sie die Anwendung mit einem MobiLink-Server synchronisieren.

Lektion 1: Einrichten eines neuen BlackBerry-Projekts

In dieser Lektion erstellen Sie ein neues BlackBerry-Projekt in der integrierten Eclipse-Entwicklungsumgebung.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie die erforderliche Software installiert haben. Siehe „Praktische Einführung: Eine BlackBerry-Anwendung erstellen“ auf Seite 49.

Kontext und Bemerkungen

In dieser praktischen Einführung wird davon ausgegangen, dass Sie mit Java und Eclipse vertraut sind.

Aufgabe

1. Führen Sie Eclipse aus.

Der Standardpfad ist *C:\Eclipse\eclipse.exe*.

2. Geben Sie im Feld **Workspace (Arbeitsbereich)** ein Arbeitsverzeichnis an und klicken Sie dann auf **OK**.

In dieser praktischen Einführung wird davon ausgegangen, dass Sie im Verzeichnis *C:\HelloBlackBerry* arbeiten.

3. Erstellen Sie Ihr neues Projekt.

Klicken Sie auf **File (Datei) » New (Neu) » Project (Projekt)**.

4. Erweitern Sie den **BlackBerry**-Ordner und wählen Sie dann **BlackBerry Project (BlackBerry-Projekt)**.

5. Klicken Sie auf **Next (Weiter)**.

6. Im Feld **Project Name (Projektname)** geben Sie **HelloBlackBerry** ein.

7. Klicken Sie auf **Finish (Fertig stellen)**.

8. Fügen Sie dem Projekt die UltraLiteJ-JAR-Datei hinzu.

- a. Zeigen Sie das Fenster **Package Explorer (Paket-Explorer)** in Eclipse an, wenn es nicht bereits angezeigt wird.

Klicken Sie auf **Window (Fenster) » Show View (Ansicht anzeigen) » Package Explorer (Paket-Explorer)**.

- b. Greifen Sie auf die Paketeigenschaften des Projekts zu.

Klicken Sie auf **HelloBlackBerry** im Fenster **Package Explorer (Paket-Explorer)** und anschließend auf **File (Datei) » Properties (Eigenschaften)**.

- c. Klicken Sie im linken Fensterausschnitt auf **Java Build Path (Java-Build-Pfad)** und anschließend auf die Registerkarte **Libraries (Bibliotheken)**.

- d. Klicken Sie auf **Add External Jars (Externe JAR-Dateien hinzufügen)** und öffnen Sie dann *\UltraLite\UltraLiteJ\BlackBerry4.2\UltraLiteJ16.jar* aus Ihrem SQL Anywhere-Installationsverzeichnis.

9. Fügen Sie den Pfad der UltraLiteJ-Javadoc-Dokumentation des Projekts hinzu.

- a. In der Liste **JARs And Class Folders On The Build Path (JAR-Dateien und Klassenordner auf dem Build-Pfad)** erweitern Sie **UltraLiteJ16.jar** und klicken Sie auf **JavaDoc Location (JavaDoc-Speicherort)**.

- b. Klicken Sie auf **Edit (Bearbeiten)**.

Das Fenster **Javadoc For UltraLiteJ16.Jar** wird angezeigt.

- c. Klicken Sie auf **Browse (Durchsuchen)** und öffnen Sie dann *\UltraLite\UltraLiteJ\BlackBerry4.2\html* aus Ihrem SQL Anywhere-Installationsverzeichnis.

d. Klicken Sie auf **OK**, um das Fenster **Javadoc For UltraLiteJ16.Jar** zu schließen.

10. Klicken Sie auf **OK**, um das Fenster zu schließen.

Ergebnisse

Die UltraLiteJ-API ist in der neuen BlackBerry-Anwendung funktionsfähig.

Nächste Schritte

Gehen Sie weiter zu „[Lektion 2: Schreiben und Testen Ihrer BlackBerry-Anwendung](#)“ auf Seite 51.

Lektion 2: Schreiben und Testen Ihrer BlackBerry-Anwendung

In dieser Lektion erstellen Sie eine Klasse mit einer **main**-Methode in Eclipse, die eine **HomeScreen**-Klasse öffnet, die einen Titel und eine Statusmeldung enthält. Anschließend kompilieren und testen Sie die Anwendung.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „[Lektion 1: Einrichten eines neuen BlackBerry-Projekts](#)“ auf Seite 49.

Aufgabe

1. Fügen Sie Ihrem Projekt eine **Application**-Klasse hinzu.
 - a. Im Fenster **Package Explorer (Paket-Explorer)** erweitern Sie **HelloBlackBerry** und klicken Sie auf **src**.
 - b. Klicken Sie auf **File (Datei) » New (Neu) » Class (Klasse)**.
Das Fenster **New Java Class (Neu Java-Klasse)** wird angezeigt.
 - c. Im Feld **Wie lautet der Name des Projekts?** geben Sie **Application** ein.
 - d. Unter der Option **Which Method Stubs Would You Like To Create (Welche Methoden-Stubs möchten Sie erstellen)** wählen Sie **Public Static Void Main ([String() Args])**.
 - e. Klicken Sie auf **Finish (Fertig stellen)**.

Die Datei *Application.java* befindet sich in Ihrem Projekt im Fenster **Package Explorer (Paket-Explorer)**.

2. Ändern Sie die **Application**-Klasse.

Doppelklicken Sie auf *Application.java* im Fenster **Package Explorer (Paket-Explorer)** und fügen Sie dann einen Konstruktor und eine **main**-Methode hinzu.

Ihr *Application.java*-Code sollte wie der folgende Code aussehen:

```
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. Fügen Sie Ihrem Projekt eine **HomeScreen**-Klasse hinzu.
 - a. Im Fenster **Package Explorer (Paket-Explorer)** erweitern Sie **HelloBlackBerry** und klicken Sie auf **src**.
 - b. Klicken Sie auf **File (Datei) » New (Neu) » Class (Klasse)**.
Das Fenster **New Java Class (Neu Java-Klasse)** wird angezeigt.
 - c. Im Feld **Wie lautet der Name des Projekts?** geben Sie **HomeScreen** ein.
 - d. Klicken Sie auf **Finish (Fertig stellen)**.
Die Datei *HomeScreen.java* befindet sich in Ihrem Projekt im Fenster **Package Explorer (Paket-Explorer)**.
4. Ändern Sie die **HomeScreen**-Klasse, sodass sie einen Titel und Statusmeldungen anzeigt.

Doppelklicken Sie auf *HomeScreen.java* im Fenster **Package Explorer (Paket-Explorer)** und aktualisieren Sie dann den Code, sodass er einen Titel und eine Statusmeldung anzeigt.

Ihr *HomeScreen.java*-Code sollte wie der folgende Code aussehen:

```
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;
}
```

Das **_statusLabel**-Objekt wird als Klassenvariable definiert, damit darauf später aus anderen Teilen der Anwendung zugegriffen werden kann.

5. Führen Sie den Simulator aus.

Klicken Sie im Fenster **Package Explorer (Paket-Explorer)** auf *Application.java* und anschließend auf **Run (Ausführen) » Run As (Ausführen als) » BlackBerry-Simulator**.

Hinweis

Wenn mehrere Projekte in Ihrem Arbeitsbereich geöffnet sind, klicken Sie auf **Run (Ausführen)** » **Run Configurations (Konfigurationen ausführen)**, wählen Sie **HelloBlackBerry** aus und klicken Sie anschließend auf **Run (Ausführen)**.

Das **HelloBlackBerry**-Projekt wird kompiliert. Daraufhin wird das Simulator-Fenster angezeigt.

Vergewissern Sie sich, dass das Projekt ohne Fehler kompiliert wird, indem Sie die Registerkarte **Problems (Probleme)** in Eclipse auswählen.

6. Klicken Sie im Simulator-Menü auf **Simulate (Simulieren)** » **Set IT Policy (IT-Richtlinie einrichten)**.

Das Fenster **Set IT Policy (IT-Richtlinie einrichten)** wird angezeigt.

7. Klicken Sie im Feld **Policy (Richtlinie)** auf **Allow Third Party Apps To Use Persistent Store (Anwendungen von Drittanbietern die Verwendung des beständigen Speichers erlauben)** » >>.
8. Klicken Sie auf **Set (Einstellen)** und dann auf **Close (Schließen)**.
9. Starten Sie Ihre Anwendung.

Im Simulator-Fenster navigieren Sie zu **Downloads** und führen dann die **HelloBlackBerry**-Anwendung aus.

Ein Bildschirm mit der **Hello BlackBerry**-Titelleiste und dem **Status: Started**-Text wird geöffnet.

10. Halten Sie die Simulation an.

Klicken Sie im Simulator-Fenster auf **File (Datei)** » **Exit (Beenden)**.

Ergebnisse

Die BlackBerry-Anwendung wird erfolgreich im Simulator ausgeführt.

Nächste Schritte

Gehen Sie weiter zu [„Lektion 3: Erstellen einer UltraLite Java Edition-Datenbank“](#) auf Seite 53.

Lektion 3: Erstellen einer UltraLite Java Edition-Datenbank

In dieser Lektion schreiben Sie Code, um eine UltraLite Java Edition-Datenbank zu erstellen und eine Verbindung mit ihr herzustellen.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe [„Lektion 1: Einrichten eines neuen BlackBerry-Projekts“](#) auf Seite 49.

Kontext und Bemerkungen

Der Code zum Erstellen einer neuen Datenbank ist in einer Singleton-Klasse namens **DataAccess** definiert und wird vom **HomeScreen**-Konstruktor aufgerufen. Die Verwendung einer Singleton-Klasse gewährleistet, dass jeweils nur eine Datenbankverbindung offen ist. Auch wenn die UltraLiteJ-API mehrere Verbindungen unterstützt, ist es bei der Entwicklung üblich, nur eine einzige Verbindung zu verwenden.

Aufgabe

1. Ändern Sie die **HomeScreen**-Klasse, um ein **DataAccess**-Objekt zu instanziiieren.

Im Folgenden sehen Sie den vollständigen und aktualisierten **HomeScreen**-Klassen-Programmcode:

```
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 gibt möglicherweise Warnmeldungen aus, dass **DataAccess** nicht aufgelöst werden kann. Das **DataAccess**-Objekt wird als Variable auf Klassenebene gehalten, damit andere Teile des Codes darauf zugreifen können. Sie erstellen die **DataAccess**-Klasse im nächsten Schritt.

2. Fügen Sie Ihrem Projekt eine **DataAccess**-Klasse hinzu.
 - a. Im Fenster **Package Explorer (Paket-Explorer)** erweitern Sie **HelloBlackBerry** und klicken Sie auf **src**.
 - b. Klicken Sie auf **File (Datei) » New (Neu) » Class (Klasse)**.
Das Fenster **New Java Class (Neu Java-Klasse)** wird angezeigt.
 - c. Im Feld **Wie lautet der Name des Projekts?** geben Sie **DataAccess** ein.

- d. Klicken Sie auf **Finish (Fertig stellen)**.

Die Datei *DataAccess.java* befindet sich in Ihrem Projekt im Fenster **Package Explorer (Paket-Explorer)**.

3. Ändern Sie die **DataAccess**-Klasse, sodass sie eine **getDataAccess**-Methode enthält, die eine einzelne Datenbankverbindung gewährleistet.

Doppelklicken Sie auf *DataAccess.java* im Fenster **Package Explorer (Paket-Explorer)** und ersetzen Sie dann den Code durch den folgenden Codeausschnitt:

```
import com.ianymwhere.ultralitej16.*;
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 uexl) {
                        if (uexl.getErrorCode() !=
                        ULjException.SQLE_ULTRALITE_DATABASE_NOT_FOUND) {
                            Dialog.alert("Exception: " + uexl.toString() + ".
                            Recreating database...");
                        }
                        _conn = DatabaseManager.createDatabase(config);
                        // _da.createDatabaseSchema();
                    }
                }
            }
            return _da;
        }
        private static Connection _conn;
        private static DataAccess _da;
    }
}
```

Diese Klasse importiert das **com.ianymwhere.ultralitej16**-Paket aus der *UltraLiteJ16.jar*-Datei. Die folgenden Schritte sind erforderlich, um eine UltraLite Java Edition-Datenbank zu erstellen oder eine Verbindung mit ihr herzustellen:

- a. Legen Sie eine Konfiguration fest. In dieser praktischen Einführung wird das Konfigurationsobjekt von der **ConfigObjectStore**-Schnittstelle definiert, mit der Sie eine beständige Datenbank konfigurieren können, die sich im BlackBerry-Objektspeicher befindet.

- b. Versuchen Sie, mit der Datenbank eine Verbindung herzustellen. In dieser praktischen Einführung wird die Datenbank mit der **createDatabase**-Methode erstellt, wenn die Verbindung fehlschlägt. Die Methode gibt dann eine offene Verbindung zurück.
4. Klicken Sie auf **File (Datei) » Save (Speichern)**.
5. Führen Sie den Simulator aus.

Klicken Sie im Fenster **Package Explorer (Paket-Explorer)** auf *Application.java* und anschließend auf **Run (Ausführen) » Run As (Ausführen als) » BlackBerry-Simulator**.

Hinweis

Wenn mehrere Projekte in Ihrem Arbeitsbereich geöffnet sind, klicken Sie auf **Run (Ausführen) » Run Configurations (Konfigurationen ausführen)**, wählen Sie **HelloBlackBerry** aus und klicken Sie anschließend auf **Run (Ausführen)**.

Das **HelloBlackBerry**-Projekt wird kompiliert. Daraufhin wird das Simulator-Fenster angezeigt.

Vergewissern Sie sich, dass das Projekt ohne Fehler kompiliert wird, indem Sie die Registerkarte **Problems (Probleme)** in Eclipse auswählen.

6. Klicken Sie im Simulator-Menü auf **File (Datei) » Load Java Program (Java-Programm laden)**.
7. Navigieren Sie zum Verzeichnis `\UltraLite\UltraLiteJ\BlackBerry4.2\` Ihrer SQL Anywhere-Installation und öffnen Sie die Datei *UltraLiteJ16.cod*.

Hinweis

Gegebenenfalls müssen Sie *UltraLiteJ16.cod* und die DBG-Dateien in das Verzeichnis des Arbeitssimulators (z.B. `C:\Eclipse\plugins\net.rim.ejde.componentpack6.0.0_6.0.0.0.26\components\simulator\`) kopieren, um die Anwendung auszuführen. Wenn die Dateien kopiert wurden, brauchen Sie das Java-Programm nicht über das Simulator-Menü zu laden.

8. Klicken Sie im Simulator-Menü auf **Simulate (Simulieren) » Set IT Policy (IT-Richtlinie einrichten)**.

Das Fenster **Set IT Policy (IT-Richtlinie einrichten)** wird angezeigt.

9. Klicken Sie im Feld **Policy (Richtlinie)** auf **Allow Third Party Apps to Use Persistent Store (Anwendungen von Drittanbietern die Verwendung des beständigen Speichers erlauben)** und anschließend auf **>>**.
10. Klicken Sie auf **Set (Einstellen)** und dann auf **Close (Schließen)**.
11. Starten Sie Ihre Anwendung.

Im Simulator-Fenster navigieren Sie zu **Downloads** und führen dann die **HelloBlackBerry**-Anwendung aus.

Ein Bildschirm mit der Titelleiste **Hello BlackBerry** und dem Text **Status: Connected** wird geöffnet, was bedeutet, dass die Anwendung erfolgreich eine Verbindung mit der UltraLite Java Edition-Datenbank hergestellt hat.

12. Halten Sie die Simulation an.

Klicken Sie im Simulator-Fenster auf **File (Datei) » Exit (Beenden)**.

Ergebnisse

Die Anwendung läuft im Simulator und der Code erstellt eine UltraLite Java Edition-Datenbank.

Nächste Schritte

Gehen Sie weiter zu „[Lektion 4: Erstellen einer Tabelle in der Datenbank](#)“ auf Seite 57.

Siehe auch

- [ConfigObjectStore-Schnittstelle \[BlackBerry\] \[UltraLiteJ\]](#) auf Seite 91
- [DatabaseManager.createDatabase-Methode \[UltraLiteJ\]](#) auf Seite 142

Lektion 4: Erstellen einer Tabelle in der Datenbank

In dieser Lektion aktualisieren Sie den Anwendungscode, um eine Tabelle namens **Names** in der UltraLite Java Edition-Datenbank zu erstellen.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „[Lektion 1: Einrichten eines neuen BlackBerry-Projekts](#)“ auf Seite 49.

Aufgabe

1. Fügen Sie eine neue Methode zur **DataAccess**-Klasse hinzu, die die **Names**-Tabelle erstellt.

Doppelklicken Sie auf *DataAccess.java* im Fenster **Package Explorer (Paket-Explorer)** und fügen Sie dann den folgenden Code nach der **getDataAccess**-Methode ein:

```
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());
    }
}
```

Diese Methode gibt eine Ausnahme aus, wenn die **Names**-Tabelle bereits in der Datenbank vorhanden ist.

Die Tabelle enthält zwei Spalten mit den folgenden Eigenschaften:

Spaltenname	Datentyp	NULL zulassen?	Standardwert	Primärschlüssel?
ID	UUID	Nein	Keine	Ja
Name	Varchar(254)	Nein	Keine	Nein

2. Rufen Sie die **createDatabaseSchema**-Methode aus der **getDataAccess**-Methode auf.

Entfernen Sie die Codekommentare aus der **getDataAccess**-Methode, sodass die **createDatabaseSchema**-Aufrufe wie der folgende Codeausschnitt aussehen:

```
_da.createDatabaseSchema( )
```

3. Vergleichen Sie Ihren **DataAccess**-Code mit dem vollständigen Programmcode der **DataAccess**-Klasse, um sicherzustellen, dass sie identisch sind.
4. Klicken Sie auf **File (Datei) » Save (Speichern)**.
5. Führen Sie den Simulator aus, um zu überprüfen, dass die Anwendung kompiliert und ausgeführt wird.

Klicken Sie im Fenster **Package Explorer (Paket-Explorer)** auf *Application.java* und anschließend auf **Run (Ausführen) » Run As (Ausführen als) » BlackBerry-Simulator**.

Hinweis

Wenn mehrere Projekte in Ihrem Arbeitsbereich geöffnet sind, klicken Sie auf **Run (Ausführen) » Run Configurations (Konfigurationen ausführen)**, wählen Sie **HelloBlackBerry** aus und klicken Sie anschließend auf **Run (Ausführen)**.

Das **HelloBlackBerry**-Projekt wird kompiliert. Daraufhin wird das Simulator-Fenster angezeigt.

Vergewissern Sie sich, dass das Projekt ohne Fehler kompiliert wird, indem Sie die Registerkarte **Problems (Probleme)** in Eclipse auswählen.

6. Klicken Sie im Simulator-Menü auf **File (Datei) » Load Java Program (Java-Programm laden)**.
7. Navigieren Sie zum Verzeichnis `\UltraLite\UltraLiteJ\BlackBerry4.2\` Ihrer SQL Anywhere-Installation und öffnen Sie die Datei *UltraLiteJ16.cod*.

Hinweis

Gegebenenfalls müssen Sie *UltraLiteJ16.cod* und die DBG-Dateien in das Verzeichnis des Arbeitssimulators (z.B. `C:\Eclipse\plugins\net.rim.ejde.componentpack6.0.0_6.0.0.0.26\components\simulator\`) kopieren, um die Anwendung auszuführen. Wenn die Dateien kopiert wurden, brauchen Sie das Java-Programm nicht über das Simulator-Menü zu laden.

8. Klicken Sie im Simulator-Menü auf **Simulate (Simulieren)** » **Set IT Policy (IT-Richtlinie einrichten)**.

Das Fenster **Set IT Policy (IT-Richtlinie einrichten)** wird angezeigt.

9. Klicken Sie auf **Policy (Richtlinie)** » **Allow Third Party Apps to Use Persistent Store (Anwendungen von Drittanbietern die Verwendung des beständigen Speichers erlauben)** und anschließend auf >>.
10. Klicken Sie auf **Set (Einstellen)** und dann auf **Close (Schließen)**.
11. Starten Sie Ihre Anwendung.

Im Simulator-Fenster navigieren Sie zu **Downloads** und führen dann die **HelloBlackBerry**-Anwendung aus.

Ein Bildschirm mit der Titelleiste **Hello BlackBerry** und dem Text **Status: Connected** wird geöffnet, was bedeutet, dass die Anwendung erfolgreich eine Verbindung mit der UltraLite Java Edition-Datenbank hergestellt hat.

12. Halten Sie die Simulation an.

Klicken Sie im Simulator-Fenster auf **File (Datei)** » **Exit (Beenden)**.

Ergebnisse

Die Anwendung führt den neuen Code aus und die **Names**-Tabelle wird in der UltraLite Java Edition-Datenbank erstellt.

Nächste Schritte

Gehen Sie weiter zu „[Lektion 5: Hinzufügen von Daten zur Tabelle](#)“ auf Seite 59.

Siehe auch

- „[DataAccess.java](#)“ auf Seite 75

Lektion 5: Hinzufügen von Daten zur Tabelle

In dieser Lektion füllen Sie die UltraLite Java Edition-Datenbank, indem Sie Ihrer Anwendung mehrere Steuerelemente hinzufügen, den Code zum Einfügen von Daten in die **Names**-Tabelle implementieren, die Anwendung in einem Simulator ausführen und dann die Steuerelemente zum Eingeben von Daten verwenden.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „[Lektion 1: Einrichten eines neuen BlackBerry-Projekts](#)“ auf Seite 49.

Aufgabe

1. Aktualisieren Sie die **HomeScreen**-Klasse, um die Steuerelemente hinzuzufügen.

Doppelklicken Sie auf *HomeScreen.java* im Fenster **Package Explorer (Paket-Explorer)** und fügen Sie den folgenden Code über der **try-catch**-Anweisung ein, die die **getDataAccess**-Methode aufruft.

```
// 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
addItem(_addToListMenuItem);
```

2. Fügen Sie Deklarationen auf Klassenebene für **_nameEditField** und **_nameListField** hinzu und definieren Sie dann ein **MenuItem**-Objekt mit einer **run**-Methode (ist jetzt noch leer). Diese Deklarationen müssen bei den Deklarationen von **_statusLabel** und **_da** stehen.

Fügen Sie den folgenden Code unter der **private DataAccess _da**;-Anweisung ein:

```
private EditField _nameEditField;
private ObjectListField _nameListField;

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

3. Fügen Sie eine neue Methode zur **DataAccess**-Klasse hinzu, die eine Zeile in eine Tabelle einfügt.

Doppelklicken Sie auf *DataAccess.java* im Fenster **Package Explorer (Paket-Explorer)** und fügen Sie dann den folgenden Code nach der **createDatabaseSchema**-Methode ein:

```
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. Fügen Sie Ihrem Projekt eine **NameRow**-Klasse hinzu.
 - a. Im Fenster **Package Explorer (Paket-Explorer)** erweitern Sie **HelloBlackBerry** und klicken Sie auf **src**.
 - b. Klicken Sie auf **File (Datei) » New (Neu) » Class (Klasse)**.
Das Fenster **New Java Class (Neu Java-Klasse)** wird angezeigt.
 - c. Im Feld **Wie lautet der Name des Projekts?** geben Sie **NameRow** ein.
 - d. Klicken Sie auf **Finish (Fertig stellen)**.
Die Datei *NameRow.java* befindet sich in Ihrem Projekt im Fenster **Package Explorer (Paket-Explorer)**.
5. Aktualisieren Sie die **NameRow**-Klasse, sodass sie eine Zeile in der **Names**-Tabelle als Objekt speichern kann.

Doppelklicken Sie auf *NameRow.java* im Fenster **Package Explorer (Paket-Explorer)** und ersetzen Sie dann den Code durch den folgenden Codeausschnitt:

```
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;  
  
}
```

Die **toString**-Methode wird vom **ObjectListField**-Steuerelement verwendet.

6. Fügen Sie eine neue Methode zur **DataAccess**-Klasse hinzu, die eine Zeile in einen Vektor von Objekten einliest.

Doppelklicken Sie auf *DataAccess.java* im Fenster **Package Explorer (Paket-Explorer)** und fügen Sie dann den folgenden Code nach der **insertName**-Methode ein:

```
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. Fügen Sie eine neue Methode zur **HomeScreen**-Klasse hinzu, die den Inhalt der auf dem Bildschirm angezeigten Liste aktualisiert.

Doppelklicken Sie auf *HomeScreen.java* im Fenster **Package Explorer (Paket-Explorer)** und fügen Sie dann den folgenden Code nach der **_addToListMenuItem**-Methode ein:

```
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. Aktualisieren Sie die **HomeScreen**-Klasse, sodass sie die **refreshNameList**-Methode aufruft, wodurch sichergestellt wird, dass die Liste beim Starten der Anwendung gefüllt wird.

Fügen Sie den folgenden Code vor dem Ende des **HomeScreen**-Konstruktors ein:

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

9. Fügen Sie eine neue Methode zur **HomeScreen**-Klasse hinzu, die eine Zeile in die Liste auf dem Bildschirm einfügt.

Fügen Sie den folgenden Code nach der **refreshNameList**-Methode ein:

```
private void onAddToList(){
    String name = _nameEditField.getText();
    _da.insertName(name);
    this.refreshNameList();
    _nameEditField.setText("");
    _statusLabel.setText(name + " added to list");
}
```

10. Aktualisieren Sie die **run**-Methode in der **HomeScreen**-Klasse, sodass sie die **onAddToList**-Methode aufruft.

Ersetzen Sie die Zeile des Codes, die **\\ TODO** mit dem folgenden Codeausschnitt angibt:

```
onAddToList();
```

11. Klicken Sie auf **File (Datei) » Save All (Alle speichern)**.
12. Führen Sie den Simulator aus, um zu überprüfen, dass die Anwendung kompiliert und ausgeführt wird.

Klicken Sie im Fenster **Package Explorer (Paket-Explorer)** auf *Application.java* und anschließend auf **Run (Ausführen) » Run As (Ausführen als) » BlackBerry-Simulator**.

Hinweis

Wenn mehrere Projekte in Ihrem Arbeitsbereich geöffnet sind, klicken Sie auf **Run (Ausführen) » Run Configurations (Konfigurationen ausführen)**, wählen Sie **HelloBlackBerry** aus und klicken Sie anschließend auf **Run (Ausführen)**.

Das **HelloBlackBerry**-Projekt wird kompiliert. Daraufhin wird das Simulator-Fenster angezeigt.

Vergewissern Sie sich, dass das Projekt ohne Fehler kompiliert wird, indem Sie die Registerkarte **Problems (Probleme)** in Eclipse auswählen.

13. Klicken Sie im Simulator-Menü auf **File (Datei) » Load Java Program (Java-Programm laden)**.
14. Navigieren Sie zum Verzeichnis `\UltraLite\UltraLiteJ\BlackBerry4.2\` Ihrer SQL Anywhere-Installation und öffnen Sie die Datei *UltraLiteJ16.cod*.

Hinweis

Gegebenenfalls müssen Sie *UltraLiteJ16.cod* und die DBG-Dateien in das Verzeichnis des Arbeitssimulators (z.B. `C:\Eclipse\plugins\net.rim.ejde.componentpack6.0.0_6.0.0.0.26\components\simulator\`) kopieren, um die Anwendung auszuführen. Wenn die Dateien kopiert wurden, brauchen Sie das Java-Programm nicht über das Simulator-Menü zu laden.

15. Klicken Sie im Simulator-Menü auf **Simulate (Simulieren) » Set IT Policy (IT-Richtlinie einrichten)**.

Das Fenster **Set IT Policy (IT-Richtlinie einrichten)** wird angezeigt.

16. Klicken Sie im Feld **Policy (Richtlinie)** auf **Allow Third Party Apps To Use Persistent Store (Anwendungen von Drittanbietern die Verwendung des beständigen Speichers erlauben) » >>**.

17. Klicken Sie auf **Set (Einstellen)** und dann auf **Close (Schließen)**.

18. Starten Sie Ihre Anwendung.

Im Simulator-Fenster navigieren Sie zu **Downloads** und führen dann die **HelloBlackBerry**-Anwendung aus.

Ein Bildschirm mit der **Hello BlackBerry**-Titelleiste, dem **Status: Connected**-Text und einem **Name**-Feld wird geöffnet.

19. Im name-Feld geben Sie **John Smith** ein.

20. Klicken Sie auf ***EMPTY*** und wählen Sie dann **Add**.

John Smith wird in der Liste angezeigt, was darauf hinweist, dass der Namenseintrag der **Names**-Tabelle in der Datenbank hinzugefügt wurde.

Namen werden in der Datenbank gespeichert, während Sie sie hinzufügen. Sie werden aus der Datenbank abgerufen werden und der Liste hinzugefügt, wenn Sie die Anwendung schließen und erneut öffnen.

21. Halten Sie die Simulation an.

Klicken Sie im Simulator-Fenster auf **File (Datei)** » **Exit (Beenden)**.

Ergebnisse

Die eingegebenen Daten werden in die **Names**-Tabelle in der UltraLite Java Edition-Datenbank eingefügt.

Nächste Schritte

Gehen Sie weiter zu „[Lektion 6: Deployment Ihrer Anwendung auf einem BlackBerry-Smartphone](#)“ auf Seite 64.

Lektion 6: Deployment Ihrer Anwendung auf einem BlackBerry-Smartphone

Diese Lektion beschreibt, wie Sie das Signieren und das Deployment der Anwendung unter Verwendung der BlackBerry Desktop Manager-Software vornehmen.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „[Lektion 1: Einrichten eines neuen BlackBerry-Projekts](#)“ auf Seite 49.

Sie müssen einen Schlüssel von RIM beziehen, damit Sie mit dem BlackBerry Signature-Tool Ihre Anwendung signieren können. Um weitere Hinweise über den Bezug von Schlüsseln zu erhalten, besuchen Sie die BlackBerry Developer Program-Website unter <http://na.blackberry.com/eng/developers/>.

Kontext und Bemerkungen

Auf einem BlackBerry laufende Anwendungen müssen unter Verwendung des BlackBerry Signature-Tools signiert werden. Dieses Tool ist von Research in Motion (RIM) als Teil des BlackBerry JDE-Komponentenpakets verfügbar. Die *UltraLiteJ16.cod*-Datei ist bereits signiert, Sie müssen jedoch die *HelloBlackBerry.cod*-Datei signieren.

Aufgabe

1. Starten Sie das BlackBerry Signature-Tool.
2. Führen Sie den folgenden Befehl aus dem `\bin`-Verzeichnis der BlackBerry-JDE-Installation (z. B. `C:\Programme\Research in Motion\BlackBerry JDE 6.0.0\bin`) aus.

```
start javaw -jar SignatureTool.jar
```
3. Wechseln Sie zu `C:\HelloBlackBerry` und wählen Sie die `HelloBlackBerry.cod`-Datei (Ihre kompilierte Anwendung).
4. Klicken Sie auf **Request To Sign The File (Anforderung zum Signieren der Datei)**.
5. Klicken Sie auf **Close (Schließen)**, um das Signature-Tool zu schließen.
6. Schließen Sie Ihr BlackBerry an Ihren Computer über das USB-Kabel an und stellen Sie sicher, dass der BlackBerry-Desktop Manager das Gerät sehen kann.
7. Klicken Sie auf **Application Loader (Anwendungslader)** und befolgen Sie die Anweisungen im Assistenten.
8. Navigieren Sie zur `HelloBlackBerry.alx`-Datei und fügen Sie sie Ihrem Gerät hinzu.
9. Navigieren Sie zur `BlackBerry4.2\UltraLiteJ.alx`-Datei und fügen Sie sie Ihrem Gerät hinzu.

Ergebnisse

Das Deployment der Anwendung wird durchgeführt und Sie können die Anwendung auf Ihrem BlackBerry-Smartphone ausführen.

Nächste Schritte

Gehen Sie weiter zu „[Teil 2: MobiLink zum Synchronisieren der BlackBerry-Anwendung verwenden](#)“ auf Seite 65.

Teil 2: MobiLink zum Synchronisieren der BlackBerry-Anwendung verwenden

Dieser Teil der praktischen Einführung erweitert die BlackBerry-Anwendung, um die MobiLink-Synchronisation zu unterstützen. Sie führen folgende Aufgaben durch:

- Erstellen Sie eine SQL Anywhere-Datenbank, die mit Ihrer UltraLite Java Edition-Datenbank synchronisiert werden kann.
- Starten Sie einen MobiLink-Server, um Synchronisationen zu verarbeiten.
- Aktualisieren Sie Ihre BlackBerry-Anwendung, um die MobiLink-Synchronisation zu unterstützen.

- Synchronisieren Sie die BlackBerry-Anwendung mit der konsolidierten Datenbank.

Lektion 1: Einrichten der konsolidierten MobiLink-Datenbank

In dieser Lektion erstellen Sie eine SQL Anywhere-Datenbank.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie die erforderliche Software installiert haben und Teil 1 dieser praktischen Einführung durchgeführt haben. Siehe „[Praktische Einführung: Eine BlackBerry-Anwendung erstellen](#)“ auf Seite 49.

Kontext und Bemerkungen

Die Datensynchronisation erfordert eine konsolidierte MobiLink-Datenbank zur Synchronisation mit der UltraLiteJ-Datenbank.

Aufgabe

1. Erstellen Sie ein Arbeitsverzeichnis, um die SQL Anywhere-Datenbank zu speichern.

In dieser praktischen Einführung wird davon ausgegangen, dass Sie im Verzeichnis *c:\HelloBlackBerry\database* arbeiten.

2. Führen Sie den folgenden Befehl aus, um eine leere SQL Anywhere-Datenbank mit der DBA Benutzer-ID "DBA" und dem Kennwort "sql" zu erstellen:

```
dbinit -dba DBA,sql HelloBlackBerry.db
```

3. Erstellen Sie eine ODBC-Datenquelle, um mit der Datenbank eine Verbindung herzustellen.
 - a. Wählen Sie **Start » Programme » SQL Anywhere 16 » Administrationstools » ODBC-Datenquellen-Administrator**.
 - b. Klicken Sie auf der Registerkarte **Benutzer-DSN** auf **Hinzufügen**.
 - c. Klicken Sie im Fenster **Neue Datenquelle erstellen** auf **SQL Anywhere 16** und auf **Fertig stellen**.
 - d. Klicken Sie auf die Registerkarte **ODBC**.
 - e. Im Feld **Datenquellenname** geben Sie **HelloBlackBerry** ein.
 - f. Klicken Sie auf die Registerkarte **Login**.
 - g. Im Feld **Benutzer-ID** geben Sie **DBA** ein.
 - h. Im Feld **Kennwort** geben Sie **sql** ein.
 - i. Klicken Sie in der Liste **Aktion** auf **Eine Datenbank auf diesem Computer starten und eine Verbindung herstellen**.
 - j. Im Feld **Datenbankdatei** geben Sie *c:\tutorial\database\HelloBlackBerry.db* ein.

- k. Im Feld **Servername** geben Sie **HelloBlackBerry** ein.
 - l. Deaktivieren Sie die Option **Datenbank nach der letzten getrennten Verbindung herunterfahren**.
 - m. Klicken Sie auf **OK**.
 - n. Klicken Sie auf **OK** im Fenster **ODBC-Datenquellenadministrator**.
4. Führen Sie den folgenden Befehl aus, um Interactive SQL zu starten und mit der SQL Anywhere-Datenbank zu verbinden:

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

5. Führen Sie in Interactive SQL die folgende SQL-Anweisung aus, um die **Names**-Tabelle in der konsolidierten Datenbank zu erstellen:

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

6. Schließen Sie Interactive SQL.

Klicken Sie auf **Datei » Beenden**.

Ergebnisse

Die SQL Anywhere-Datenbank wird erstellt.

Nächste Schritte

Gehen Sie weiter zu [„Lektion 2: Einrichten eines MobiLink-Servers und Deployment eines Synchronisationsmodells“](#) auf Seite 67.

Lektion 2: Einrichten eines MobiLink-Servers und Deployment eines Synchronisationsmodells

In dieser Lektion verwenden Sie Sybase Central, um Ihre konsolidierte Datenbank für die Synchronisation vorbereiten.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe [„Lektion 1: Einrichten der konsolidierten MobiLink-Datenbank“](#) auf Seite 66.

Aufgabe

1. Klicken Sie auf **Start » Programme » SQL Anywhere 16 » Administrationstools » Sybase Central**.

2. Klicken Sie auf **Extras » MobiLink 16 » Neues Projekt**.

Der **Assistent zum Erstellen eines Projekts** wird angezeigt.

3. Im Feld **Name** geben Sie **rim_project** ein.
4. Im Feld **Wo soll das neue Projekt gespeichert werden?** geben Sie **C:\HelloBlackBerry\database** ein und klicken dann auf **Weiter**.
5. Geben Sie im Feld **Anzeigenname der Datenbank** **HelloBlackBerry** ein.
6. Klicken Sie auf **Bearbeiten**.
7. Führen Sie die folgenden Aufgaben auf der Seite **Mit einer allgemeinen ODBC-Datenbank verbinden** durch:
 - a. Im Feld **Benutzer-ID** geben Sie **DBA** ein.
 - b. Im Feld **Kennwort** geben Sie **sql** ein.
 - c. Im Feld **ODBC-Datenquellennamen** klicken Sie auf **Durchsuchen** und wählen **HelloBlackBerry**.
 - d. Klicken Sie auf **OK** und dann auf **Speichern**.
8. Wählen Sie die Option **Kennwort speichern** und klicken Sie auf **Weiter**.
9. Stellen Sie sicher, dass nur die **Names**-Tabelle aus der Liste **Welche konsolidierten Datenbanktabellen und Spalten sollen in der entfernten Datenbank enthalten sein?** ausgewählt ist und klicken Sie in den folgenden Dialogfeldern auf **Weiter** bis zum Ende des Assistenten.

Ein Synchronisationsmodell wurde erstellt.

10. Rechtsklicken Sie auf das neue Synchronisationsmodell und wählen Sie die Option **Eigenschaften**.
11. Geben Sie in das erste Textfeld **HelloBlackBerrySyncModel** ein, klicken Sie auf **Übernehmen** und dann auf **OK**.
12. Erweitern Sie im linken Fensterausschnitt von Sybase Central unter **MobiLink 16 rim_project, Synchronisationsmodelle** und dann **HelloBlackBerrySyncModel**.
13. Klicken Sie auf **Datei » Speichern**.
14. Vergewissern Sie sich, dass unter der Option **Deployment-Details für eine oder mehrere der folgenden Komponenten angeben** nur die Option **Konsolidierte Datenbank** ausgewählt ist. Klicken Sie auf **Next (Weiter)**.
15. Führen Sie auf der Seite **Deployment-Ziel der konsolidierten Datenbank** folgende Aufgaben aus:
 - a. Wählen Sie **Änderungen in der folgenden SQL-Datei speichern** und akzeptieren Sie den Standardspeicherort für die Datei.

MobiLink generiert eine *SQL*-Datei, die Änderungen an der konsolidierten Datenbank vornimmt, um sie für die Synchronisation einzurichten. Sie können die *SQL*-Datei später überprüfen und eigene Änderungen vornehmen. Sie müssen dann die *.sql*-Datei manuell ausführen.

- b. Übernehmen Sie die Änderungen sofort in die konsolidierte Datenbank.
Wählen Sie **Mit der konsolidierten Datenbank verbinden, um die Änderungen direkt zu übernehmen**.
- c. Wählen Sie die konsolidierte Datenbank **HelloBlackBerry** aus der Liste aus.
- d. Klicken Sie auf **Next (Weiter)**.

In einer Eingabeaufforderung werden Sie aufgefordert, das Verzeichnis *consolidated* zu erstellen. Klicken Sie auf **Ja**.

16. Auf der Seite **MobiLink-Benutzer- und Synchronisationsprofil** geben Sie **mluser** für den Benutzernamen und **mlpassword** für das Kennwort ein. Klicken Sie anschließend auf **Fertig stellen**.

Ergebnisse

Ein MobiLink-Projekt wird erstellt und das Synchronisationsmodell für die konsolidierte Datenbank kann nun per Deployment bereitgestellt werden.

Nächste Schritte

Gehen Sie weiter zu „[Lektion 3: Hinzufügen von MobiLink-Unterstützung zu Ihrer BlackBerry-Anwendung](#)“ auf Seite 69.

Lektion 3: Hinzufügen von MobiLink-Unterstützung zu Ihrer BlackBerry-Anwendung

In dieser Lektion fügen Sie Ihrer Anwendung Synchronisationsfunktionalität hinzu.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „[Lektion 1: Einrichten der konsolidierten MobiLink-Datenbank](#)“ auf Seite 66.

Aufgabe

1. Aktualisieren Sie die **HomeScreen**-Klasse, um eine **Sync**-Menüoption hinzuzufügen.

Doppelklicken Sie auf *HomeScreen.java* im Fenster **Package Explorer (Paket-Explorer)** und fügen Sie den folgenden Code über der **try-catch**-Anweisung ein, die die **getDataAccess**-Methode aufruft.

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

2. Aktualisieren Sie die **HomeScreen**-Klasse, um eine neue Methode hinzuzufügen, die die Menüoption in den Klassenvariablen-Deklarationen definiert.

Fügen Sie den folgenden Code unter der **_addToListMenuItem**-Methode ein:

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

```
        onSync();  
    }  
};
```

3. Aktualisieren Sie die **HomeScreen**-Klasse, um die **onSync**-Methode hinzuzufügen, die im vorherigen Schritt aufgerufen wird.

Fügen Sie den folgenden Code unter der **onAddToList**-Methode ein:

```
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. Aktualisieren Sie die **DataAccess**-Klasse, um die **_syncParms**-Variable festzulegen.

Doppelklicken Sie auf *DataAccess.java* im Fenster **Package Explorer (Paket-Explorer)** und fügen Sie dann den folgenden Code unter dem **private static DataAccess _da;**-Aufruf ein:

```
private static SyncParms _syncParms;
```

5. Aktualisieren Sie die **DataAccess**-Klasse, um eine **sync**-Methode hinzuzufügen.

Fügen Sie den folgenden Code unter der **getNameVector**-Methode ein:

Hinweis

Sie müssen *your-host-name* durch den Namen Ihres Computers ersetzen. Es ist nicht möglich, diesen Begriff in Ihrer Anwendung verwenden.

```
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;  
    }  
}
```

Das Synchronisationsparameter-Objekt, **_syncParms**, enthält den Benutzernamen und das Kennwort, das Sie beim Deployment des Synchronisationsmodells angegeben haben. Es enthält auch den Namen

des von Ihnen erstellten Synchronisationsmodells. In MobiLink kann sich dieser Name auf die Synchronisationsversion oder auf einen Satz von Synchronisationslogik beziehen, dessen Deployment auf Ihre konsolidierte Datenbank vorgenommen wurde.

Das Datenstromparameter-Objekt, **StreamHTTTParms**, gibt den Hostnamen und die Portnummer des MobiLink-Servers an. Wenn Sie den MobiLink-Server in der nächsten Lektion starten, verwenden Sie Ihren eigenen Computernamen für Simulatortests und wählen einen verfügbaren Port aus.

Hinweis

Wenn Sie mit einem Gerät arbeiten, verwenden Sie einen extern sichtbaren Computer oder einen Computer, auf den Sie über den BlackBerry Enterprise Server zugreifen können, dem Ihr Gerät zugeordnet ist, z.B. den von Sybase gehosteten Relay Server. Weitere Hinweise zum Relay Server finden Sie unter „[Einführung in den Relay Server](#)“ [[Relay Server](#)].

6. Klicken Sie auf **File (Datei)** » **Save All (Alle speichern)**.

Ergebnisse

Die BlackBerry-Anwendung ist in der Lage, eine Synchronisation mit der konsolidierten Datenbank durchzuführen.

Nächste Schritte

Gehen Sie weiter zu „[Lektion 4: Starten des MobiLink-Servers und Synchronisieren der Anwendung](#)“ auf Seite 71.

Lektion 4: Starten des MobiLink-Servers und Synchronisieren der Anwendung

In dieser Lektion starten Sie den MobiLink-Server und synchronisieren die UltraLite Java Edition-Datenbank mit der konsolidierten SQL Anywhere-Datenbank.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „[Lektion 1: Einrichten der konsolidierten MobiLink-Datenbank](#)“ auf Seite 66.

Kontext und Bemerkungen

Bevor Sie die BlackBerry-Anwendung ausführen und synchronisieren können, muss der MobiLink-Server laufen. Der MDS-Simulator muss ebenfalls laufen, um einen Kommunikationskanal zwischen dem Gerätsimulator und MobiLink bereitzustellen.

Aufgabe

1. Starten Sie MobiLink, indem Sie den folgenden Befehl von `c:\HelloBlackBerry\database\` ausführen:

```
mlsrv16 -c "DSN=HelloBlackBerry" -v+ -x http(port=8081) -ot ml.mls
```

Die Option `-c` verbindet MobiLink mit der SQL Anywhere-Datenbank. Die Option `-v+` setzt eine hohe Ausführlichkeitsstufe, damit Sie im Meldungsfenster des MobiLink-Servers das Geschehen verfolgen können. Die Option `-x` gibt an, welche Portnummer für die Kommunikation verwendet wird. Die Option `-ot` gibt an, dass eine Logdatei (*ml.mls*) im Verzeichnis erstellt werden soll, in dem Sie den MobiLink-Server gestartet haben.

2. Führen Sie den MDS-Simulator aus, sodass der BlackBerry-Simulator über ein Netzwerk kommunizieren kann.

Klicken Sie auf **Start » Programme » Research In Motion » BlackBerry Email And MDS Services Simulator 4.1.4 (BlackBerry E-Mail- und MDS-Services-Simulator 4.1.4) » MDS**.

3. Fügen Sie der konsolidierten MobiLink-Datenbank Namen hinzu, damit Ihre Anwendung die UltraLite Java Edition-Datenbank bei der Synchronisation aktualisiert.
 - a. Führen Sie den folgenden Befehl aus, um Interactive SQL zu starten und mit der SQL Anywhere-Datenbank zu verbinden:

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

- b. Führen Sie in Interactive SQL die folgende SQL-Anweisung aus, um Namen zur **Names**-Tabelle hinzuzufügen:

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

- c. Schließen Sie Interactive SQL.

Klicken Sie auf **Datei » Beenden**.

4. Führen Sie die Simulator aus Eclipse aus.

Klicken Sie im Fenster **Package Explorer (Paket-Explorer)** auf *Application.java* und anschließend auf **Run (Ausführen) » Run As (Ausführen als) » BlackBerry-Simulator**.

Hinweis

Wenn mehrere Projekte in Ihrem Arbeitsbereich geöffnet sind, klicken Sie auf **Run (Ausführen) » Run Configurations (Konfigurationen ausführen)**, wählen Sie **HelloBlackBerry** aus und klicken Sie anschließend auf **Run (Ausführen)**.

Das **HelloBlackBerry**-Projekt wird kompiliert. Daraufhin wird das Simulator-Fenster angezeigt.

Vergewissern Sie sich, dass das Projekt ohne Fehler kompiliert wird, indem Sie die Registerkarte **Problems (Probleme)** in Eclipse auswählen.

5. Klicken Sie im Simulator-Menü auf **File (Datei) » Load Java Program (Java-Programm laden)**.
6. Navigieren Sie zum Verzeichnis *\UltraLite\UltraLiteJ\BlackBerry4.2\ Ihrer SQL Anywhere-Installation* und öffnen Sie die Datei *UltraLiteJ16.cod*.

Hinweis

Gegebenenfalls müssen Sie *UltraLiteJ16.cod* und die DBG-Dateien in das Verzeichnis des Arbeitssimulators (z.B. *C:\Eclipse\plugins\net.rim.ejde.componentpack6.0.0_6.0.0.0.26\components\simulator*) kopieren, um die Anwendung auszuführen. Wenn die Dateien kopiert wurden, brauchen Sie das Java-Programm nicht über das Simulator-Menü zu laden.

7. Klicken Sie im Simulator-Menü auf **Simulate (Simulieren)** » **Set IT Policy (IT-Richtlinie einrichten)**.

Das Fenster **Set IT Policy (IT-Richtlinie einrichten)** wird angezeigt.

8. Klicken Sie im Feld **Policy (Richtlinie)** auf **Allow Third Party Apps To Use Persistent Store (Anwendungen von Drittanbietern die Verwendung des beständigen Speichers erlauben)** » >>.
9. Klicken Sie auf **Set (Einstellen)** und dann auf **Close (Schließen)**.
10. Starten Sie Ihre Anwendung.

Im Simulator-Fenster navigieren Sie zu **Downloads** und führen dann die **HelloBlackBerry**-Anwendung aus.

Ein Bildschirm mit der **Hello BlackBerry**-Titelleiste, dem **Status: Connected**-Text und einem **Name**-Feld wird geöffnet.

11. Synchronisieren Sie die Anwendung mit dem MobiLink-Server.

Klicken Sie auf ***EMPTY*** und wählen Sie dann **Sync**.

Jane Smith und **David Smith** erscheinen in der Liste, die angibt, dass die Anwendung mit der konsolidierten MobiLink-Datenbank synchronisiert werden konnte. Wenn Sie mit Interactive SQL die Namen in der **Names**-Tabelle abfragen, sollten Sie sehen, dass die Namen, die Sie in den Simulator eingegeben haben, den Server erreicht haben.

12. Halten Sie die Simulation an.

Klicken Sie im Simulator-Fenster auf **File (Datei)** » **Exit (Beenden)**.

Ergebnisse

Die UltraLite Java Edition-Datenbank und die konsolidierte SQL Anywhere Datenbank werden synchronisiert.

Nächste Schritte

Gehen Sie weiter zu „[Aufräumen](#)“ auf Seite 73.

Aufräumen

Entfernen Sie die zuletzt erstellten Daten der praktischen Einführung von Ihrem Computer.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „[Teil 1: Erstellen einer neuen BlackBerry-Anwendung](#)“ auf Seite 49.

Aufgabe

1. Schließen Sie Eclipse.

Klicken Sie auf **Datei** » **Beenden**.

2. Schließen Sie das Befehlsfenster, das den MDS-Simulator ausführt.
3. Schließen Sie Sybase Central und Interactive SQL, indem Sie mit der rechten Maustaste auf die einzelnen Anzeigen in der Tastleiste klicken und **Exit (Beenden)** wählen.
4. Löschen Sie die **HelloBlackBerry**-Datenquelle:
 - a. Starten Sie den ODBC-Datenquellen-Administrator.
Wählen Sie **Start** » **Programme** » **SQL Anywhere 16** » **Administrationstools** » **ODBC-Datenquellen-Administrator**.
 - b. In der Liste **Benutzerdatenquellen** wählen Sie **HelloBlackBerry** und klicken Sie dann auf **Entfernen**.
 - c. Schließen Sie den ODBC-Datenquellenadministrator.
5. Löschen Sie das *C:\HelloBlackBerry*-Verzeichnis.

Ergebnisse

Die Daten werden von Ihrem Computer entfernt und Sie können diese praktische Einführung erneut ab der ersten Lektion starten.

Programmcode der praktischen Einführung

Dieser Abschnitt stellt den vollständigen Code für die vorhergehende praktische Einführung bereit. In den praktischen Einführungen werden vier Java-Klassen verwendet. Diese befinden sich auch unter *%SQLANYSAMPI6%\UltraLiteJ\BlackBerry\HelloBlackBerry\myapp*.

Siehe auch

- „[Teil 1: Erstellen einer neuen BlackBerry-Anwendung](#)“ auf Seite 49
- „[Teil 2: MobiLink zum Synchronisieren der BlackBerry-Anwendung verwenden](#)“ auf Seite 65

Application.java

```
//  
*****  
// Copyright © 2013 SAP AG oder ein SAP-Konzernunternehmen. All 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 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 © 2013 SAP AG oder ein SAP-Konzernunternehmen. All 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 code.
//
// *****
package myapp;

import com.ianywhere.ultralitej16.*;

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 © 2013 SAP AG oder ein SAP-Konzernunternehmen. All 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 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 © 2013 SAP AG oder ein SAP-Konzernunternehmen. All 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 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;  
}
```

UltraLiteJ-API-Referenz

Die folgende Liste beschreibt einige der häufig verwendeten API-Objekte:

- **DatabaseManager** Stellt Methoden für die Verwaltung von Datenbankverbindungen bereit, z.B. CreateDatabase und Connect.
- **Connection** Repräsentiert eine Verbindung zu einer UltraLite-Datenbank. Sie können ein oder mehrere Connection-Objekte erstellen.
- **SyncParms** Synchronisiert die UltraLite-Datenbank mit einem MobiLink-Server.
- **PreparedStatement, ResultSet** Erstellt Dynamic SQL-Anweisungen, führt Abfragen aus, führt INSERT-, UPDATE- und DELETE-Anweisungen aus und bietet programmierseitige Kontrolle über Datenbank-Ergebnismengen.

Paket [Android]

`com.ianywhere.ultralitejni16`

Paket [BlackBerry]

`com.ianywhere.ultralitej16`

Siehe auch

- [„Hinweise zum Android- und BlackBerry-Setup“ auf Seite 4](#)
- [DatabaseManager-Klasse \[UltraLiteJ\] auf Seite 138](#)
- [Connection-Schnittstelle \[UltraLiteJ\] auf Seite 105](#)
- [SyncParms-Klasse \[UltraLiteJ\] auf Seite 250](#)
- [PreparedStatement-Schnittstelle \[UltraLiteJ\] auf Seite 180](#)
- [ResultSet-Schnittstelle \[UltraLiteJ\] auf Seite 197](#)

ColumnSchema-Schnittstelle

Gibt das Schema für eine Spalte an.

Syntax

```
public interface ColumnSchema
```

Mitglieder

Alle Mitglieder der ColumnSchema-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
COLUMN_DEFAULT_AUTOFILNAME-Variable	Gibt das Spalten-Standardattribut AUTOFILNAME an.

Name	Beschreibung
COLUMN_DEFAULT_AUTOINC-Variable	Gibt das Spalten-Standardattribut AUTOINCREMENT an.
COLUMN_DEFAULT_CONSTANT-Variable	Gibt das Spalten-Standardattribut constant an.
COLUMN_DEFAULT_CURRENT_DATE-Variable	Gibt das Spalten-Standardattribut CURRENT DATE (aktuelles Datum: Jahr, Monat, Tag) an.
COLUMN_DEFAULT_CURRENT_TIME-Variable	Gibt das Spalten-Standardattribut CURRENT TIME an.
COLUMN_DEFAULT_CURRENT_TIMESTAMP-Variable	Gibt das Spalten-Standardattribut CURRENT TIMESTAMP an.
COLUMN_DEFAULT_CURRENT_UTC_TIMESTAMP-Variable	Gibt das Spalten-Standardattribut CURRENT UTC TIMESTAMP an.
COLUMN_DEFAULT_GLOBAL_AUTOINC-Variable	Gibt das Spalten-Standardattribut GLOBAL AUTOINCREMENT an.
COLUMN_DEFAULT_NONE-Variable	Gibt an, dass die Spalte kein Standardattribut hat.
COLUMN_DEFAULT_UNIQUE_ID-Variable	Gibt das Spalten-Standardattribut new_unique_identifizier an.

Bemerkungen

Diese Schnittstelle enthält nur Konstanten für verschiedene Spalten-Standardwerte, die in der column_default-Spalte der syscolumn-Systemtabelle gespeichert sind.

COLUMN_DEFAULT_AUTOFILENAME-Variable

Gibt das Spalten-Standardattribut AUTOFILENAME an.

Syntax

```
final byte ColumnSchema.COLUMN_DEFAULT_AUTOFILENAME
```

Bemerkungen

Wenn eine VARCHAR-Spalte diesen Standardwert hat, ist sie die filename-Spalte in einer externen Blob-Definition.

Wenn eine Spalte diese Art von Standardwert hat, enthält die column_default_value-Spalte der TableSchema.SYS_COLUMNS-Systemtabelle die Präfix- und Erweiterungszeichenfolgen aus der externen Blob-Definition, und zwar in der Form 'Präfix|Erweiterung'.

Sie können den Standardwert von vorhandenen Tabellen ermitteln, indem Sie die column_default-Spalte der TableSchema.SYS_COLUMNS-Systemtabelle abfragen.

Siehe auch

- [TableSchema.SYS_COLUMNS-Variable \[UltraLiteJ\] auf Seite 282](#)

COLUMN_DEFAULT_AUTOINC-Variable

Gibt das Spalten-Standardattribut AUTOINCREMENT an.

Syntax

```
final byte ColumnSchema.COLUMN_DEFAULT_AUTOINC
```

Bemerkungen

Wenn das AUTOINCREMENT-Attribut verwendet wird, muss die Spalte einer der Ganzzahl-Datentypen oder ein numerisch exakter Typ sein. Ist bei einer INSERT-Anweisung kein Wert für die autoincrement-Spalte vorgegeben, wird ein eindeutiger Wert erstellt, der größer ist als alle anderen Werte in der Spalte. Wenn eine INSERT-Anweisung einen Wert für die Spalte festlegt, der größer als der aktuelle Maximalwert für die Spalte ist, wird der Wert als Startpunkt für nachfolgende Einfügungen verwendet.

In UltraLiteJ wird der selbstinkrementierende Wert beim Erstellen der Tabelle nicht auf 0 gesetzt und das AUTOINCREMENT-Attribut generiert negative Zahlen, wenn für die Spalte ein Datentyp mit Vorzeichen verwendet wird. Sie sollten daher AUTOINCREMENT-Spalten als Integer-Datentyp ohne Vorzeichen deklarieren, um die Verwendung negativer Werte zu verhindern.

Sie können den Standardwert von vorhandenen Tabellen ermitteln, indem Sie die column_default-Spalte der TableSchema.SYS_COLUMNS-Systemtabelle abfragen.

Siehe auch

- [TableSchema.SYS_COLUMNS-Variable \[UltraLiteJ\] auf Seite 282](#)

COLUMN_DEFAULT_CONSTANT-Variable

Gibt das Spalten-Standardattribut constant an.

Syntax

```
final byte ColumnSchema.COLUMN_DEFAULT_CONSTANT
```

Bemerkungen

Sie können den Standardwert von vorhandenen Tabellen ermitteln, indem Sie die column_default-Spalte der TableSchema.SYS_COLUMNS-Systemtabelle abfragen.

Siehe auch

- [TableSchema.SYS_COLUMNS-Variable \[UltraLiteJ\] auf Seite 282](#)

COLUMN_DEFAULT_CURRENT_DATE-Variable

Gibt das Spalten-Standardattribut CURRENT DATE (aktuelles Datum: Jahr, Monat, Tag) an.

Syntax

```
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT_DATE
```

Bemerkungen

Siehe "CURRENT DATE-Spezialwert" unter "Spezialwerte in UltraLite" in der SQL Anywhere-Dokumentation.

Sie können den Standardwert von vorhandenen Tabellen ermitteln, indem Sie die column_default-Spalte der TableSchema.SYS_COLUMNS-Systemtabelle abfragen.

Siehe auch

- [TableSchema.SYS_COLUMNS-Variable \[UltraLiteJ\] auf Seite 282](#)

COLUMN_DEFAULT_CURRENT_TIME-Variable

Gibt das Spalten-Standardattribut CURRENT TIME an.

Syntax

```
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT_TIME
```

Bemerkungen

Siehe "CURRENT TIME-Spezialwert" unter "Spezialwerte in UltraLite" in der SQL Anywhere-Dokumentation.

Sie können den Standardwert von vorhandenen Tabellen ermitteln, indem Sie die column_default-Spalte der TableSchema.SYS_COLUMNS-Systemtabelle abfragen.

Siehe auch

- [TableSchema.SYS_COLUMNS-Variable \[UltraLiteJ\] auf Seite 282](#)

COLUMN_DEFAULT_CURRENT_TIMESTAMP-Variable

Gibt das Spalten-Standardattribut CURRENT TIMESTAMP an.

Syntax

```
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT_TIMESTAMP
```

Bemerkungen

Diese Konstante kombiniert den CURRENT DATE-Wert und den CURRENT TIME-Wert zu einem TIMESTAMP-Wert, der Jahr, Monat, Tag, Stunde, Minute, Sekunde und Sekundenbruchteil enthält. Die

Gesamtstellenzahl des Bruchteils ist auf 3 Dezimalstellen festgelegt. Die Genauigkeit dieser Konstante ist durch die Genauigkeit der Systemuhr begrenzt.

Siehe "CURRENT TIMESTAMP-Spezialwert" unter "Spezialwerte in UltraLite" in der SQL Anywhere-Dokumentation.

Sie können den Standardwert von vorhandenen Tabellen ermitteln, indem Sie die column_default-Spalte der TableSchema.SYS_COLUMNS-Systemtabelle abfragen.

Siehe auch

- [TableSchema.SYS_COLUMNS-Variable \[UltraLiteJ\] auf Seite 282](#)

COLUMN_DEFAULT_CURRENT_UTC_TIMESTAMP-Variable

Gibt das Spalten-Standardattribut CURRENT UTC TIMESTAMP an.

Syntax

```
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT_UTC_TIMESTAMP
```

Bemerkungen

Diese Konstante kombiniert den CURRENT DATE-Wert und den CURRENT TIME-Wert zu einem UTC TIMESTAMP-Wert, der Jahr, Monat, Tag, Stunde, Minute, Sekunde und Sekundenbruchteil in GMT enthält. Die Gesamtstellenzahl des Bruchteils ist auf 3 Dezimalstellen festgelegt. Die Genauigkeit dieser Konstante ist durch die Genauigkeit der Systemuhr begrenzt.

Siehe "CURRENT UTC TIMESTAMP-Spezialwert" unter "Spezialwerte in UltraLite" in der SQL Anywhere-Dokumentation.

Sie können den Standardwert von vorhandenen Tabellen ermitteln, indem Sie die column_default-Spalte der TableSchema.SYS_COLUMNS-Systemtabelle abfragen.

Siehe auch

- [TableSchema.SYS_COLUMNS-Variable \[UltraLiteJ\] auf Seite 282](#)

COLUMN_DEFAULT_GLOBAL_AUTOINC-Variable

Gibt das Spalten-Standardattribut GLOBAL AUTOINCREMENT an.

Syntax

```
final byte ColumnSchema.COLUMN_DEFAULT_GLOBAL_AUTOINC
```

Bemerkungen

Diese Konstante ähnelt dem AUTOINCREMENT-Attribut, aber die Domäne ist partitioniert. Jede Teilmenge enthält dieselbe Anzahl von Werten. Sie müssen jeder Kopie der Datenbank eine eindeutige

globale Datenbank-Identifizierungsnummer zuordnen. UltraLiteJ liefert Standardwerte in einer Datenbank von der Partition, die eindeutig durch diese Datenbanknummer gekennzeichnet ist.

Sie können den Standardwert von vorhandenen Tabellen ermitteln, indem Sie die `column_default`-Spalte der `TableSchema.SYS_COLUMNS`-Systemtabelle abfragen.

Siehe auch

- [TableSchema.SYS_COLUMNS-Variable \[UltraLiteJ\] auf Seite 282](#)
- [Connection.setDatabaseId-Methode \[UltraLiteJ\] auf Seite 120](#)

COLUMN_DEFAULT_NONE-Variable

Gibt an, dass die Spalte kein Standardattribut hat.

Syntax

```
final byte ColumnSchema.COLUMN_DEFAULT_NONE
```

Bemerkungen

Folgende Standardwerte gelten, wenn kein Standardattribut zugeordnet ist:

- Nullwertfähige Spalten werden standardmäßig auf NULL gesetzt.
- Nicht nullwertfähige numerische Spalten werden standardmäßig auf Null gesetzt.
- Nicht nullwertfähige Spalten mit variabler Länge werden standardmäßig auf Werte der Länge Null gesetzt.

Sie können den Standardwert von vorhandenen Tabellen ermitteln, indem Sie die `column_default`-Spalte der `TableSchema.SYS_COLUMNS`-Systemtabelle abfragen.

Siehe auch

- [TableSchema.SYS_COLUMNS-Variable \[UltraLiteJ\] auf Seite 282](#)

COLUMN_DEFAULT_UNIQUE_ID-Variable

Gibt das Spalten-Standardattribut `new_unique_identifizier` an.

Syntax

```
final byte ColumnSchema.COLUMN_DEFAULT_UNIQUE_ID
```

Bemerkungen

UUIDs können verwendet werden, um Zeilen in einer Tabelle eindeutig zu identifizieren. Die generierten Werte sind auf jedem Computer oder Gerät eindeutig, d.h. dass sie als Schlüssel in Synchronisations- und Replikationsumgebungen verwendet werden können.

Sie können den Standardwert von vorhandenen Tabellen ermitteln, indem Sie die `column_default`-Spalte der `TableSchema.SYS_COLUMNS`-Systemtabelle abfragen.

Siehe auch

- [TableSchema.SYS_COLUMNS-Variable \[UltraLiteJ\] auf Seite 282](#)

ConfigFile-Schnittstelle

Richtet ein Configuration-Objekt für eine beständige Datenbank ein, die in einer Datei gespeichert ist.

Syntax

```
public interface ConfigFile
```

Basisklassen

- [ConfigPersistent-Schnittstelle \[UltraLiteJ\] auf Seite 93](#)

Abgeleitete Klassen.

- [ConfigFileAndroid-Schnittstelle \[Android\] \[UltraLiteJ\] auf Seite 89](#)

Mitglieder

Alle Mitglieder der ConfigFile-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
enableAesDBEncryption-Methode	Aktiviert die AES-Verschlüsselung der Datenbank.
enableObfuscation-Methode	Aktiviert die Verschleierung der Datenbank.
getCacheSize-Methode	Gibt die Cachegröße der Datenbank (in Byte) zurück.
getConnectionString-Methode [Android]	Ruft die mit der setConnectionString-Methode registrierte Verbindungszeichenfolge ab.
getCreationString-Methode [Android]	Ruft die mit der setCreationString-Methode registrierte Erstellungszeichenfolge ab.
getDatabaseName-Methode	Gibt den Datenbanknamen zurück.
getEncryptionKey-Methode	Ruft den mit der setEncryptionKey-Methode registrierten Chiffrierschlüssel für die Datenbank ab.
getLazyLoadIndexes-Methode [BlackBerry]	Ermittelt, ob "lazy loading" (bei Bedarf geladene) Indizes aktiviert sind.
getPageSize-Methode	Gibt die Seitengröße der Datenbank (in Byte) zurück
getRowScoreFlushSize-Methode [BlackBerry]	Gibt die Bereinigungsgröße beim aktuellen Zeilen-Punktwert zurück.

Name	Beschreibung
getRowScoreMaximum-Methode [BlackBerry]	Gibt die maximale Größe beim aktuellen Zeilen-Punktwert zurück.
getUserName-Methode [Android]	Ruft den mit der setUsername-Methode festgelegten Benutzernamen ab.
hasShadowPaging-Methode [BlackBerry]	Ermittelt, ob Shadow Paging aktiviert ist.
setCacheSize-Methode	Setzt die Cachegröße der Datenbank (in Byte).
setConnectionString-Methode [Android]	Legt die Verbindungszeichenfolge fest, die zum Erstellen einer Datenbank oder zum Verbinden mit einer Datenbank verwendet werden soll.
setCreationString-Methode [Android]	Legt die Erstellungszeichenfolge fest, die zum Erstellen einer Datenbank verwendet werden soll.
setDatabaseName-Methode	Legt den Datenbanknamen fest.
setEncryptionKey-Methode	Legt den Schlüssel für die Verschlüsselung fest.
setLazyLoadIndexes-Methode [BlackBerry]	Legt fest, dass Indizes bei Bedarf (lazy load) oder dass alle Indizes auf einmal beim Start geladen werden.
setPageSize-Methode	Legt die Seitengröße der Datenbank fest.
setPassword-Methode	Setzt das Datenbankkennwort.
setRowScoreFlushSize-Methode [BlackBerry]	Aktiviert die Zeilenbegrenzung durch die Angabe des Punktwerts, der bei der Bereinigung zum Löschen alter Zeilen verwendet wird.
setRowScoreMaximum-Methode [BlackBerry]	Setzt den Schwellenwert für den maximalen Punktwert der Zeilen, die im Speicher gehalten werden sollen.
setUserName-Methode [Android]	Legt den Namen des Benutzers fest.

Bemerkungen

Ein Objekt zum Implementieren der ConfigFile-Schnittstelle wird mithilfe der `DatabaseManager.createConfigurationFile`-Methode erstellt.

Siehe auch

- [DatabaseManager-Klasse \[UltraLiteJ\] auf Seite 138](#)
- [DatabaseManager.createConfigurationFile-Methode \[UltraLiteJ\] auf Seite 140](#)

ConfigFileAndroid-Schnittstelle [Android]

Richtet ein Configuration-Objekt für eine beständige Datenbank ein, die in einer Datei auf einem Android-Gerät gespeichert ist.

Syntax

```
public interface ConfigFileAndroid
```

Basisklassen

- [ConfigFile-Schnittstelle \[UltraLiteJ\] auf Seite 87](#)

Mitglieder

Alle Mitglieder der ConfigFileAndroid-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
enableAesDBEncryption-Methode	Aktiviert die AES-Verschlüsselung der Datenbank.
enableObfuscation-Methode	Aktiviert die Verschleierung der Datenbank.
getCacheSize-Methode	Gibt die Cachegröße der Datenbank (in Byte) zurück.
getConnectionString-Methode [Android]	Ruft die mit der setConnectionString-Methode registrierte Verbindungszeichenfolge ab.
getCreationString-Methode [Android]	Ruft die mit der setCreationString-Methode registrierte Erstellungszeichenfolge ab.
getDatabaseName-Methode	Gibt den Datenbanknamen zurück.
getEncryptionKey-Methode	Ruft den mit der setEncryptionKey-Methode registrierten Chiffrierschlüssel für die Datenbank ab.
getLazyLoadIndexes-Methode [BlackBerry]	Ermittelt, ob "lazy loading" (bei Bedarf geladene) Indizes aktiviert sind.
getPageSize-Methode	Gibt die Seitengröße der Datenbank (in Byte) zurück
getRowScoreFlushSize-Methode [BlackBerry]	Gibt die Bereinigungsgröße beim aktuellen Zeilen-Punktwert zurück.
getRowScoreMaximum-Methode [BlackBerry]	Gibt die maximale Größe beim aktuellen Zeilen-Punktwert zurück.
getUserName-Methode [Android]	Ruft den mit der setUsername-Methode festgelegten Benutzernamen ab.

Name	Beschreibung
hasShadowPaging-Methode [BlackBerry]	Ermittelt, ob Shadow Paging aktiviert ist.
setCacheSize-Methode	Setzt die Cachegröße der Datenbank (in Byte).
setConnectionString-Methode [Android]	Legt die Verbindungszeichenfolge fest, die zum Erstellen einer Datenbank oder zum Verbinden mit einer Datenbank verwendet werden soll.
setCreationString-Methode [Android]	Legt die Erstellungszeichenfolge fest, die zum Erstellen einer Datenbank verwendet werden soll.
setDatabaseName-Methode	Legt den Datenbanknamen fest.
setEncryptionKey-Methode	Legt den Schlüssel für die Verschlüsselung fest.
setLazyLoadIndexes-Methode [BlackBerry]	Legt fest, dass Indizes bei Bedarf (lazy load) oder dass alle Indizes auf einmal beim Start geladen werden.
setPageSize-Methode	Legt die Seitengröße der Datenbank fest.
setPassword-Methode	Setzt das Datenbankkennwort.
setRowScoreFlushSize-Methode [BlackBerry]	Aktiviert die Zeilenbegrenzung durch die Angabe des Punktwerts, der bei der Bereinigung zum Löschen alter Zeilen verwendet wird.
setRowScoreMaximum-Methode [BlackBerry]	Setzt den Schwellenwert für den maximalen Punktwert der Zeilen, die im Speicher gehalten werden sollen.
setUserName-Methode [Android]	Legt den Namen des Benutzers fest.

Bemerkungen

Ein Objekt zum Implementieren der ConfigFileAndroid-Schnittstelle wird mithilfe der DatabaseManager.createConfigurationFileAndroid-Methode erstellt.

Siehe auch

- [DatabaseManager-Klasse \[UltraLiteJ\] auf Seite 138](#)
- [DatabaseManager.createConfigurationFileAndroid-Methode \[UltraLiteJ\] auf Seite 140](#)

ConfigNonPersistent-Schnittstelle [BlackBerry]

Richtet ein Configuration-Objekt für eine nicht-beständige (speicherresidente) Datenbank ein.

Syntax

```
public interface ConfigNonPersistent
```

Basisklassen

- [Configuration-Schnittstelle \[UltraLiteJ\] auf Seite 102](#)

Mitglieder

Alle Mitglieder der ConfigNonPersistent-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getDatabaseName-Methode	Gibt den Datenbanknamen zurück.
getPageSize-Methode	Gibt die Seitengröße der Datenbank (in Byte) zurück
setDatabaseName-Methode	Legt den Datenbanknamen fest.
setPageSize-Methode	Legt die Seitengröße der Datenbank fest.
setPassword-Methode	Setzt das Datenbankkennwort.

Bemerkungen

Ein Objekt zum Implementieren der ConfigNonPersistent-Schnittstelle wird mithilfe der `DatabaseManager.createConfigurationNonPersistent`-Methode erstellt.

Das Erstellen eines NonPersistent-Objekts konfiguriert einen Datenbankspeicher, der nur im Arbeitsspeicher existiert. Die Datenbank wird beim Starten erstellt und während der Ausführung der Anwendung verwendet, um dann beim Schließen der Anwendung verworfen zu werden. Wenn die Anwendung geschlossen wird, werden alle im nicht-beständigen Speicher enthaltenen Daten gelöscht.

Siehe auch

- [DatabaseManager-Klasse \[UltraLiteJ\] auf Seite 138](#)

ConfigObjectStore-Schnittstelle [BlackBerry]

Richtet ein Configuration-Objekt für eine beständige Datenbank ein, die in einem Objektspeicher gespeichert ist.

Syntax

```
public interface ConfigObjectStore
```

Basisklassen

- [ConfigPersistent-Schnittstelle \[UltraLiteJ\] auf Seite 93](#)

Mitglieder

Alle Mitglieder der ConfigObjectStore-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
enableAesDBEncryption-Methode	Aktiviert die AES-Verschlüsselung der Datenbank.
enableObfuscation-Methode	Aktiviert die Verschleierung der Datenbank.
getCacheSize-Methode	Gibt die Cachegröße der Datenbank (in Byte) zurück.
getConnectionString-Methode [Android]	Ruft die mit der setConnectionString-Methode registrierte Verbindungszeichenfolge ab.
getCreationString-Methode [Android]	Ruft die mit der setCreationString-Methode registrierte Erstellungszeichenfolge ab.
getDatabaseName-Methode	Gibt den Datenbanknamen zurück.
getEncryptionKey-Methode	Ruft den mit der setEncryptionKey-Methode registrierten Chiffrierschlüssel für die Datenbank ab.
getLazyLoadIndexes-Methode [BlackBerry]	Ermittelt, ob "lazy loading" (bei Bedarf geladene) Indizes aktiviert sind.
getPageSize-Methode	Gibt die Seitengröße der Datenbank (in Byte) zurück
getRowScoreFlushSize-Methode [BlackBerry]	Gibt die Bereinigungsgröße beim aktuellen Zeilen-Punktwert zurück.
getRowScoreMaximum-Methode [BlackBerry]	Gibt die maximale Größe beim aktuellen Zeilen-Punktwert zurück.
getUserName-Methode [Android]	Ruft den mit der setUsername-Methode festgelegten Benutzernamen ab.
hasShadowPaging-Methode [BlackBerry]	Ermittelt, ob Shadow Paging aktiviert ist.
setCacheSize-Methode	Setzt die Cachegröße der Datenbank (in Byte).
setConnectionString-Methode [Android]	Legt die Verbindungszeichenfolge fest, die zum Erstellen einer Datenbank oder zum Verbinden mit einer Datenbank verwendet werden soll.
setCreationString-Methode [Android]	Legt die Erstellungszeichenfolge fest, die zum Erstellen einer Datenbank verwendet werden soll.
setDatabaseName-Methode	Setzt den Datenbanknamen.

Name	Beschreibung
setEncryptionKey-Methode	Legt den Schlüssel für die Verschlüsselung fest.
setLazyLoadIndexes-Methode [BlackBerry]	Legt fest, dass Indizes bei Bedarf (lazy load) oder dass alle Indizes auf einmal beim Start geladen werden.
setPageSize-Methode	Legt die Seitengröße der Datenbank fest.
setPassword-Methode	Setzt das Datenbankkennwort.
setRowScoreFlushSize-Methode [BlackBerry]	Aktiviert die Zeilenbegrenzung durch die Angabe des Punktwerts, der bei der Bereinigung zum Löschen alter Zeilen verwendet wird.
setRowScoreMaximum-Methode [BlackBerry]	Setzt den Schwellenwert für den maximalen Punktwert der Zeilen, die im Speicher gehalten werden sollen.
setUserName-Methode [Android]	Legt den Namen des Benutzers fest.

Bemerkungen

Ein Objekt zum Implementieren der ConfigObjectStore-Schnittstelle wird mithilfe der `DatabaseManager.createConfigurationObjectStore-Methode` erstellt.

Siehe auch

- [DatabaseManager-Klasse \[UltraLiteJ\] auf Seite 138](#)
- [DatabaseManager.createConfigurationObjectStore-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 141](#)

ConfigPersistent-Schnittstelle

Richtet ein Configuration-Objekt für eine beständige Datenbank ein.

Syntax

```
public interface ConfigPersistent
```

Basisklassen

- [Configuration-Schnittstelle \[UltraLiteJ\] auf Seite 102](#)

Abgeleitete Klassen.

- [ConfigFile-Schnittstelle \[UltraLiteJ\] auf Seite 87](#)
- [ConfigObjectStore-Schnittstelle \[BlackBerry\] \[UltraLiteJ\] auf Seite 91](#)

Mitglieder

Alle Mitglieder der ConfigPersistent-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
enableAesDBEncryption-Methode	Aktiviert die AES-Verschlüsselung der Datenbank.
enableObfuscation-Methode	Aktiviert die Verschleierung der Datenbank.
getCacheSize-Methode	Gibt die Cachegröße der Datenbank (in Byte) zurück.
getConnectionString-Methode [Android]	Ruft die mit der setConnectionString-Methode registrierte Verbindungszeichenfolge ab.
getCreationString-Methode [Android]	Ruft die mit der setCreationString-Methode registrierte Erstellungszeichenfolge ab.
getDatabaseName-Methode	Gibt den Datenbanknamen zurück.
getEncryptionKey-Methode	Ruft den mit der setEncryptionKey-Methode registrierten Chiffrierschlüssel für die Datenbank ab.
getLazyLoadIndexes-Methode [BlackBerry]	Ermittelt, ob "lazy loading" (bei Bedarf geladene) Indizes aktiviert sind.
getPageSize-Methode	Gibt die Seitengröße der Datenbank (in Byte) zurück
getRowScoreFlushSize-Methode [BlackBerry]	Gibt die Bereinigungsgröße beim aktuellen Zeilen-Punktwert zurück.
getRowScoreMaximum-Methode [BlackBerry]	Gibt die maximale Größe beim aktuellen Zeilen-Punktwert zurück.
getUserName-Methode [Android]	Ruft den mit der setUsername-Methode festgelegten Benutzernamen ab.
hasShadowPaging-Methode [BlackBerry]	Ermittelt, ob Shadow Paging aktiviert ist.
setCacheSize-Methode	Setzt die Cachegröße der Datenbank (in Byte).
setConnectionString-Methode [Android]	Legt die Verbindungszeichenfolge fest, die zum Erstellen einer Datenbank oder zum Verbinden mit einer Datenbank verwendet werden soll.
setCreationString-Methode [Android]	Legt die Erstellungszeichenfolge fest, die zum Erstellen einer Datenbank verwendet werden soll.
setDatabaseName-Methode	Legt den Datenbanknamen fest.
setEncryptionKey-Methode	Legt den Schlüssel für die Verschlüsselung fest.

Name	Beschreibung
setLazyLoadIndexes-Methode [BlackBerry]	Legt fest, dass Indizes bei Bedarf (lazy load) oder dass alle Indizes auf einmal beim Start geladen werden.
setPageSize-Methode	Legt die Seitengröße der Datenbank fest.
setPassword-Methode	Setzt das Datenbankkennwort.
setRowScoreFlushSize-Methode [BlackBerry]	Aktiviert die Zeilenbegrenzung durch die Angabe des Punktwerts, der bei der Bereinigung zum Löschen alter Zeilen verwendet wird.
setRowScoreMaximum-Methode [BlackBerry]	Setzt den Schwellenwert für den maximalen Punktwert der Zeilen, die im Speicher gehalten werden sollen.
setUserName-Methode [Android]	Legt den Namen des Benutzers fest.

Bemerkungen

Standardmäßig verwendet eine Datenbank einen beständigen Speicher mit Shadow Paging, der aktualisiert wird, wenn Transaktionen festgeschrieben werden, oder wenn Indizes oder im Cache abgelegte Seiten aus dem Arbeitsspeicher ausgelagert werden.

Optionen wie etwa "lazy loading" (Laden bei Bedarf) sowie auf dem Zeilen-Punktwert basierende Bereinigungsgröße und maximale Größe gelten nur für beständige Datenbanken mit Shadow Paging.

enableAesDBEncryption-Methode

Aktiviert die AES-Verschlüsselung der Datenbank.

Syntax

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

Bemerkungen

Geben Sie den DBKEY-Verbindungsparameter bei der Erstellung oder beim Herstellen einer Verbindung mit der Datenbank an oder verwenden Sie die setEncryptionKey-Methode.

Siehe auch

- [ConfigPersistent.setEncryptionKey-Methode \[UltraLiteJ\] auf Seite 100](#)
- „UltraLite-Verbindungsparameter DBKEY“ [*UltraLite - Datenbankverwaltung*]

enableObfuscation-Methode

Aktiviert die Verschleierung der Datenbank.

Syntax

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

getCacheSize-Methode

Gibt die Cachegröße der Datenbank (in Byte) zurück.

Syntax

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

Rückgabe

Die Cachegröße.

Siehe auch

- [ConfigPersistent.setCacheSize-Methode \[UltraLiteJ\] auf Seite 98](#)

getConnectionString-Methode [Android]

Ruft die mit der setConnectionString-Methode registrierte Verbindungszeichenfolge ab.

Syntax

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

Rückgabe

Die mit der setConnectionString-Methode registrierte Verbindungszeichenfolge.

Siehe auch

- [ConfigPersistent.setConnectionString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)

getCreationString-Methode [Android]

Ruft die mit der setCreationString-Methode registrierte Erstellungszeichenfolge ab.

Syntax

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

Rückgabe

Die mit der setCreationString-Methode registrierte Erstellungszeichenfolge.

Siehe auch

- [ConfigPersistent.setCreationString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)

getEncryptionKey-Methode

Ruft den mit der setEncryptionKey-Methode registrierten Chiffrierschlüssel für die Datenbank ab.

Syntax

```
String ConfigPersistent.getEncryptionKey( )
```

Rückgabe

Der mit der setEncryptionKey-Methode registrierte Chiffrierschlüssel für die Datenbank.

Siehe auch

- [ConfigPersistent.setEncryptionKey-Methode \[UltraLiteJ\] auf Seite 100](#)
- [Connection.changeEncryptionKey-Methode \[UltraLiteJ\] auf Seite 110](#)

getLazyLoadIndexes-Methode [BlackBerry]

Ermittelt, ob "lazy loading" (bei Bedarf geladene) Indizes aktiviert sind.

Syntax

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

Rückgabe

TRUE, wenn "lazy loading" aktiviert ist, ansonsten FALSE.

Siehe auch

- [ConfigPersistent.setLazyLoadIndexes-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 100](#)

getRowScoreFlushSize-Methode [BlackBerry]

Gibt die Bereinigungsgröße beim aktuellen Zeilen-Punktwert zurück.

Syntax

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

Rückgabe

Die Bereinigungsgröße der aktuellen Zeile auf der Basis ihres Punktwerts.

Siehe auch

- [ConfigPersistent.setRowScoreFlushSize-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 101](#)

getRowScoreMaximum-Methode [BlackBerry]

Gibt die maximale Größe beim aktuellen Zeilen-Punktwert zurück.

Syntax

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

Rückgabe

Die maximale Größe der aktuellen Zeile auf der Basis ihres Punktwerts.

Siehe auch

- [ConfigPersistent.setRowScoreMaximum-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 101](#)

getUserName-Methode [Android]

Ruft den mit der setUsername-Methode festgelegten Benutzernamen ab.

Syntax

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

Rückgabe

Der mit der setUsername-Methode festgelegte Benutzername.

Siehe auch

- [ConfigPersistent.setUsername-Methode \[Android\] \[UltraLiteJ\] auf Seite 102](#)

hasShadowPaging-Methode [BlackBerry]

Ermittelt, ob Shadow Paging aktiviert ist.

Syntax

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

Rückgabe

TRUE, wenn Shadow Paging aktiviert ist, ansonsten FALSE.

setCacheSize-Methode

Setzt die Cachegröße der Datenbank (in Byte).

Syntax

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

Parameter

- **cache_size** Die Cachegröße. Der Standardwert für die Cachegröße beträgt 20480 (20 kB) auf allen Plattformen.

Rückgabe

Dieses ConfigPersistent-Objekt mit der angegebenen Cachegröße.

Bemerkungen

Die Cachegröße legt die Anzahl der Datenbankseiten fest, die im Seitencache aufbewahrt werden. Ein Erhöhen der Größe bewirkt, dass weniger Datenbankseiten gelesen und geschrieben werden, allerdings erhöht sich der Zeitbedarf, um die Position der Seiten im Cache zu ermitteln.

Siehe auch

- [ConfigPersistent.getCacheSize-Methode \[UltraLiteJ\] auf Seite 96](#)

setConnectionString-Methode [Android]

Legt die Verbindungszeichenfolge fest, die zum Erstellen einer Datenbank oder zum Verbinden mit einer Datenbank verwendet werden soll.

Syntax

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

Parameter

- **connection_string** Die Verbindungszeichenfolge, die bei der Datenbankverbindung oder -erstellung verwendet wird.

Bemerkungen

Andere in dieser Konfiguration festgelegte Elemente werden ebenfalls zum Erstellen oder Verbinden mit einer Datenbank übergeben.

setCreationString-Methode [Android]

Legt die Erstellungszeichenfolge fest, die zum Erstellen einer Datenbank verwendet werden soll.

Syntax

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

Parameter

- **creation_string** Die Erstellungszeichenfolge, die bei der Datenbankerstellung verwendet wird.

Bemerkungen

Andere in dieser Konfiguration festgelegte Elemente werden ebenfalls zum Erstellen einer Datenbank übergeben.

setEncryptionKey-Methode

Legt den Schlüssel für die Verschlüsselung fest.

Syntax

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

Parameter

- **encryption_key** Die Zeichenfolge, die für den Chiffrierschlüssel verwendet werden soll.

Siehe auch

- [ConfigPersistent.getEncryptionKey-Methode \[UltraLiteJ\] auf Seite 97](#)
- [ConfigPersistent.enableAesDBEncryption-Methode \[UltraLiteJ\] auf Seite 95](#)
- [Connection.changeEncryptionKey-Methode \[UltraLiteJ\] auf Seite 110](#)

setLazyLoadIndexes-Methode [BlackBerry]

Legt fest, dass Indizes bei Bedarf (lazy load) oder dass alle Indizes auf einmal beim Start geladen werden.

Syntax

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

Parameter

- **lazy_load** Auf TRUE setzen, um Indizes bei Bedarf zu laden, ansonsten auf FALSE, um alle Indizes auf einmal beim Start zu laden.

Rückgabe

Dieses ConfigPersistent-Objekt mit dem angegebenen Indexladeschema.

Bemerkungen

Ein Aktivieren dieser Option verkürzt die Startzeit der Datenbank, aber spätere Vorgänge werden möglicherweise langsamer ausgeführt.

Die Deaktivierung von "lazy loading" deaktiviert auch die Zeilenbegrenzung durch Einstellung der auf dem Zeilen-Punktwert basierenden Bereinigungsgröße auf Null.

Siehe auch

- [ConfigPersistent.getLazyLoadIndexes-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 97](#)
- [ConfigPersistent.setRowScoreFlushSize-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 101](#)

setRowScoreFlushSize-Methode [BlackBerry]

Aktiviert die Zeilenbegrenzung durch die Angabe des Punktwerts, der bei der Bereinigung zum Löschen alter Zeilen verwendet wird.

Syntax

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

Parameter

- **flushSize** Der Zeilen-Punktwert, mit dem ermittelt wird, wie viele Zeilen bei der Bereinigung gelöscht werden. Der Standardwert ist 0, d.h. keine Zeilenbegrenzung.

Rückgabe

Dieses ConfigPersistent-Objekt mit dem angegebenen Bereinigungsgrößenwert.

Bemerkungen

Der Zeilen-Punktwert ist ein Maß für die Referenzen, die dazu verwendet werden, zuletzt verwendete Zeilen im Speicher zu halten. Jeder Zeile im Speicher wird ein Punktwert zugeordnet, basierend auf Anzahl und Typen von enthaltenen Spalten. Dieser Wert entspricht ungefähr der maximalen Anzahl der Referenzen, die verwendet werden können.

Die meisten Spalten werden mit 1 bewertet, VARCHAR BINARY, LONG BINARY und UUID mit 2 Punkten und LONG VARCHAR mit 4 Punkten.

Wenn der maximale Punktwert erreicht ist, wird mithilfe der Bereinigungsgröße ermittelt, wie viele alte Zeilen entfernt werden sollen.

Es wird empfohlen, die Bereinigungsgröße (gemessen als Zeilen-Punktwert) in einem angemessenen Rahmen zu halten (weniger als 1000), um lange Unterbrechungen zu verhindern.

Bei Datenbanken, auf die mit aktivierter Zeilenbegrenzung zugegriffen wird, ist immer auch "lazy loading" von Indizes aktiviert. Die Deaktivierung von "lazy loading" deaktiviert auch die Zeilenbegrenzung durch Einstellung der auf dem Zeilen-Punktwert basierenden Bereinigungsgröße auf Null.

Siehe auch

- [ConfigPersistent.setRowScoreMaximum-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 101](#)
- [ConfigPersistent.setLazyLoadIndexes-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 100](#)

setRowScoreMaximum-Methode [BlackBerry]

Setzt den Schwellenwert für den maximalen Punktwert der Zeilen, die im Speicher gehalten werden sollen.

Syntax

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

Parameter

- **threshold** Der maximale Wert für den Schwellenwert. Der Höchstwert ist 200.000. Der Standardwert ist 50.000.

Rückgabe

Dieses ConfigPersistent-Objekt mit dem angegebenen maximalen Schwellenwert.

Bemerkungen

Bei Datenbanken, auf die mit aktivierter Zeilenbegrenzung zugegriffen wird, ist immer auch "lazy loading" von Indizes aktiviert.

Siehe auch

- [ConfigPersistent.setRowScoreFlushSize-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 101](#)
- [ConfigPersistent.setLazyLoadIndexes-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 100](#)

setUserName-Methode [Android]

Legt den Namen des Benutzers fest.

Syntax

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

Parameter

- **user_name** Der Name des Benutzers.

Bemerkungen

Dieser Name wird verwendet, um mithilfe der UID=Phrase in der Verbindungszeichenfolge eine Verbindung mit der nativen Datenbank herzustellen oder diese zu erstellen.

Configuration-Schnittstelle

Richtet ein Configuration-Objekt für eine Datenbank ein.

Syntax

```
public interface Configuration
```

Abgeleitete Klassen

- [ConfigNonPersistent-Schnittstelle \[BlackBerry\] \[UltraLiteJ\] auf Seite 90](#)
- [ConfigPersistent-Schnittstelle \[UltraLiteJ\] auf Seite 93](#)

Mitglieder

Alle Mitglieder der Configuration-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getDatabaseName-Methode	Gibt den Datenbanknamen zurück.
getPageSize-Methode	Gibt die Seitengröße der Datenbank (in Byte) zurück
setDatabaseName-Methode	Legt den Datenbanknamen fest.
setPageSize-Methode	Legt die Seitengröße der Datenbank fest.
setPassword-Methode	Setzt das Datenbankkennwort.

Bemerkungen

Manche Attribute werden nur während der Datenbankerstellung verwendet, während andere bei der anfänglichen Verbindungsherstellung zu einer Datenbank gelten. Attribute werden ignoriert, wenn sie nach der Erstellung einer Datenbank oder der Verbindungsherstellung zu einer Datenbank festgelegt werden.

getDatabaseName-Methode

Gibt den Datenbanknamen zurück.

Syntax

```
String Configuration.getDatabaseName()
```

Rückgabe

Der Name der Datenbank.

getPageSize-Methode

Gibt die Seitengröße der Datenbank (in Byte) zurück

Syntax

```
int Configuration.getPageSize()
```

Rückgabe

Die Seitengröße.

setDatabaseName-Methode

Legt den Datenbanknamen fest.

Syntax

```
Configuration Configuration.setDatabaseName(  
    String db_name  
) throws ULjException
```

Parameter

- **db_name** Der Name der Datenbank.

Rückgabe

Dieses Configuration-Objekt mit dem angegebenen Datenbanknamen.

setPageSize-Methode

Legt die Seitengröße der Datenbank fest.

Syntax

```
Configuration Configuration.setPageSize(  
    int page_size  
) throws ULjException
```

Parameter

- **page_size** Die Seitengröße in Byte.

Rückgabe

Dieses Configuration-Objekt mit der angegebenen Seitengröße.

Bemerkungen

Die Seitengrößeneinstellung wird verwendet, um die maximale Größe einer Zeile zu bestimmen, die in einer beständigen Datenbank gespeichert wird. Sie legt die Größe einer Indexseite fest und bestimmt die Anzahl von untergeordneten Elementen, die jede Seite haben kann.

Wenn eine vorhandene Datenbank verwendet wird, ist die Größe bereits auf die Seitengröße der Datenbank zum Zeitpunkt ihrer Erstellung gesetzt. Sie können die Seitengröße einer vorhandenen Datenbank mit dieser Methode nicht neu festlegen.

Bei Android-Smartphones kann die Seitengröße 1024, 2048, 4096, 8192 oder 16384 Byte betragen. Der Standardwert ist 4096 Byte.

Bei BlackBerry-Smartphones kann die Seitengröße von 256 bis 16384 Byte reichen. Der Standardwert ist 1024 Byte. Die Seitengröße wird immer so angepasst, dass sie ein Vielfaches von 32 ist.

setPassword-Methode

Setzt das Datenbankkennwort.

Syntax

```
Configuration Configuration.setPassword(  
    String password  
) throws ULjException
```

Parameter

- **password** Ein Kennwort für eine neue Datenbank oder das Kennwort, um Zugriff auf eine vorhandene Datenbank zu erhalten.

Rückgabe

Dieses Configuration-Objekt mit dem festgelegten Datenbankkennwort.

Bemerkungen

Das Kennwort wird für den Zugriff auf die Datenbank verwendet und muss mit dem Kennwort übereinstimmen, das angegeben wurde, als die Datenbank erstellt wurde. Der Standardwert ist "dba".

Connection-Schnittstelle

Beschreibt eine Datenbankverbindung, die erforderlich ist, um Datenbankvorgänge zu initiieren.

Syntax

```
public interface Connection
```

Mitglieder

Alle Mitglieder der Connection-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
cancelWaitForEvent-Methode [Android]	Hebt alle waitForEvent-Aufrufe für dieses Connection-Objekt auf.
changeEncryptionKey-Methode	Ändert den Chiffrierschlüssel für die Verschlüsselung einer UltraLite-Datenbank.
commit-Methode	Schreibt die Datenbankänderungen fest.
createDecimalNumber-Methode	Erstellt ein neues DecimalNumber-Objekt.
createSyncParms-Methode	Erstellt eine Menge von Synchronisationsparametern.
createUUIDValue-Methode	Erstellt einen UUID-Wert.
dropDatabase-Methode	Löscht eine Datenbank.

Name	Beschreibung
emergencyShutdown-Methode [BlackBerry]	Führt ein Notfall-Herunterfahren der verbundenen Datenbank durch.
getDatabaseId-Methode [BlackBerry]	Gibt den Wert der Datenbank-ID zurück.
getDatabaseInfo-Methode	Gibt ein DataInfo-Objekt zurück, das Informationen zu Datenbankeigenschaften enthält.
getDatabaseProperty-Methode	Gibt eine Datenbankeigenschaft zurück.
getLastDownloadTime-Method	Gibt die Uhrzeit des letzten Downloads der angegebenen Publikation zurück.
getLastIdentity-Methode	Ruft den Wert ab, der zuletzt in eine DEFAULT AUTOINCREMENT- oder DEFAULT GLOBAL AUTOINCREMENT-Spalte eingefügt wurde, oder Null, wenn die bei der letzten INSERT-Transaktion verwendete Tabelle keine solche Spalte hatte.
getLastWarning-Methode	Gibt Informationen über die letzte SQL-Anweisung zurück, die auf dieser Verbindung ausgeführt wurde.
getOption-Methode [BlackBerry]	Gibt eine Datenbankoption zurück.
getState-Methode	Gibt den Status der Verbindung zurück.
getSyncObserver-Methode	Gibt das SyncObserver-Objekt zurück, das derzeit für dieses Connection-Objekt registriert ist.
getSyncResult-Methode	Gibt das Ergebnis der letzten SYNCHRONIZE SQL-Anweisung zurück.
isSynchronizationDeleteDisabled-Methode [BlackBerry]	Ermittelt, ob die Synchronisation von Löschvorgängen deaktiviert ist.
prepareStatement-Methode	Bereitet eine Anweisung für die Ausführung vor.
registerForEvent-Methode [Android]	Registriert ein Systemereignis, um Benachrichtigungen zu erhalten.
release-Methode	Gibt diese Verbindung frei.
resetLastDownloadTime-Methode	Setzt die Zeit des Downloads bei den angegebenen Publikationen zurück.

Name	Beschreibung
rollback-Methode	Schreibt ein Rollback fest, um Änderungen in der Datenbank rückgängig zu machen.
rollbackPartialDownload-Methode [Android]	Setzt die Änderungen einer fehlgeschlagenen Synchronisation zurück.
setDatabaseId-Methode	Legt die Datenbank-ID fest, die für global auto-increment-Spalten verwendet wird.
setOption-Methode	Setzt die Datenbankoption.
setSyncObserver-Methode	Setzt ein SyncObserver-Objekt, das den Fortschritt der Synchronisationen für diese Verbindung überwacht.
synchronize-Methode	Synchronisiert die Datenbank mit einem Mobi-Link-Server.
unregisterForEvent-Methode [Android]	Hebt die Registrierung eines Systemereignisses auf, um den Empfang von Benachrichtigungen zu stoppen.
validateDatabase-Methode [Android]	Validiert die Datenbank in dieser Verbindung.
waitForEvent-Methode [Android]	Wartet auf eine Ereignisbenachrichtigung.
CONNECTED-Variable	Bezeichnet einen verbundenen Zustand.
NOT_CONNECTED-Variable	Bezeichnet einen nicht verbundenen Zustand.
OPTION_BLOB_FILE_BASE_DIR-Variable [BlackBerry]	Datenbankoption: Basisverzeichnis der Blob-Datei.
OPTION_DATABASE_ID-Variable [BlackBerry]	Datenbankoption: Datenbank-ID.
OPTION_DATE_FORMAT-Variable	Datenbankoption: Datumsformat.
OPTION_DATE_ORDER-Variable	Datenbankoption: Datumsreihenfolge.
OPTION_MAX_HASH_SIZE-Variable	Datenbankoption: maximale Hash-Größe.
OPTION_ML_REMOTE_ID-Variable [BlackBerry]	Datenbankoption: Entfernte ML-ID.
OPTION_ML_SERVER_VERSION-Variable [BlackBerry]	Datenbankoption: Protokollversion des Mobi-Link-Servers.

Name	Beschreibung
OPTION_NEAREST_CENTURY-Variable	Datenbankoption: Nächstliegendes Jahrhundert.
OPTION_PRECISION-Variable	Datenbankoption: Gesamtstellenzahl.
OPTION_SCALE-Variable	Datenbankoption: Dezimalstellen.
OPTION_TIME_FORMAT-Variable	Datenbankoption: Zeitformat.
OPTION_TIMESTAMP_FORMAT-Variable	Datenbankoption: Zeitstempelformat.
OPTION_TIMESTAMP_INCREMENT-Variable	Datenbankoption: Zeitstempel-Inkrement.
OPTION_TIME-STAMP_WITH_TIME_ZONE_FORMAT-Variable	Datenbankoption: Zeitstempel mit Zeitzoneformat.
PROPERTY_DATABASE_NAME-Variable	Datenbankeigenschaft: Datenbankname.
PROPERTY_PAGE_SIZE-Variable	Datenbankeigenschaft: Seitengröße.
SYNC_ALL-Variable	Die Publikationsliste, mit der die Synchronisation aller Tabellen in der Datenbank angefordert wird, einschließlich der Tabellen, die in keiner Publikation verwendet werden.
SYNC_ALL_DB_PUB_NAME -Variable	Der reservierte Name für die SYNC_ALL_PUB-Publikation.
SYNC_ALL_PUBS-Variable	Die Publikationsliste, mit der die Synchronisation aller Publikationen in der Datenbank angefordert wird.
ULVF_DATABASE-Variable [Android]	Zur Validierung der Datenbank.
ULVF_EXPRESS-Variable [Android]	Für eine schnellere, allerdings weniger präzise Validierung.
ULVF_FULL_VALIDATE-Variable [Android]	Führt alle Arten von Validierungen der Datenbank durch.
ULVF_INDEX-Variable [Android]	Zur Validierung von Indizes.
ULVF_TABLE-Variable [Android]	Zum Validieren von Tabellen.

Bemerkungen

Eine Verbindung wird unter Verwendung der connect- oder createDatabase-Methoden der DatabaseManager-Klasse erhalten. Verwenden Sie die release-Methode, wenn die Verbindung nicht mehr

benötigt wird. Wenn alle Verbindungen für eine Datenbank freigegeben sind, wird die Datenbank geschlossen.

Ein Connection-Objekt stellt die folgenden Funktionalitäten bereit:

- Neues Schema erstellen (Tabellen, Indizes und Publikationen)
- Neue Wert- und Domänenobjekte erstellen
- Änderungen in der Datenbank dauerhaft festschreiben
- SQL-Anweisungen für die Ausführung vorbereiten
- Nicht festgeschriebene Änderungen in der Datenbank zurücksetzen

Das folgende Beispiel zeigt, wie Sie ein Schema für eine einfache Datenbank erstellen, für die ein Connection-Objekt, `conn`, erstellt wurde. Die Datenbank enthält eine Tabelle namens `T1`, die eine einzige Ganzzahl-Primärschlüsselspalte namens `"num"` hat, und eine Tabelle namens `T2`, die eine Ganzzahl-Primärschlüsselspalte namens `"num"` und eine Ganzzahlspalte namens `"quantity"` hat. `T2` hat einen Additionsindex auf `"quantity"`. Eine Publikation namens `PubA` enthält `T1`.

```
// 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();
```

Siehe auch

- [DatabaseManager-Klasse \[UltraLiteJ\] auf Seite 138](#)
- [DatabaseManager.createDatabase-Methode \[UltraLiteJ\] auf Seite 142](#)
- [DatabaseManager.connect-Methode \[UltraLiteJ\] auf Seite 139](#)
- [Connection.release-Methode \[UltraLiteJ\] auf Seite 119](#)

cancelWaitForEvent-Methode [Android]

Hebt alle `waitForEvent`-Aufrufe für dieses Connection-Objekt auf.

Syntax

```
void Connection.cancelWaitForEvent() throws ULjException
```

Siehe auch

- [Connection.waitForEvent-Methode \[Android\] \[UltraLiteJ\] auf Seite 124](#)

changeEncryptionKey-Methode

Ändert den Chiffrierschlüssel für die Verschlüsselung einer UltraLite-Datenbank.

Syntax

```
void Connection.changeEncryptionKey(String newKey) throws ULjException
```

Parameter

- **newKey** Der neue Chiffrierschlüssel für die Datenbank.

Bemerkungen

Anwendungen, die diese Methode aufrufen, müssen zunächst sicherstellen, dass der Benutzer entweder die Datenbank synchronisiert oder eine zuverlässige Sicherungskopie der Datenbank erstellt hat. Die zuverlässige Sicherung der Datenbank ist wichtig, da die changeEncryptionKey-Methode vollständig ausgeführt werden muss. Wenn Sie den Datenbank-Chiffrierschlüssel ändern, wird jede Zeile in der Datenbank zuerst mit dem alten Schlüssel entschlüsselt und dann mit dem neueren Schlüssel verschlüsselt, bevor die Zeile neu geschrieben wird. Dieser Vorgang kann nicht rückgängig gemacht werden. Wenn der Änderungsvorgang der Verschlüsselung nicht abgeschlossen wird, hat die Datenbank einen ungültigen Status und Sie können nicht mehr auf sie zugreifen.

commit-Methode

Schreibt die Datenbankänderungen fest.

Syntax

```
void Connection.commit() throws ULjException
```

Bemerkungen

Ein Aufrufen dieser Methode bewirkt, dass alle Änderungen der Tabellendaten seit dem letzten Festschreiben oder Zurücksetzen dauerhaft werden.

createDecimalNumber-Methode

Erstellt ein neues DecimalNumber-Objekt.

Überladungsliste

Name	Beschreibung
createDecimalNumber(int, int)-Methode	Erstellt ein DecimalNumber-Objekt.
createDecimalNumber(int, int, String)-Methode	Erstellt ein DecimalNumber-Objekt.

createDecimalNumber(int, int)-Methode

Erstellt ein DecimalNumber-Objekt.

Syntax

```
DecimalNumber Connection.createDecimalNumber(  
    int precision,  
    int scale  
) throws ULjException
```

Parameter

- **precision** Die Gesamtstellenanzahl der Zahl.
- **scale** Die Anzahl der Dezimalstellen der Zahl.

Rückgabe

Das DecimalNumber-Objekt mit dem angegebenen Typ.

Siehe auch

- [DecimalNumber-Schnittstelle \[UltraLiteJ\] auf Seite 146](#)

createDecimalNumber(int, int, String)-Methode

Erstellt ein DecimalNumber-Objekt.

Syntax

```
DecimalNumber Connection.createDecimalNumber(  
    int precision,  
    int scale,  
    String value  
) throws ULjException
```

Parameter

- **precision** Die Gesamtstellenanzahl der Zahl.
- **scale** Die Anzahl der Dezimalstellen der Zahl.
- **value** Der zu setzende Wert.

Rückgabe

Das DecimalNumber-Objekt mit dem angegebenen Typ.

Siehe auch

- [DecimalNumber-Schnittstelle \[UltraLiteJ\] auf Seite 146](#)

createSyncParms-Methode

Erstellt eine Menge von Synchronisationsparametern.

Überladungsliste

Name	Beschreibung
createSyncParms(int, String, String)-Methode	Erstellt eine Menge von Synchronisationsparametern für die HTTP-Synchronisation.
createSyncParms(String, String)-Methode	Erstellt eine Menge von Synchronisationsparametern für die HTTP-Synchronisation.

createSyncParms(int, String, String)-Methode

Erstellt eine Menge von Synchronisationsparametern für die HTTP-Synchronisation.

Syntax

```
SyncParms Connection.createSyncParms(  
    int streamType,  
    String userName,  
    String version  
) throws ULjException
```

Parameter

- **streamType** Eine der in der SyncParms-Klasse definierten Konstanten, die zur Kennzeichnung des Typs des Synchronisationsdatenstroms verwendet werden
- **userName** Der MobiLink-Benutzername
- **version** Die MobiLink-Skriptversion.

Rückgabe

Ein SyncParms-Objekt.

Siehe auch

- [Connection.createSyncParms-Methode \[UltraLiteJ\] auf Seite 112](#)
- [SyncParms.HTTP_STREAM-Variable \[UltraLiteJ\] auf Seite 268](#)
- [SyncParms.HTTPS_STREAM-Variable \[UltraLiteJ\] auf Seite 268](#)

createSyncParms(String, String)-Methode

Erstellt eine Menge von Synchronisationsparametern für die HTTP-Synchronisation.

Syntax

```
SyncParms Connection.createSyncParms(  
    String userName,  
    String version  
) throws ULjException
```

Parameter

- **userName** Der eindeutige MobiLink-Benutzername für diese Client-Datenbank.
- **version** Die MobiLink-Skriptversion.

Rückgabe

Das SyncParms-Objekt.

Siehe auch

- [Connection.createSyncParms-Methode \[UltraLiteJ\] auf Seite 112](#)
- [SyncParms.setUserName-Methode \[UltraLiteJ\] auf Seite 267](#)

createUUIDValue-Methode

Erstellt einen UUID-Wert.

Syntax

```
UUIDValue Connection.createUUIDValue() throws ULjException
```

Rückgabe

Die UUIDValue-Instanz für die Domäne.

dropDatabase-Methode

Löscht eine Datenbank.

Syntax

```
void Connection.dropDatabase() throws ULjException
```

Bemerkungen

Die von der Verbindung referenzierte Datenbank wird gelöscht und die Verbindung freigegeben. Diese Verbindung ist die einzige, die für die zu löschende Datenbank aktiv sein darf.

emergencyShutdown-Methode [BlackBerry]

Führt ein Notfall-Herunterfahren der verbundenen Datenbank durch.

Syntax

```
void Connection.emergencyShutdown() throws ULjException
```

Bemerkungen

Diese Methode sollte nur bei schwerwiegenden Fehlersituationen aufgerufen werden. Sie sollte nur verwendet werden, wenn Hardware oder Daten zerstört wurden.

Diese Methode schließt alle offenen Verbindungen und fährt die verbundene Datenbank herunter.

getDatabaseId-Methode [BlackBerry]

Gibt den Wert der Datenbank-ID zurück.

Syntax

```
int Connection.getDatabaseId() throws ULjException
```

Rückgabe

Die Datenbank-ID.

Ausnahmen

- [ULjException-Klasse](#) , wenn die Datenbank-ID nicht festgelegt wurde.

Siehe auch

- [Connection.getLastIdentity-Methode \[UltraLiteJ\] auf Seite 115](#)

getDatabaseInfo-Methode

Gibt ein DataInfo-Objekt zurück, das Informationen zu Datenbankeigenschaften enthält.

Syntax

```
DatabaseInfo Connection.getDatabaseInfo() throws ULjException
```

Rückgabe

Das DatabaseInfo-Objekt.

getDatabaseProperty-Methode

Gibt eine Datenbankeigenschaft zurück.

Syntax

`String Connection.getDatabaseProperty(String name)` throws **ULjException**

Parameter

- **name** Der Name der Datenbankeigenschaft. Bei Android-Geräten können Sie diesen Parameter auf den Namen einer beliebigen unterstützten UltraLite-Datenbankeigenschaft setzen. Bei BlackBerry-Geräten können Sie diesen Parameter auf eine beliebige Konstante in der Connection-Schnittstelle setzen, die das Präfix **PROPERTY_** hat.

Rückgabe

Der Wert der Eigenschaft, die dem angegebenen Namen entspricht.

Siehe auch

- [Connection.PROPERTY_DATABASE_NAME-Variable \[UltraLiteJ\] auf Seite 130](#)
- [Connection.PROPERTY_PAGE_SIZE-Variable \[UltraLiteJ\] auf Seite 131](#)
- „UltraLite-Datenbankeigenschaften“ [*UltraLite - Datenbankverwaltung*]

getLastDownloadTime-Method

Gibt die Uhrzeit des letzten Downloads der angegebenen Publikation zurück.

Syntax

`Date Connection.getLastDownloadTime(String pub_name)` throws **ULjException**

Parameter

- **pub_name** Der Name der zu überprüfenden Publikation. Der pub_name-Parameter muss eine einzelne Publikation referenzieren oder die Spezialpublikation `Connection.SYNC_ALL_DB_PUB_NAME` zum Zeitpunkt des letzten Downloads der gesamten Datenbank sein.

Rückgabe

Der Zeitstempel des letzten Downloads.

Siehe auch

- [Connection.SYNC_ALL_DB_PUB_NAME-Variable \[UltraLiteJ\] auf Seite 131](#)
- [Connection.resetLastDownloadTime-Methode \[UltraLiteJ\] auf Seite 119](#)

getLastIdentity-Methode

Ruft den Wert ab, der zuletzt in eine DEFAULT AUTOINCREMENT- oder DEFAULT GLOBAL AUTOINCREMENT-Spalte eingefügt wurde, oder Null, wenn die bei der letzten INSERT-Transaktion verwendete Tabelle keine solche Spalte hatte.

Syntax

```
long Connection.getLastIdentity()
```

Rückgabe

Der zuletzt verwendete Identity-Wert.

Bemerkungen

Wenn eine Tabelle mehr als eine Spalte vom Typ (GLOBAL) AUTOINCREMENT enthält, ist die Spalte, zu der dieser Wert gehört, unbestimmt.

getLastWarning-Methode

Gibt Informationen über die letzte SQL-Anweisung zurück, die auf dieser Verbindung ausgeführt wurde.

Syntax

```
SQLInfo Connection.getLastWarning()
```

Rückgabe

Das SQLInfo-Objekt für die letzte ausgeführte SQL-Anweisung.

getOption-Methode [BlackBerry]

Gibt eine Datenbankoption zurück.

Syntax

```
String Connection.getOption(String option_name) throws ULjException
```

Parameter

- **option_name** Der Name der abzurufenden Option.
- **option_name** Der Name der abzurufenden Option. Sie können diesen Parameter auf eine beliebige Konstante in der Connection-Schnittstelle setzen, die das Präfix **OPTION_** hat.

Rückgabe

Der Wert der Datenbankoption.

Bemerkungen

Datenbankoptionen werden in der Datenbank gespeichert und können, wenn eine Datenbank verbunden ist, zu einem späteren Zeitpunkt abgerufen werden, nachdem die Option eingestellt wurde.

Eine Gruppe von erforderlichen Optionen wird bei der Erstellung der Datenbank mit erstellt.

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)

getState-Methode

Gibt den Status der Verbindung zurück.

Syntax

```
byte Connection.getState() throws ULjException
```

Rückgabe

Das Byte, das den Status der Verbindung darstellt.

Bemerkungen

Das folgende Beispiel zeigt, wie Sie den Verbindungsstatus prüfen und die Verbindung freigeben.

```
if( _conn.getState() == Connection.CONNECTED ){  
    _conn.release();  
}
```

Siehe auch

- [Connection-Schnittstelle \[UltraLiteJ\] auf Seite 105](#)

getSyncObserver-Methode

Gibt das SyncObserver-Objekt zurück, das derzeit für dieses Connection-Objekt registriert ist.

Syntax

```
SyncObserver Connection.getSyncObserver( )
```

Rückgabe

Das SyncObserver-Objekt oder NULL, wenn kein Observer existiert.

Siehe auch

- [Connection.setSyncObserver-Methode \[UltraLiteJ\] auf Seite 122](#)

getSyncResult-Methode

Gibt das Ergebnis der letzten SYNCHRONIZE SQL-Anweisung zurück.

Syntax

```
SyncResult Connection.getSyncResult( )
```

Rückgabe

Das SyncResult-Objekt, das das Ergebnis der letzten SYNCHRONIZE SQL-Anweisung darstellt.

Bemerkungen

Das folgende Beispiel zeigt, wie Sie das Ergebnis der letzten SYNCHRONIZE SQL-Anweisung abrufen:

```
PreparedStatement ps = conn.prepareStatement("SYNCHRONIZE PROFILE
myprofile");
ps.execute();
ps.close();
SyncResult result = conn.getSyncResult();
display(
    "*** Synchronized *** sent=" + result.getSentRowCount()
    + ", received=" + result.getReceivedRowCount()
);
```

Hinweis

Diese Methode gibt nicht das Ergebnis des letzten Aufrufs für die Connection.synchronize-Methode zurück. Um das SyncResult-Objekt für den letzten Connection.synchronize(SyncParms)-Methodenaufruf abzurufen, wenden Sie die getSyncResult-Methode auf das übergebene SyncParms-Objekt an.

Siehe auch

- [SyncResult-Klasse \[UltraLiteJ\] auf Seite 268](#)
- [SyncParms.getSyncResult-Methode \[UltraLiteJ\] auf Seite 257](#)

isSynchronizationDeleteDisabled-Methode [BlackBerry]

Ermittelt, ob die Synchronisation von Löschvorgängen deaktiviert ist.

Syntax

```
boolean Connection.isSynchronizationDeleteDisabled()
```

Rückgabe

TRUE nur, wenn die Synchronisation von Löschvorgängen deaktiviert ist.

prepareStatement-Methode

Bereitet eine Anweisung für die Ausführung vor.

Syntax

```
PreparedStatement Connection.prepareStatement(
    String sql
) throws ULjException
```

Parameter

- **sql** Eine vorzubereitende SQL-Anweisung.

Rückgabe

Ein PreparedStatement-Objekt.

Siehe auch

- [PreparedStatement-Schnittstelle \[UltraLiteJ\] auf Seite 180](#)

registerForEvent-Methode [Android]

Registriert ein Systemereignis, um Benachrichtigungen zu erhalten.

Syntax

```
void Connection.registerForEvent(  
    short event_type,  
    String object_name  
) throws ULjException
```

Parameter

- **event_type** Der Typ des Ereignisses, für das eine Registrierung erfolgen soll.
- **object_name** Das Objekt, für das das Ereignis gilt, z.B. ein Tabellenname.

Siehe auch

- [ULjEvent-Schnittstelle \[Android\] \[UltraLiteJ\] auf Seite 285](#)

release-Methode

Gibt diese Verbindung frei.

Syntax

```
void Connection.release() throws ULjException
```

Bemerkungen

Sobald eine Verbindung freigegeben ist, kann sie nicht mehr für den Zugriff auf die Datenbank verwendet werden.

Es führt zu einem Fehler, wenn versucht wird, eine Verbindung freizugeben, bei der es nicht festgeschriebene Transaktionen gibt.

resetLastDownloadTime-Methode

Setzt die Zeit des Downloads bei den angegebenen Publikationen zurück.

Syntax

```
void Connection.resetLastDownloadTime(  
    String pub_name  
) throws ULjException
```

Parameter

- **pub_name** Der Name der zu überprüfenden Publikation.

Bemerkungen

Um die Downloadzeit für die Synchronisation der gesamten Datenbank neu einzustellen, verwenden Sie die Spezialpublikation `Connection.SYNC_ALL_DB_PUB_NAME`.

Diese Methode erfordert, dass es keine nicht festgeschriebenen Transaktionen auf der aktuellen Verbindung gibt.

rollback-Methode

Schreibt ein Rollback fest, um Änderungen in der Datenbank rückgängig zu machen.

Syntax

```
void Connection.rollback() throws ULjException
```

Bemerkungen

Das Aufrufen dieser Methode macht alle Änderungen an diesem Connection-Objekt rückgängig, die seit dem letzten Festschreiben oder Zurücksetzen vorgenommen wurden.

rollbackPartialDownload-Methode [Android]

Setzt die Änderungen einer fehlgeschlagenen Synchronisation zurück.

Syntax

```
void Connection.rollbackPartialDownload() throws ULjException
```

Bemerkungen

Diese Methode wirkt sich nur auf wiederaufnehmbare Downloads aus. (Synchronisation mit `SyncParams.setKeepPartialDownload` auf `TRUE` gesetzt)

Wenn während der Downloadphase der Synchronisation ein Kommunikationsfehler auftritt, während der `KeepPartialDownload`-Parameter den Wert `TRUE` hat, werden die heruntergeladenen Änderungen beibehalten, damit die Synchronisation an der Stelle wieder aufgenommen werden kann, an der der Download unterbrochen wurde.

Diese Methode verwirft den partiellen Download, wenn Sie nicht mehr wollen, dass der Download wieder aufgenommen werden soll.

Siehe auch

- [SyncParams.setKeepPartialDownload-Methode \[Android\] \[UltraLiteJ\] auf Seite 261](#)

setDatabaseId-Methode

Legt die Datenbank-ID fest, die für global autoincrement-Spalten verwendet wird.

Syntax

```
void Connection.setDatabaseId(int id) throws ULjException
```

Parameter

- **id** Die Datenbank-ID.

Bemerkungen

Die Datenbank-ID hat keinen Standardwert.

Während eines INSERT-Vorgangs wird in GLOBAL AUTOINCREMENT-Spalten NULL eingefügt, es sei denn, die Datenbank-ID wurde explizit gesetzt.

Siehe auch

- [Connection.getDatabaseId-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 114](#)

setOption-Methode

Setzt die Datenbankoption.

Syntax

```
void Connection.setOption(  
    String option_name,  
    String option_value  
) throws ULjException
```

Parameter

- **option_name** Der Name der festzulegenden Option. Bei Android-Geräten können Sie diesen Parameter auf den Namen einer beliebigen unterstützten UltraLite-Datenbankoption setzen. Bei BlackBerry-Geräten können Sie diesen Parameter auf eine beliebige Konstante in der Connection-Schnittstelle setzen, die das Präfix **OPTION_** hat.
- **option_value** Der neue Wert für die Option.

Bemerkungen

Wenn die Option derzeit nicht in der Datenbank gespeichert ist, wird sie erstellt.

Es darf keine nicht-festgeschriebenen Transaktionen bei dieser Verbindung geben, wenn diese Methode aufgerufen wird.

Siehe auch

- [Connection.getOption-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 116](#)
- [Connection.OPTION_BLOB_FILE_BASE_DIR-Variable \[BlackBerry\] \[UltraLiteJ\] auf Seite 125](#)
- [Connection.OPTION_DATABASE_ID-Variable \[BlackBerry\] \[UltraLiteJ\] auf Seite 125](#)
- [Connection.OPTION_DATE_FORMAT-Variable \[UltraLiteJ\] auf Seite 125](#)
- [Connection.OPTION_DATE_ORDER-Variable \[UltraLiteJ\] auf Seite 126](#)
- [Connection.OPTION_ML_REMOTE_ID-Variable \[BlackBerry\] \[UltraLiteJ\] auf Seite 127](#)
- [Connection.OPTION_NEAREST_CENTURY-Variable \[\] \[UltraLiteJ\] auf Seite 127](#)
- [Connection.OPTION_PRECISION-Variable \[UltraLiteJ\] auf Seite 128](#)
- [Connection.OPTION_SCALE-Variable \[UltraLiteJ\] auf Seite 128](#)
- [Connection.OPTION_TIME_FORMAT-Variable \[UltraLiteJ\] auf Seite 129](#)
- [Connection.OPTION_TIMESTAMP_FORMAT-Variable \[UltraLiteJ\] auf Seite 129](#)
- [Connection.OPTION_TIMESTAMP_INCREMENT-Variable \[UltraLiteJ\] auf Seite 130](#)
- [Connection.OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT-Variable \[UltraLiteJ\] auf Seite 130](#)

setSyncObserver-Methode

Setzt ein SyncObserver-Objekt, das den Fortschritt der Synchronisationen für diese Verbindung überwacht.

Syntax

```
void Connection.setSyncObserver(SyncObserver so)
```

Parameter

- **so** Ein SyncObserver-Objekt oder NULL, um das derzeit registrierte SyncObserver-Objekt zu entfernen.

Bemerkungen

Dieses SyncObserver-Objekt wird von nachfolgenden SYNCHRONIZE SQL-Anweisungen verwendet.

Der Standardwert ist NULL, womit kein Observer angegeben wird.

Siehe auch

- [SyncObserver-Schnittstelle \[UltraLiteJ\] auf Seite 244](#)

synchronize-Methode

Synchronisiert die Datenbank mit einem MobiLink-Server.

Syntax

```
void Connection.synchronize(SyncParms config) throws ULjException
```

Parameter

- **config** Das SyncParms-Objekt, das die für die Synchronisation verwendeten Parameter enthält.

Siehe auch

- [SyncParams-Klasse \[UltraLiteJ\] auf Seite 250](#)

unregisterForEvent-Methode [Android]

Hebt die Registrierung eines Systemereignisses auf, um den Empfang von Benachrichtigungen zu stoppen.

Syntax

```
void Connection.unregisterForEvent(  
    short event_type,  
    String object_name  
) throws ULjException
```

Parameter

- **event_type** Der Typ des Ereignisses, für das die Registrierung aufgehoben werden soll.
- **object_name** Das Objekt, für das das Ereignis gilt, z.B. ein Tabellennamen.

Siehe auch

- [ULjEvent-Schnittstelle \[Android\] \[UltraLiteJ\] auf Seite 285](#)

validateDatabase-Methode [Android]

Validiert die Datenbank in dieser Verbindung.

Syntax

```
void Connection.validateDatabase(  
    int flags,  
    ValidateDatabaseProgressListener listener,  
    String tableName  
) throws ULjException
```

Parameter

- **flags** Parameter, die die Art der Validierung steuern.
- **listener** Listener, der die Informationen über den Verarbeitungsfortschritt der Validierung erhält.
- **tableName** Eine bestimmte zu validierende Tabelle oder NULL für alle Tabellen.

Bemerkungen

Je nach den an diese Routine übergebenen Parametern können Tabellen, Indizes und Datenbankseiten validiert werden. Um während der Validierung Informationen zu erhalten, implementieren Sie eine Callback-Funktion und übergeben die Adresse an diese Routine. Um die Validierung auf eine bestimmte Tabelle zu beschränken, übergeben Sie den Tabellennamen oder die ID als letzten Parameter.

Der flags-Parameter ist eine Kombination aus den folgenden Werten:

- ULVF_TABLE
- ULVF_INDEX
- ULVF_DATABASE
- ULVF_EXPRESS
- ULVF_FULL_VALIDATE

Das folgende Beispiel zeigt die Tabelle und die Indexvalidierung im Express Modus:

```
flags = ULVF_TABLE | ULVF_INDEX | ULVF_EXPRESS;
```

Siehe auch

- [Connection.ULVF_TABLE-Variable \[Android\] \[UltraLiteJ\] auf Seite 133](#)
- [Connection.ULVF_INDEX-Variable \[Android\] \[UltraLiteJ\] auf Seite 133](#)
- [Connection.ULVF_DATABASE-Variable \[Android\] \[UltraLiteJ\] auf Seite 132](#)
- [Connection.ULVF_EXPRESS-Variable \[Android\] \[UltraLiteJ\] auf Seite 132](#)
- [Connection.ULVF_FULL_VALIDATE-Variable \[Android\] \[UltraLiteJ\] auf Seite 133](#)

waitForEvent-Methode [Android]

Wartet auf eine Ereignisbenachrichtigung.

Syntax

```
ULJEvent Connection.waitForEvent(int wait_ms) throws ULJException
```

Parameter

- **wait_ms** Die abzuwartende Zeitspanne (Blockierung in Millisekunden) vor der Rückgabe. Um unbegrenzt zu warten, setzen Sie den Wert auf -1.

Rückgabe

Das Ereignis, das innerhalb der Wartezeit auftrat, oder NULL, wenn innerhalb der Wartezeit keine Nachricht erhalten wurde.

Bemerkungen

Dieser Aufruf wird blockiert, bis eine Benachrichtigung empfangen wird oder die angegebene Wartezeit abgelaufen ist. Um einen Wartezustand zu beenden, verwenden Sie die cancelWaitForEvent-Methode.

Siehe auch

- [ULJEvent-Schnittstelle \[Android\] \[UltraLiteJ\] auf Seite 285](#)
- [Connection.cancelWaitForEvent-Methode \[Android\] \[UltraLiteJ\] auf Seite 109](#)

CONNECTED-Variable

Bezeichnet einen verbundenen Zustand.

Syntax

```
final byte Connection.CONNECTED
```

NOT_CONNECTED-Variable

Bezeichnet einen nicht verbundenen Zustand.

Syntax

```
final byte Connection.NOT_CONNECTED
```

OPTION_BLOB_FILE_BASE_DIR-Variable [BlackBerry]

Datenbankoption: Basisverzeichnis der Blob-Datei.

Syntax

```
final String Connection.OPTION_BLOB_FILE_BASE_DIR
```

Bemerkungen

Bei BlackBerry-Geräten ist der Standardwert der entsprechenden Option "file:///SDCard/", sonst "".

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)
- „blob_file_base_dir-Option der UltraLite Java Edition“ [[UltraLite - Datenbankverwaltung](#)]

OPTION_DATABASE_ID-Variable [BlackBerry]

Datenbankoption: Datenbank-ID.

Syntax

```
final String Connection.OPTION_DATABASE_ID
```

Bemerkungen

Es ist kein Standardwert angegeben. Er muss explizit zugeordnet werden.

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)

OPTION_DATE_FORMAT-Variable

Datenbankoption: Datumsformat.

Syntax

```
final String Connection.OPTION_DATE_FORMAT
```

Bemerkungen

Verwenden Sie bei BlackBerry-Smartphones diese Konstante mit der `Connection.setOption`-Methode.

Für Android-Smartphones setzen Sie diese Option nur in der Datenbank-Erstellungszeichenfolge für die `ConfigPersistent.setCreationString`-Methode.

Der Standardwert für die entsprechende Option ist "JJJJ-MM-TT".

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)
- [ConfigPersistent.setCreationString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)
- „date_format-Option der UltraLite Java Edition“ [*UltraLite - Datenbankverwaltung*]

OPTION_DATE_ORDER-Variable

Datenbankoption: Datumsreihenfolge.

Syntax

```
final String Connection.OPTION_DATE_ORDER
```

Bemerkungen

Verwenden Sie bei BlackBerry-Smartphones diese Konstante mit der `Connection.setOption`-Methode.

Für Android-Smartphones setzen Sie diese Option in der Datenbank-Erstellungszeichenfolge nur für die `ConfigPersistent.setCreationString`-Methode.

Der Standardwert für die entsprechende Option ist "JMT".

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)
- [ConfigPersistent.setCreationString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)
- „date_order-Option der UltraLite Java Edition“ [*UltraLite - Datenbankverwaltung*]

OPTION_MAX_HASH_SIZE-Variable

Datenbankoption: maximale Hash-Größe.

Syntax

```
final String Connection.OPTION_MAX_HASH_SIZE
```

Bemerkungen

Verwenden Sie bei BlackBerry-Smartphones diese Konstante mit der `Connection.setOption`-Methode, um die `MaxHashSize`-Datenbankeigenschaft einzustellen.

Für Android-Smartphones setzen Sie diese Option in der Datenbank-Erstellungszeichenfolge nur für die `ConfigPersistent.setCreationString`-Methode. Der Optionsname lautet `max_hash_size`.

Der Standardwert für die entsprechende Option ist 4".

Wenn die SQL-Anweisung zur Erstellung eines Indexes keine Hash-Größe angibt, wird der von dieser Option angegebene Wert als Standardwert verwendet.

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)
- [ConfigPersistent.setCreationString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)
- „Index-Hash-Methode“ [*UltraLite - Datenbankverwaltung*]

OPTION_ML_REMOTE_ID-Variable [BlackBerry]

Datenbankoption: Entfernte ML-ID.

Syntax

```
final String Connection.OPTION_ML_REMOTE_ID
```

Bemerkungen

Es ist kein Standardwert angegeben. Ein Wert wird nach der ersten MobiLink-Synchronisation gesetzt.

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)

OPTION_ML_SERVER_VERSION-Variable [BlackBerry]

Datenbankoption: Protokollversion des MobiLink-Servers.

Syntax

```
final String Connection.OPTION_ML_SERVER_VERSION
```

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)

OPTION_NEAREST_CENTURY-Variable

Datenbankoption: Nächstliegendes Jahrhundert.

Syntax

```
final String Connection.OPTION_NEAREST_CENTURY
```

Bemerkungen

Verwenden Sie bei BlackBerry-Smartphones diese Konstante mit der Connection.setOption-Methode.

Für Android-Smartphones setzen Sie diese Option in der Datenbank-Erstellungszeichenfolge nur für die `ConfigPersistent.setCreationString`-Methode.

Der Standardwert für die entsprechende Option ist "50".

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)
- [ConfigPersistent.setCreationString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)
- „nearest_century-Option der UltraLite Java Edition“ [*UltraLite - Datenbankverwaltung*]

OPTION_PRECISION-Variable

Datenbankoption: Gesamtstellenzahl.

Syntax

```
final String Connection.OPTION_PRECISION
```

Bemerkungen

Verwenden Sie bei BlackBerry-Smartphones diese Konstante mit der `Connection.setOption`-Methode.

Für Android-Smartphones setzen Sie diese Option in der Datenbank-Erstellungszeichenfolge nur für die `ConfigPersistent.setCreationString`-Methode.

Der Standardwert für die entsprechende Option ist "30".

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)
- [ConfigPersistent.setCreationString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)
- „precision-Option der UltraLite Java Edition“ [*UltraLite - Datenbankverwaltung*]

OPTION_SCALE-Variable

Datenbankoption: Dezimalstellen.

Syntax

```
final String Connection.OPTION_SCALE
```

Bemerkungen

Verwenden Sie bei BlackBerry-Smartphones diese Konstante mit der `Connection.setOption`-Methode.

Für Android-Smartphones setzen Sie diese Option in der Datenbank-Erstellungszeichenfolge nur für die `ConfigPersistent.setCreationString`-Methode.

Der Standardwert für die entsprechende Option ist "6".

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)
- [ConfigPersistent.setCreationString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)
- [„scale-Option der UltraLite Java Edition“ \[*UltraLite - Datenbankverwaltung*\]](#)

OPTION_TIME_FORMAT-Variable

Datenbankoption: Zeitformat.

Syntax

```
final String Connection.OPTION_TIME_FORMAT
```

Bemerkungen

Verwenden Sie bei BlackBerry-Smartphones diese Konstante mit der Connection.setOption-Methode.

Für Android-Smartphones setzen Sie diese Option in der Datenbank-Erstellungszeichenfolge nur für die ConfigPersistent.setCreationString-Methode.

Der Standardwert für die entsprechende Option ist "HH:NN:SS.SSS".

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)
- [ConfigPersistent.setCreationString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)
- [„time_format-Option der UltraLite Java Edition“ \[*UltraLite - Datenbankverwaltung*\]](#)

OPTION_TIMESTAMP_FORMAT-Variable

Datenbankoption: Zeitstempelformat.

Syntax

```
final String Connection.OPTION_TIMESTAMP_FORMAT
```

Bemerkungen

Verwenden Sie bei BlackBerry-Smartphones diese Konstante mit der Connection.setOption-Methode.

Für Android-Smartphones setzen Sie diese Option in der Datenbank-Erstellungszeichenfolge nur für die ConfigPersistent.setCreationString-Methode.

Der Standardwert für die entsprechende Option ist "JJJJ-MM-TT HH:NN:SS.SSS".

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)
- [ConfigPersistent.setCreationString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)
- [„timestamp_format-Option der UltraLite Java Edition“ \[*UltraLite - Datenbankverwaltung*\]](#)

OPTION_TIMESTAMP_INCREMENT-Variable

Datenbankoption: Zeitstempel-Inkrement.

Syntax

```
final String Connection.OPTION_TIMESTAMP_INCREMENT
```

Bemerkungen

Verwenden Sie bei BlackBerry-Smartphones diese Konstante mit der Connection.setOption-Methode.

Für Android-Smartphones setzen Sie diese Option in der Datenbank-Erstellungszeichenfolge nur für die ConfigPersistent.setCreationString-Methode.

Der Standardwert für die entsprechende Option ist "1".

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)
- [ConfigPersistent.setCreationString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)
- „timestamp_increment-Option der UltraLite Java Edition“ [[UltraLite - Datenbankverwaltung](#)]

OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT-Variable

Datenbankoption: Zeitstempel mit Zeitzonenformat.

Syntax

```
final String Connection.OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT
```

Bemerkungen

Verwenden Sie bei BlackBerry-Smartphones diese Konstante mit der Connection.setOption-Methode.

Für Android-Smartphones setzen Sie diese Option in der Datenbank-Erstellungszeichenfolge nur für die ConfigPersistent.setCreationString-Methode.

Der Standardwert der entsprechenden Option ist "JJJJ-MM-TT HH:NN:SS.SSS+HH:NN".

Siehe auch

- [Connection.setOption-Methode \[UltraLiteJ\] auf Seite 121](#)
- [ConfigPersistent.setCreationString-Methode \[Android\] \[UltraLiteJ\] auf Seite 99](#)
- „timestamp_with_time_zone_format-Option der UltraLite Java Edition“ [[UltraLite - Datenbankverwaltung](#)]

PROPERTY_DATABASE_NAME-Variable

Datenbankeigenschaft: Datenbankname.

Syntax

```
final String Connection.PROPERTY_DATABASE_NAME
```

Bemerkungen

Setzen Sie diese Eigenschaft mit der Configuration.setDatabaseName-Methode.

Siehe auch

- [Configuration.setDatabaseName-Methode \[UltraLiteJ\] auf Seite 103](#)
- [Connection.getDatabaseProperty-Methode \[UltraLiteJ\] auf Seite 114](#)

PROPERTY_PAGE_SIZE-Variable

Datenbankeigenschaft: Seitengröße.

Syntax

```
final String Connection.PROPERTY_PAGE_SIZE
```

Bemerkungen

Setzen Sie diese Eigenschaft mit der Configuration.setPageSize-Methode.

Siehe auch

- [Configuration.setPageSize-Methode \[UltraLiteJ\] auf Seite 104](#)
- [Connection.getDatabaseProperty-Methode \[UltraLiteJ\] auf Seite 114](#)

SYNC_ALL-Variable

Die Publikationsliste, mit der die Synchronisation aller Tabellen in der Datenbank angefordert wird, einschließlich der Tabellen, die in keiner Publikation verwendet werden.

Syntax

```
final String Connection.SYNC_ALL
```

Bemerkungen

Als "NoSync" gekennzeichnete Tabellen werden nie synchronisiert.

Diese Konstante ist mit der Nullreferenz oder einer leeren Zeichenfolge gleichwertig.

SYNC_ALL_DB_PUB_NAME -Variable

Der reservierte Name für die SYNC_ALL_PUB-Publikation.

Syntax

```
final String Connection.SYNC_ALL_DB_PUB_NAME
```

Siehe auch

- [Connection.getLastDownloadTime-Methode \[UltraLiteJ\] auf Seite 115](#)
- [Connection.resetLastDownloadTime-Methode \[UltraLiteJ\] auf Seite 119](#)

SYNC_ALL_PUBS-Variable

Die Publikationsliste, mit der die Synchronisation aller Publikationen in der Datenbank angefordert wird.

Syntax

```
final String Connection.SYNC_ALL_PUBS
```

Bemerkungen

Als "NoSync" gekennzeichnete Tabellen werden nie synchronisiert.

ULVF_DATABASE-Variable [Android]

Zur Validierung der Datenbank.

Syntax

```
final int Connection.ULVF_DATABASE
```

Bemerkungen

Überprüfen Sie alle Datenbankseiten mithilfe von Prüfsummen und zusätzlichen Prüfungen.

Siehe auch

- [Connection.validateDatabase-Methode \[Android\] \[UltraLiteJ\] auf Seite 123](#)

ULVF_EXPRESS-Variable [Android]

Für eine schnellere, allerdings weniger präzise Validierung.

Syntax

```
final int Connection.ULVF_EXPRESS
```

Bemerkungen

Dieser Parameter ändert andere angegebene Parameter.

Siehe auch

- [Connection.validateDatabase-Methode \[Android\] \[UltraLiteJ\] auf Seite 123](#)

ULVF_FULL_VALIDATE-Variable [Android]

Führt alle Arten von Validierungen der Datenbank durch.

Syntax

```
final int Connection.ULVF_FULL_VALIDATE
```

Siehe auch

- [Connection.validateDatabase-Methode \[Android\] \[UltraLiteJ\] auf Seite 123](#)

ULVF_INDEX-Variable [Android]

Zur Validierung von Indizes.

Syntax

```
final int Connection.ULVF_INDEX
```

Bemerkungen

Prüfen Sie die Integrität des Indexes.

Siehe auch

- [Connection.validateDatabase-Methode \[Android\] \[UltraLiteJ\] auf Seite 123](#)

ULVF_TABLE-Variable [Android]

Zum Validieren von Tabellen.

Syntax

```
final int Connection.ULVF_TABLE
```

Bemerkungen

Prüfen Sie, ob die Zeilenanzahlen von Tabelle und Index übereinstimmen.

Siehe auch

- [Connection.validateDatabase-Methode \[Android\] \[UltraLiteJ\] auf Seite 123](#)

DatabaseInfo-Schnittstelle

Ist einem Connection-Objekt zugeordnet und stellt Methoden bereit, um Datenbankinformationen zu beziehen.

Syntax

```
public interface DatabaseInfo
```

Mitglieder

Alle Mitglieder der DatabaseInfo-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getCommitCount-Methode [BlackBerry]	Gibt die Gesamtzahl der festgeschriebenen Vorgänge zurück, die in der Datenbank durchgeführt wurden.
getDbFormat-Methode [BlackBerry]	Gibt die Datenbank-Versionsnummer zurück.
getDbSize-Methode [BlackBerry]	Gibt die Datenbankgröße zurück.
getLogSize-Methode [BlackBerry]	Gibt die Gesamtgröße des Transaktionslogs (in Byte) zurück.
getNumberRowsToUpload-Methode	Gibt die Anzahl der Zeilen zurück, die auf einen Upload warten.
getPageReads-Methode	Gibt die Anzahl der Seitenlesevorgänge zurück.
getPageSize-Methode	Gibt die Seitengröße der Datenbank (in Byte) zurück
getPageWrites-Methode	Gibt die Anzahl der Seitenschreibvorgänge zurück.
getRelease-Methode	Gibt die Software-Versionsnummer zurück.

Bemerkungen

Diese Schnittstelle wird mit der getDatabaseInfo-Methode eines Connection-Objekts aufgerufen.

Siehe auch

- [Connection-Schnittstelle \[UltraLiteJ\] auf Seite 105](#)
- [Connection.getDatabaseInfo-Methode \[UltraLiteJ\] auf Seite 114](#)

getCommitCount-Methode [BlackBerry]

Gibt die Gesamtzahl der festgeschriebenen Vorgänge zurück, die in der Datenbank durchgeführt wurden.

Syntax

```
int DatabaseInfo.getCommitCount()
```

Rückgabe

Die Gesamtzahl der Festschreibungsvorgänge.

getDbFormat-Methode [BlackBerry]

Gibt die Datenbank-Versionsnummer zurück.

Syntax

```
int DatabaseInfo.getDbFormat( )
```

Rückgabe

Die Versionsnummer.

getDbSize-Methode [BlackBerry]

Gibt die Datenbankgröße zurück.

Syntax

```
int DatabaseInfo.getDbSize( )
```

Rückgabe

-1, wenn die Datenbank nicht beständig ist; sonst wird die aktuelle Größe des beständigen Speichers zurückgegeben.

getLogSize-Methode [BlackBerry]

Gibt die Gesamtgröße des Transaktionslogs (in Byte) zurück.

Syntax

```
int DatabaseInfo.getLogSize( )
```

Rückgabe

Die Größe des Transaktionslogs.

getNumberRowsToUpload-Methode

Gibt die Anzahl der Zeilen zurück, die auf einen Upload warten.

Überladungsliste

Name	Beschreibung
getNumberRowsToUpload()-Methode	Gibt die Anzahl der Zeilen zurück, die auf einen Upload warten.
getNumberRowsToUpload(String, int)-Methode [Android]	Gibt die Anzahl der Zeilen zurück, die auf einen Upload warten, bis zu einem bestimmten Schwellenwert.

getNumberRowsToUpload()-Methode

Gibt die Anzahl der Zeilen zurück, die auf einen Upload warten.

Syntax

```
int DatabaseInfo.getNumberRowsToUpload( )
```

Rückgabe

Die Anzahl der Zeilen.

getNumberRowsToUpload(String, int)-Methode [Android]

Gibt die Anzahl der Zeilen zurück, die auf einen Upload warten, bis zu einem bestimmten Schwellenwert.

Syntax

```
int DatabaseInfo.getNumberRowsToUpload(String pubList, int threshold)
```

Parameter

- **pubList** Eine Zeichenfolge mit einer kommagetrennten Liste von Publikationen, die geprüft werden sollen. Eine leere Zeichenfolge (der UL_SYNC_ALL-Makro) bedeutet, alle Tabellen außer den als **no sync** markierten. Eine Zeichenfolge, die nur ein Sternchen enthält (der UL_SYNC_ALL_PUBS-Makro), bedeutet alle Tabellen, die in einer Publikation referenziert werden. Einige Tabellen sind in keiner Publikation referenziert und werden daher nicht einbezogen, wenn dieser Wert * ist.
- **threshold** Die maximale Anzahl der zu zählenden Zeilen, wodurch die Zeitdauer des Aufrufs begrenzt wird. Der Schwellenwert 0 entspricht keiner Begrenzung (alle Zeilen, die synchronisiert werden müssen, werden gezählt). Der Schwellenwert 1 kann verwendet werden, um schnell zu ermitteln, ob Zeilen synchronisiert werden müssen.

Rückgabe

Die Anzahl der Zeilen, die synchronisiert werden müssen, entweder in einer angegebenen Gruppe von Publikationen oder in der gesamten Datenbank.

getPageReads-Methode

Gibt die Anzahl der Seitenlesevorgänge zurück.

Syntax

```
int DatabaseInfo.getPageReads( )
```

Rückgabe

Die Anzahl der Seitenlesevorgänge.

Bemerkungen

Für Android ist es die Anzahl der angesammelten Seitenlesevorgänge, die in einer Instanzvariablen dieser Klasse gespeichert wird. Für Blackberry und J2SE ist es die Anzahl der angesammelten Seitenlesevorgänge, die in der Datenbank gespeichert werden.

getPageSize-Methode

Gibt die Seitengröße der Datenbank (in Byte) zurück

Syntax

```
int DatabaseInfo.getPageSize( )
```

Rückgabe

Die Seitengröße.

getPageWrites-Methode

Gibt die Anzahl der Seitenschreibvorgänge zurück.

Syntax

```
int DatabaseInfo.getPageWrites( )
```

Rückgabe

Die Anzahl der Seitenschreibvorgänge. Für Android ist es die Anzahl der angesammelten Seitenschreibvorgänge, die in einer Instanzvariablen dieser Klasse gespeichert wird. Für Blackberry und J2SE ist es die Anzahl der angesammelten Seitenschreibvorgänge, die in der Datenbank gespeichert werden.

getRelease-Methode

Gibt die Software-Versionsnummer zurück.

Syntax

```
String DatabaseInfo.getRelease( )
```

Rückgabe

Die Versionsnummer.

Bemerkungen

Beispiel: Der Software-Versionswert von "12.0.1.1234" steht für die Version 12.0.1 und die Build-Nummer 1234.

DatabaseManager-Klasse

Stellt statische Methoden bereit, um grundlegende Konfigurationen zu beziehen, eine neue Datenbank zu erstellen und mit einer vorhandenen Datenbank zu verbinden.

Syntax

```
public class DatabaseManager
```

Mitglieder

Alle Mitglieder der DatabaseManager-Klasse einschließlich aller geerbten Mitglieder.

Name	Beschreibung
connect-Methode	Stellt eine Verbindung mit einer bestehenden Datenbank her, basierend auf einer Konfigurationsmenge.
createConfigurationNonPersistent-Methode [BlackBerry]	Erstellt ein Configuration-Objekt für einen nicht-beständigen Datenbankspeicher und gibt ein ConfigNonPersist-Objekt zurück.
createConfigurationObjectStore-Methode [BlackBerry]	Erstellt ein Configuration-Objekt für einen RIM-Objektspeicher und gibt ein ConfigObjectStore-Objekt zurück.
createDatabase-Methode	Erstellt eine neue Datenbank basierend auf einer Gruppe von Konfigurationen und stellt eine Verbindung mit der Datenbank her.
createFileTransfer-Methode	Erstellt ein FileTransfer-Objekt für die Übertragung von Dateien in oder aus MobiLink.
createFileTransferAndroid-Methode [Android]	Erstellt ein FileTransfer-Objekt für die Übertragung von Dateien in oder aus MobiLink.
createObjectStoreTransfer-Methode [BlackBerry]	Erstellt ein FileTransfer-Objekt für die Übertragung von UltraLite Java Edition-Datenbankdateien in oder aus MobiLink und deren Speicherung in einem RIM-Objektspeicher.
createSISHTTPListener-Methode [BlackBerry]	Erstellt ein SISListener-Objekt für serverinitiierte Synchronisationen.
release-Methode	Schließt dieses DatabaseManager-Objekt, um alle Verbindungen freizugeben und alle Datenbanken herunterzufahren.
setErrorLanguage-Methode	Setzt die Sprache für Fehlermeldungen.

Bemerkungen

Das folgende Beispiel zeigt, wie Sie auf der J2SE-Plattform eine vorhandene Datenbank öffnen bzw. eine neue erstellen, falls keine vorhanden ist:

```

Connection conn;
ConfigFile config = DatabaseManager.createConfigurationFile(
    "test.ulj"
);
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}

```

Das folgende Beispiel zeigt, wie Sie auf einem BlackBerry-Gerät eine vorhandene Datenbank öffnen bzw. eine neue erstellen, falls keine vorhanden ist:

```

Connection conn;
ConfigObjectStore config = DatabaseManager.createConfigurationObjectStore(
    "test.ulj"
);
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}

```

Das folgende Beispiel zeigt, wie Sie auf einem Android-Gerät eine vorhandene Datenbank öffnen bzw. eine neue erstellen, falls keine vorhanden ist:

```

Connection conn = null;
ConfigFileAndroid config = null;

try {
    config = DatabaseManager.createConfigurationFileAndroid(
        "test.udb", getApplicationContext()
    );
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    if (config != null) {
        try {
            conn = DatabaseManager.createDatabase(config);
            // Create the schema here.
        } catch(ULjException exception) {
            // An error has occurred.
        }
    }
}

```

Siehe auch

- [Connection-Schnittstelle \[UltraLiteJ\] auf Seite 105](#)
- [Configuration-Schnittstelle \[UltraLiteJ\] auf Seite 102](#)

connect-Methode

Stellt eine Verbindung mit einer bestehenden Datenbank her, basierend auf einer Konfigurationsmenge.

Syntax

```
Connection DatabaseManager.connect(  
    Configuration config  
) throws ULjException
```

Parameter

- **config** Das Configuration-Objekt mit den Spezifikationen für die vorhandene Datenbank.

Rückgabe

Ein Connection-Objekt, das die Verbindung mit der Datenbank herstellt.

Bemerkungen

Es kann maximal eine Anwendung eine Verbindung mit einer UltraLite Java Edition-Datenbank haben.

Siehe auch

- [Configuration-Schnittstelle \[UltraLiteJ\] auf Seite 102](#)
- [Connection-Schnittstelle \[UltraLiteJ\] auf Seite 105](#)
- [„UltraLite- und UltraLite Java Edition-Datenbank, Methoden zum Erstellen und Verbinden“ auf Seite 5](#)

createConfigurationFile-Methode

Erstellt ein Configuration-Objekt für einen physischen Datenbankspeicher aus einer Datei und gibt ein ConfigFile-Objekt zurück.

Syntax

```
ConfigFile DatabaseManager.createConfigurationFile(  
    String file_name  
) throws ULjException
```

Parameter

- **file_name** Der Name der Datei, die verwendet oder erstellt werden soll.

Rückgabe

Das ConfigFile-Objekt, das zum Konfigurieren einer Datenbank verwendet wird.

Siehe auch

- [ConfigFile-Schnittstelle \[UltraLiteJ\] auf Seite 87](#)

createConfigurationFileAndroid-Methode

Erstellt ein Configuration-Objekt für einen physischen Datenbankspeicher aus einer Datei auf einem Android-Gerät, und gibt ein ConfigFileAndroid-Objekt zurück.

Syntax

```
ConfigFileAndroid DatabaseManager.createConfigurationFileAndroid(  
    String file_name,  
    android.content.Context context  
) throws ULjException
```

Parameter

- **file_name** Der Name der Datenbankdatei, die verwendet oder erstellt werden soll. Sie müssen Schreib- und Lesezugriff auf den Datenbankpfad haben. Der Standardpfad für diese Datei ist */data/data/your-application-package-name/*, wobei *your-application-package-name* der Paketname ist, den Sie Ihrer Anwendung zugewiesen haben. Sie können einen absoluten Pfad mit dem Dateinamen angeben, um einen anderen Speicherort für die Datenbank festzulegen.
- **context** Das Context-Objekt aus einer Android-Anwendung. Dieser Parameter darf nicht NULL sein.

Rückgabe

Das ConfigObjectAndroid-Objekt, das zum Konfigurieren einer Datenbank verwendet wird.

Siehe auch

- [ConfigFileAndroid-Schnittstelle \[Android\] \[UltraLiteJ\] auf Seite 89](#)

createConfigurationNonPersistent-Methode [BlackBerry]

Erstellt ein Configuration-Objekt für einen nicht-beständigen Datenbankspeicher und gibt ein ConfigNonPersist-Objekt zurück.

Syntax

```
ConfigNonPersistent DatabaseManager.createConfigurationNonPersistent(  
    String db_name  
) throws ULjException
```

Parameter

- **db_name** Der Name der nicht-beständigen Datenbank

Rückgabe

Das ConfigNonPersistent-Objekt, das zum Konfigurieren der Datenbank verwendet wird.

Siehe auch

- [ConfigNonPersistent-Schnittstelle \[BlackBerry\] \[UltraLiteJ\] auf Seite 90](#)

createConfigurationObjectStore-Methode [BlackBerry]

Erstellt ein Configuration-Objekt für einen RIM-Objektspeicher und gibt ein ConfigObjectStore-Objekt zurück.

Syntax

```
ConfigObjectStore DatabaseManager.createConfigurationObjectStore(  
    String db_name  
) throws ULjException
```

Parameter

- **db_name** Der Name der Datenbank.

Rückgabe

Das ConfigObjectStore-Objekt, das zum Konfigurieren der Datenbank verwendet wird.

Siehe auch

- [ConfigObjectStore-Schnittstelle \[BlackBerry\] \[UltraLiteJ\] auf Seite 91](#)

createDatabase-Methode

Erstellt eine neue Datenbank basierend auf einer Gruppe von Konfigurationen und stellt eine Verbindung mit der Datenbank her.

Syntax

```
Connection DatabaseManager.createDatabase(  
    Configuration config  
) throws ULjException
```

Parameter

- **config** Ein Configuration-Objekt mit den Spezifikationen für die neue Datenbank.

Rückgabe

Ein Connection-Objekt, das eine Verbindung mit der neuen Datenbank herstellt.

Bemerkungen

Diese Methode ersetzt jede vorhandene Datenbank auf dem Gerät, die den gleichen Namen hat.

Siehe auch

- [Configuration-Schnittstelle \[UltraLiteJ\] auf Seite 102](#)
- [Connection-Schnittstelle \[UltraLiteJ\] auf Seite 105](#)

createFileTransfer-Methode

Erstellt ein FileTransfer-Objekt für die Übertragung von Dateien in oder aus MobiLink.

Syntax

```
FileTransfer DatabaseManager.createFileTransfer(  
    String fileName,  
    int streamType,
```

```
        String userName,  
        String version  
    ) throws ULjException
```

Parameter

- **fileName** Der Name der zu übertragenden Serverdatei. Dieser Parameter darf keine Pfadinformationen enthalten.
- **streamType** Eine der in der SyncParms-Klasse festgelegten Konstanten, die zur Kennzeichnung des Typs des Kommunikationsdatenstroms verwendet werden.
- **userName** Der MobiLink-Benutzername
- **version** Die MobiLink-Skriptversion.

Rückgabe

Das FileTransfer-Objekt.

Siehe auch

- [SyncParms-Klasse \[UltraLiteJ\] auf Seite 250](#)

createFileTransferAndroid-Methode [Android]

Erstellt ein FileTransfer-Objekt für die Übertragung von Dateien in oder aus MobiLink.

Syntax

```
FileTransfer DatabaseManager.createFileTransferAndroid(  
    android.content.Context context,  
    String fileName,  
    int streamType,  
    String userName,  
    String version  
    ) throws ULjException
```

Parameter

- **context** Das Context-Objekt aus einer Android-Anwendung. Dieser Parameter darf nicht NULL sein.
- **fileName** Der Name der zu übertragenden Serverdatei. Dieser Parameter darf keine Pfadinformationen enthalten.
- **streamType** Eine der in der SyncParms-Klasse festgelegten Konstanten, die zur Kennzeichnung des Typs des Kommunikationsdatenstroms verwendet werden.
- **userName** Der MobiLink-Benutzername
- **version** Die MobiLink-Skriptversion.

Rückgabe

Das FileTransfer-Objekt.

Bemerkungen

Diese Methode wird für Android empfohlen. Sie muss verwendet werden, wenn eine Datenbankverbindung noch nicht mit der `createConfigurationFileAndroid`-Methode erstellt wurde.

Siehe auch

- [DatabaseManager.createConfigurationFileAndroid-Methode \[UltraLiteJ\] auf Seite 140](#)

createObjectStoreTransfer-Methode [BlackBerry]

Erstellt ein FileTransfer-Objekt für die Übertragung von UltraLite Java Edition-Datenbankdateien in oder aus MobiLink und deren Speicherung in einem RIM-Objektspeicher.

Syntax

```
FileTransfer DatabaseManager.createObjectStoreTransfer(  
    String fileName,  
    int streamType,  
    String userName,  
    String version  
) throws ULjException
```

Parameter

- **fileName** Der Name der zu übertragenden Serverdatei. Dieser Parameter darf keine Pfadinformationen enthalten.
- **streamType** Eine der in der SyncParms-Klasse festgelegten Konstanten, die zur Kennzeichnung des Typs des Kommunikationsdatenstroms verwendet werden.
- **userName** Der MobiLink-Benutzername
- **version** Die MobiLink-Skriptversion.

Rückgabe

Das FileTransfer-Objekt.

Siehe auch

- [SyncParms-Klasse \[UltraLiteJ\] auf Seite 250](#)

createSISHTTPListener-Methode [BlackBerry]

Erstellt ein SISListener-Objekt für serverinitiierte Synchronisationen.

Syntax

```
SISListener DatabaseManager.createSISHTTPListener(  
    SISRequestHandler handler,  
    int port,  
    String httpOptions  
) throws ULjException
```

Parameter

- **handler** Ein SISRequestHandler-Objekt, das für serverinitiierte Synchronisationsanforderungen angegeben wird.
- **port** Ein HTTP-Port zum Abhören von Servermeldungen.
- **httpOptions** Die HTTP-Optionen für die Verbindung zum Datenbankserver.

Rückgabe

Das für serverinitiierte Synchronisationen zu verwendende SISListener-Objekt.

Bemerkungen

4400 ist die empfohlene Porteinstellung.

"deviceside=false" ist die empfohlene HTTP-Option für BlackBerry-Simulatoren.

Die MobiLink-Seite des BlackBerry-HTTP-SISListeners erfordert, dass das Zielgerät einem BlackBerry Enterprise Server zugeordnet wird, z.B. einem BES-aktivierten Gerät.

release-Methode

Schließt dieses DatabaseManager-Objekt, um alle Verbindungen freizugeben und alle Datenbanken herunterzufahren.

Syntax

```
void DatabaseManager.release() throws ULjException
```

Bemerkungen

Bei Android gibt diese Methode alle Verbindungen frei, die mit diesem DatabaseManager erstellt wurden.

Jede nicht-festgeschriebene Transaktion wird zurückgesetzt.

setErrorLanguage-Methode

Setzt die Sprache für Fehlermeldungen.

Syntax

```
void DatabaseManager.setErrorLanguage(String lang)
```

Parameter

- **lang** Der aus zwei Buchstaben bestehende Sprachencode.

Bemerkungen

Erkannte Sprachen sind EN, DE, FR, JA, ZH. Wenn eine nicht erkannte Sprache angegeben wird, verwendet das System den Standardwert "EN".

In J2SE- und BlackBerry-Umgebungen wird die aktuelle Sprachumgebung verwendet, um die Standardsprache zu bestimmen.

DecimalNumber-Schnittstelle

Beschreibt einen exakten Dezimalwert und stellt dezimal-arithmetische Unterstützung für Java-Plattformen bereit, auf denen java.math.BigDecimal nicht zur Verfügung steht.

Syntax

```
public interface DecimalNumber
```

Mitglieder

Alle Mitglieder der DecimalNumber-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
add-Methode	Addiert zwei DecimalNumber-Objekte und gibt die Summe zurück.
divide-Methode	Dividiert das erste DecimalNumber-Objekt durch das zweite DecimalNumber-Objekt und gibt den Quotienten zurück.
getString-Methode	Gibt die Zeichenfolgendarstellung des DecimalNumber-Objekts zurück.
isNull-Methode	Ermittelt, ob das DecimalNumber-Objekt Null ist.
multiply-Methode	Multipliziert zwei DecimalNumber-Objekte und gibt das Produkt zurück.
set-Methode	Setzt das DecimalNumber-Objekt auf einen String-Wert.
setNull-Methode	Setzt das DecimalNumber-Objekt auf Null.
subtract-Methode	Subtrahiert das zweite DecimalNumber-Objekt vom ersten DecimalNumber-Objekt und gibt die Differenz zurück.

add-Methode

Addiert zwei DecimalNumber-Objekte und gibt die Summe zurück.

Syntax

```
DecimalNumber DecimalNumber.add(  
    DecimalNumber num1,  
    DecimalNumber num2  
) throws ULjException
```

Parameter

- **num1** Eine Zahl.
- **num2** Eine weitere Zahl.

Rückgabe

Die Summe aus num1 und num2.

divide-Methode

Dividiert das erste DecimalNumber-Objekt durch das zweite DecimalNumber-Objekt und gibt den Quotienten zurück.

Syntax

```
DecimalNumber DecimalNumber.divide(  
    DecimalNumber num1,  
    DecimalNumber num2  
) throws ULjException
```

Parameter

- **num1** Ein Dividend.
- **num2** Ein Divisor.

Rückgabe

Der Quotient aus num1 dividiert durch num2.

getString-Methode

Gibt die Zeichenfolgendarstellung des DecimalNumber-Objekts zurück.

Syntax

```
String DecimalNumber.getString() throws ULjException
```

Rückgabe

Der Zeichenfolgenwert.

isNull-Methode

Ermittelt, ob das DecimalNumber-Objekt Null ist.

Syntax

```
boolean DecimalNumber.isNull()
```

Rückgabe

TRUE, wenn das Objekt Null ist, ansonsten FALSE.

multiply-Methode

Multipliziert zwei DecimalNumber-Objekte und gibt das Produkt zurück.

Syntax

```
DecimalNumber DecimalNumber.multiply(  
    DecimalNumber num1,  
    DecimalNumber num2  
) throws ULjException
```

Parameter

- **num1** Ein Multiplikand.
- **num2** Ein Multiplikator.

Rückgabe

Das Produkt aus num1 und num2.

set-Methode

Setzt das DecimalNumber-Objekt auf einen String-Wert.

Syntax

```
void DecimalNumber.set(String value) throws ULjException
```

Parameter

- **value** Ein numerischer Wert, als Zeichenfolge dargestellt.

setNull-Methode

Setzt das DecimalNumber-Objekt auf Null.

Syntax

```
void DecimalNumber.setNull() throws ULjException
```

subtract-Methode

Subtrahiert das zweite DecimalNumber-Objekt vom ersten DecimalNumber-Objekt und gibt die Differenz zurück.

Syntax

```
DecimalNumber DecimalNumber.subtract(
    DecimalNumber num1,
    DecimalNumber num2
) throws ULjException
```

Parameter

- **num1** Ein Minuend.
- **num2** Ein Subtrahend.

Rückgabe

Die Differenzmenge zwischen num1 und num2.

Domain-Schnittstelle

Beschreibt die Typinformationen des Domain-Objekts für eine Spalte in einer Tabelle.

Syntax

```
public interface Domain
```

Mitglieder

Alle Mitglieder der Domain-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
BIG-Variable	Bezeichnet die Domänen-ID-Konstante für eine 64-Bit-Ganzzahl (SQL-Datentyp BIGINT).
BINARY-Variable	Bezeichnet die Domänen-ID-Konstante für ein Binärdatenobjekt mit variabler Länge mit einer Maximalgröße von <i>size</i> Byte (SQL-Datentyp BINARY(<i>size</i>)).
BIT-Variable	Bezeichnet die Domänen-ID-Konstante für ein Bit (SQL-Datentyp BIT).
DATE-Variable	Bezeichnet die Domänen-ID-Konstante für ein Datum (SQL-Datentyp DATE).
DOMAIN_MAX-Variable	Bezeichnet die maximalen Arten von Domärentypen.
DOUBLE-Variable	Bezeichnet die Domänen-ID-Konstante für einen 8-Bit-Gleitkommawert (SQL-Datentyp DOUBLE).

Name	Beschreibung
INTEGER-Variable	Bezeichnet die Domänen-ID-Konstante für eine 32-Bit-Ganzzahl (SQL-Datentyp INTEGER).
LONGBINARY-Variable	Bezeichnet die Domänen-ID-Konstante für einen beliebig langen Block von Binärdaten (BLOB) (SQL-Datentyp LONG BINARY).
LONGBINARYFILE-Variable	Bezeichnet die Domänen-ID-Konstante für eine beliebige Datei von Daten.
LONGVARCHAR-Variable	Bezeichnet die Domänen-ID-Konstante für einen beliebig langen Block von Zeichendaten (CLOB) (SQL-Datentyp LONG VARCHAR).
NUMERIC-Variable	Bezeichnet die Domänen-ID-Konstante für einen numerischen Wert mit fester Gesamtstellenzahl (Größe) und mit <i>scale</i> hinter dem Dezimalzeichen (SQL-Datentyp NUMERIC(<i>precision</i> , <i>scale</i>)).
REAL-Variable	Bezeichnet die Domänen-ID-Konstante für einen 4-Bit-Gleitkommawert (SQL-Datentyp REAL).
SHORT-Variable	Bezeichnet die Domänen-ID-Konstante für eine 16-Bit-Ganzzahl (SQL-Datentyp SMALLINT).
ST_GEOMETRY-Variable	Bezeichnet die Domänen-ID-Konstante für eine Geometrie (SQL-Datentyp GEOMETRY).
TIME-Variable	Bezeichnet die Domänen-ID-Konstante für eine Uhrzeit (SQL-Datentyp TIME).
TIMESTAMP-Variable	Bezeichnet die Domänen-ID-Konstante für einen Zeitstempel (SQL-Datentyp TIMESTAMP).
TIMESTAMP_ZONE-Variable	Bezeichnet die Domänen-ID-Konstante für einen Zeitstempel mit Zeitzone (SQL-Datentyp DATETIMEOFFSET).
TINY-Variable	Bezeichnet die Domänen-ID-Konstante für eine 8-Bit-Ganzzahl ohne Vorzeichen (SQL-Datentyp TINYINT).
UNSIGNED_BIG-Variable	Bezeichnet die Domänen-ID-Konstante für eine 64-Bit-Ganzzahl ohne Vorzeichen (SQL-Datentyp UNSIGNED BIGINT).
UNSIGNED_INTEGER-Variable	Bezeichnet die Domänen-ID-Konstante für eine 32-Bit-Ganzzahl ohne Vorzeichen (SQL-Datentyp UNSIGNED INTEGER).
UNSIGNED_SHORT-Variable	Bezeichnet die Domänen-ID-Konstante für eine 16-Bit-Ganzzahl ohne Vorzeichen (SQL-Datentyp UNSIGNED SMALLINT).

Name	Beschreibung
UUID-Variable	Bezeichnet die Domänen-ID-Konstante für ein UniqueIdentifier-Objekt (SQL-Datentyp UNIQUEIDENTIFIER).
VARCHAR-Variable	Bezeichnet die Domänen-ID-Konstante für eine Zeichenfolge mit variabler Länge mit einer Maximalgröße von <i>size</i> Byte (SQL-Datentyp VARCHAR(<i>size</i>)).

Bemerkungen

Diese Schnittstelle enthält Konstanten, um die verschiedenen Domänen zu bezeichnen, sowie Methoden, um Informationen aus einem Domain-Objekt zu extrahieren.

Ein Beispiel für die Erstellung eines Schemas für eine einfache Datenbank finden Sie in der Connection-Schnittstelle.

Typen können wie folgt klassifiziert werden:

Ganzzahltypen:

Domänenkonstante	SQL-Datentyp	Wertebereich
BIT	BIT	0 oder 1
TINY	TINYINT	0 bis 255 (Ganzzahl ohne Vorzeichen, die 1 Speicherbyte verwendet)
SHORT	SMALLINT	-32768 bis 32767 (Ganzzahl mit Vorzeichen, die 2 Speicherbyte verwendet)
UNSIGNED_SHORT	UNSIGNED SMALLINT	0 bis 65535 (Ganzzahl ohne Vorzeichen, die 2 Speicherbyte verwendet)
INTEGER	INTEGER	-2 ³¹ bis 2 ³¹ - 1, oder -2147483648 bis 2147483647 (Ganzzahl mit Vorzeichen, die 4 Speicherbyte verwendet)
UNSIGNED_INTEGER	UNSIGNED INTEGER	0 bis 2 ³² - 1, oder 0 bis 4294967295 (Ganzzahl ohne Vorzeichen, die 4 Speicherbyte verwendet)
BIG	BIGINT	-2 ⁶³ bis 2 ⁶³ - 1, oder -9223372036854775808 bis 9223372036854775807 (Ganzzahl mit Vorzeichen, die 8 Speicherbyte verwendet)
UNSIGNED_BIG	UNSIGNED BIGINT	0 bis 2 ⁶⁴ - 1, oder 0 bis 18446744073709551615 (Ganzzahl ohne Vorzeichen, die 8 Speicherbyte verwendet)

Nummerische Nicht-Ganzzahltypen:

Domänenkonstante	SQL-Datentyp	Wertebereich
REAL	REAL	-3.402823e+38 bis 3.402823e+38, mit Zahlen nahe Null nur 1.175495e-38 groß (Gleitkommazahl mit einfacher Genauigkeit, die 4 Speicherbyte verwendet und bei der Rundungsfehler nach der sechsten Stelle auftreten können)
DOUBLE	DOUBLE	-1.79769313486231e+308 to 1.79769313486231e+308, mit Zahlen nahe Null nur 2.22507385850721e-308 groß (Gleitkommazahl mit einfacher Genauigkeit, die 8 Speicherbyte verwendet und bei der Rundungsfehler nach der fünfzehnten Stelle auftreten können)
NUMERIC	NUMERIC(Gesamtstellenzahl, Dezimalstellen)	Jede Dezimalzahl mit insgesamt <i>precision</i> (Größe) Stellen und <i>scale</i> Stellen hinter dem Dezimalzeichen (kein Runden innerhalb der Gesamtstellenzahl)

Zeichen- und Binärdatentypen:

Domänenkonstante	SQL-Datentyp	Größenbereich
VARCHAR	VAR-CHAR(Größe)	1 bis 32767 Byte (Zeichen werden als 1- bis 3-Byte-UTF-8-Zeichen gespeichert). Beim Auswerten von Ausdrücken beträgt die maximale Länge eines temporären Zeichenwerts 2048 Byte.
LONGVARCHAR	LONG VARCHAR	Beliebige Länge (soweit sie der Speicher zulässt). Die einzigen Vorgänge, die in LONG VARCHAR-Spalten zulässig sind, sind das Einfügen, Aktualisieren oder Löschen der Spalten sowie das Eintragen in die Auswahlliste einer Abfrage.
BINARY	BINARY(Größe)	1 bis 32767 Byte. Beim Auswerten von Ausdrücken ist die Maximallänge eines temporären Zeichenwerts 2048 Byte.
LONGBINARY	LONG BINARY	Beliebige Länge (soweit sie der Speicher zulässt). Die einzigen Vorgänge, die in LONG BINARY-Spalten zulässig sind, sind das Einfügen, Aktualisieren oder Löschen der Spalten sowie das Eintragen in die Auswahlliste einer Abfrage.
UUID	UNIQUEIDENTIFIER	Immer 16 Byte-Binärdaten mit spezieller Interpretation.

Datums- und Uhrzeit-Datentypen:

Domänenkonstante	SQL-Datentyp	Wert
DATE	DATE	Jahr, Monat, Tag.
TIME	TIME	Stunde, Minute, Sekunde und Sekundenbruchteile
TIMESTAMP	TIMESTAMP	DATE und TIME
TIMESTAMP_ZONE	TIMESTAMP_ZONE	DATE und TIME mit Zeitzone.

BIT-Spalten sind standardmäßig nicht nullwertfähig. Alle anderen Datentypen sind standardmäßig nullwertfähig.

Siehe auch

- [Connection-Schnittstelle \[UltraLiteJ\] auf Seite 105](#)

BIG-Variable

Bezeichnet die Domänen-ID-Konstante für eine 64-Bit-Ganzzahl (SQL-Datentyp BIGINT).

Syntax

```
final short Domain.BIG
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

BINARY-Variable

Bezeichnet die Domänen-ID-Konstante für ein Binärdatenobjekt mit variabler Länge mit einer Maximalgröße von *size* Byte (SQL-Datentyp BINARY(*size*)).

Syntax

```
final short Domain.BINARY
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

BIT-Variable

Bezeichnet die Domänen-ID-Konstante für ein Bit (SQL-Datentyp BIT).

Syntax

```
final short Domain.BIT
```

Bemerkungen

BIT-Spalten sind standardmäßig nicht nullwertfähig.

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

DATE-Variable

Bezeichnet die Domänen-ID-Konstante für ein Datum (SQL-Datentyp DATE).

Syntax

```
final short Domain.DATE
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

DOMAIN_MAX-Variable

Bezeichnet die maximalen Arten von Domärentypen.

Syntax

```
final short Domain.DOMAIN_MAX
```

DOUBLE-Variable

Bezeichnet die Domänen-ID-Konstante für einen 8-Bit-Gleitkommawert (SQL-Datentyp DOUBLE).

Syntax

```
final short Domain.DOUBLE
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

INTEGER-Variable

Bezeichnet die Domänen-ID-Konstante für eine 32-Bit-Ganzzahl (SQL-Datentyp INTEGER).

Syntax

```
final short Domain.INTEGER
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

LONGBINARY-Variable

Bezeichnet die Domänen-ID-Konstante für einen beliebig langen Block von Binärdaten (BLOB) (SQL-Datentyp LONG BINARY).

Syntax

```
final short Domain.LONGBINARY
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

LONGBINARYFILE-Variable

Bezeichnet die Domänen-ID-Konstante für eine beliebige Datei von Daten.

Syntax

```
final short Domain.LONGBINARYFILE
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

LONGVARCHAR-Variable

Bezeichnet die Domänen-ID-Konstante für einen beliebig langen Block von Zeichendaten (CLOB) (SQL-Datentyp LONG VARCHAR).

Syntax

```
final short Domain.LONGVARCHAR
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

NUMERIC-Variable

Bezeichnet die Domänen-ID-Konstante für einen numerischen Wert mit fester Gesamtstellenzahl (Größe) und mit *scale* hinter dem Dezimalzeichen (SQL-Datentyp NUMERIC(*precision*,*scale*)).

Syntax

```
final short Domain.NUMERIC
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

REAL-Variable

Bezeichnet die Domänen-ID-Konstante für einen 4-Bit-Gleitkommawert (SQL-Datentyp REAL).

Syntax

```
final short Domain.REAL
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

SHORT-Variable

Bezeichnet die Domänen-ID-Konstante für eine 16-Bit-Ganzzahl (SQL-Datentyp SMALLINT).

Syntax

```
final short Domain.SHORT
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

ST_GEOMETRY-Variable

Bezeichnet die Domänen-ID-Konstante für eine Geometrie (SQL-Datentyp GEOMETRY).

Syntax

```
final short Domain.ST_GEOMETRY
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

TIME-Variable

Bezeichnet die Domänen-ID-Konstante für eine Uhrzeit (SQL-Datentyp TIME).

Syntax

```
final short Domain.TIME
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

TIMESTAMP-Variable

Bezeichnet die Domänen-ID-Konstante für einen Zeitstempel (SQL-Datentyp TIMESTAMP).

Syntax

```
final short Domain.TIMESTAMP
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

TIMESTAMP_ZONE-Variable

Bezeichnet die Domänen-ID-Konstante für einen Zeitstempel mit Zeitzone (SQL-Datentyp DATETIMEOFFSET).

Syntax

```
final short Domain.TIMESTAMP_ZONE
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

TINY-Variable

Bezeichnet die Domänen-ID-Konstante für eine 8-Bit-Ganzzahl ohne Vorzeichen (SQL-Datentyp TINYINT).

Syntax

```
final short Domain.TINY
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

UNSIGNED_BIG-Variable

Bezeichnet die Domänen-ID-Konstante für eine 64-Bit-Ganzzahl ohne Vorzeichen (SQL-Datentyp UNSIGNED BIGINT).

Syntax

```
final short Domain.UNSIGNED_BIG
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

UNSIGNED_INTEGER-Variable

Bezeichnet die Domänen-ID-Konstante für eine 32-Bit-Ganzzahl ohne Vorzeichen (SQL-Datentyp UNSIGNED INTEGER).

Syntax

```
final short Domain.UNSIGNED_INTEGER
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

UNSIGNED_SHORT-Variable

Bezeichnet die Domänen-ID-Konstante für eine 16-Bit-Ganzzahl ohne Vorzeichen (SQL-Datentyp UNSIGNED SMALLINT).

Syntax

```
final short Domain.UNSIGNED_SHORT
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

UUID-Variable

Bezeichnet die Domänen-ID-Konstante für ein UniqueIdentifier-Objekt (SQL-Datentyp UNIQUEIDENTIFIER).

Syntax

```
final short Domain.UUID
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

VARCHAR-Variable

Bezeichnet die Domänen-ID-Konstante für eine Zeichenfolge mit variabler Länge mit einer Maximalgröße von *size* Byte (SQL-Datentyp VARCHAR(*size*)).

Syntax

```
final short Domain.VARCHAR
```

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)

FileTransfer-Schnittstelle

Stellt einen Mechanismus zur Übertragung von Dateien zwischen dem Client und einem MobiLink-Server bereit.

Syntax

```
public interface FileTransfer
```

Mitglieder

Alle Mitglieder der FileTransfer-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
downloadFile-Methode	Lädt die Datei mit dem angegebenen Eigenschaften dieses Objekts herunter.
getAuthenticationParms-Methode	Gibt Parameter zurück, die für ein angepasstes Benutzerauthentifizierungsskript bestimmt sind.
getAuthStatus-Methode	Gibt den Autorisierungsstatuscode für den letzten Dateiübertragungsversuch zurück.
getAuthValue-Methode	Gibt den Wert zurück, der in Synchronisationsskripten mit angepasster Benutzerauthentifizierung angegeben ist.
getFileAuthCode-Methode	Gibt den Rückgabewert aus dem Skript <code>authenticate_file_transfer</code> für den letzten Dateiübertragungsversuch zurück.
getLivenessTimeout-Methode	Gibt die Länge des Verfügbarkeits-Timeouts (in Sekunden) zurück.
getLocalFileName-Methode	Ermittelt den Namen der lokalen Datei.
getLocalPath-Methode	Gibt an, wo die Datei im lokalen Dateisystem gesucht bzw. gespeichert werden soll.
getPassword-Methode	Gibt das MobiLink-Kennwort für den mit der <code>setUserName-Methode</code> angegebenen Benutzer zurück.
getRemoteKey-Methode	Ermittelt den aktuellen entfernten Schlüsselwert.
getServerFileName-Methode	Gibt den Namen der Datei auf dem Server zurück.
getStreamErrorCode-Methode	Gibt den Fehlercode zurück, der vom Datenstrom gemeldet wurde.

Name	Beschreibung
getStreamErrorMessage-Methode	Gibt den Fehler zurück, der vom Datenstrom selbst gemeldet wurde.
getStreamParms-Methode	Gibt die Parameter zur Konfiguration des Synchronisationsdatenstroms zurück.
getUserName-Methode	Gibt den MobiLink-Benutzernamen zurück, der den Client für den MobiLink-Server eindeutig kennzeichnet.
getVersion-Methode	Gibt das zu verwendende Synchronisationsskript zurück.
isResumePartialTransfer-Methode	Ermittelt, ob eine frühere teilweise Übertragung wieder aufgenommen oder verworfen werden soll.
isTransferredFile-Methode	Prüft, ob die Datei tatsächlich während des letzten Dateiübertragungsversuchs heruntergeladen wurde.
setAuthenticationParms-Methode	Gibt Parameter für ein angepasstes Benutzerauthentifizierungsskript (MobiLink authenticate_parameters-Verbindungsereignis) an.
setLivenessTimeout-Methode	Legt die Länge des Verfügbarkeits-Timeouts (in Sekunden) fest.
setLocalFileName-Methode	Gibt den Namen der lokalen Datei an.
setLocalPath-Methode	Gibt an, wo die Datei im lokalen Dateisystem gesucht bzw. gespeichert werden soll.
setPassword-Methode	Setzt das MobiLink-Kennwort für den mit der setUsername-Methode angegebenen Benutzer.
setRemoteKey-Methode	Gibt den entfernten Schlüssel an.
setResumePartialTransfer-Methode	Legt fest, ob eine frühere teilweise Übertragung wieder aufgenommen oder verworfen werden soll.
setServerFileName-Methode	Gibt den Namen der Datei auf dem Server an.
setUserName-Methode	Setzt den MobiLink-Benutzernamen, der den Client für den MobiLink-Server eindeutig kennzeichnet.
setVersion-Methode	Setzt das zu verwendende Synchronisationsskript.
uploadFile-Methode	Lädt die Datei mit den angegebenen Eigenschaften dieses Objekts hoch.

Bemerkungen

Ein FileTransfer-Objekt erhalten Sie durch den Aufrufen der Methode `DatabaseManager.createFileTransfer` bzw. `DatabaseManager.createObjectStoreTransfer` [BlackBerry].

Die von der `createFileTransfer`-Methode zurückgegebene Instanz kann verwendet werden, um Dateien zwischen MobiLink und dem lokalen Dateisystem zu übertragen.

Bei Android-Geräten und -Simulatoren ist das lokale Dateisystem entweder eine Speicherkarte oder das interne Dateisystem, auf dem die Anwendung die erforderlichen Berechtigungen hat, z.B. `/sdcard/Android/data/your.package.name/files/`

Die von der `createObjectStoreTransfer`-Methode zurückgegebene Instanz kann verwendet werden, um UltraLite Java Edition-Datenbankdateien in den lokalen BlackBerry-Objektspeicher herunterzuladen oder umgekehrt.

Es können nur gültige, nicht verschlüsselte UltraLite Java Edition-Datenbankdateien übertragen werden. Bei dem Versuch, andere Dateien als UltraLite Java Edition-Datenbanken herunterzuladen, werden Ausnahmebedingungen ausgegeben.

Hinweis

Die Anwendung darf nicht gleichzeitig zwei Downloads in dieselbe lokale Datei starten.

Siehe auch

- [DatabaseManager.createFileTransfer-Methode \[UltraLiteJ\] auf Seite 142](#)
- [DatabaseManager.createObjectStoreTransfer-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 144](#)

downloadFile-Methode

Lädt die Datei mit dem angegebenen Eigenschaften dieses Objekts herunter.

Überladungsliste

Name	Beschreibung
downloadFile()-Methode	Lädt die Datei mit dem angegebenen Eigenschaften dieses Objekts herunter.
downloadFile(FileTransfer-ProgressListener)-Methode	Lädt die Datei herunter, die durch die Eigenschaften dieser Datei festgelegt wird, wobei an den angegebenen Listener Fortschrittsereignisse gesendet werden.

downloadFile()-Methode

Lädt die Datei mit dem angegebenen Eigenschaften dieses Objekts herunter.

Syntax

```
abstract boolean FileTransfer.downloadFile() throws ULjException
```

Rückgabe

TRUE, wenn der Download erfolgreich ist. Andernfalls wird ein **ULjException**-Objekt erzeugt und die Methode liefert nicht die normale Rückgabe.

Bemerkungen

Die durch die `setServerFileName`-Methode angegebene Datei wird vom MobiLink-Server in den durch die `setLocalPath`-Methode angegebenen Pfad heruntergeladen. Dabei werden die angegebenen Werte für Datenstrom, Benutzername, Kennwort und Skriptversion verwendet.

Weitere Optionen können mithilfe der Methoden `setLocalFileName()`, `setAuthenticationParms()` und `setResumePartialTransfer()` angegeben werden.

Um eine Beschädigung der Datei zu vermeiden, führt UltraLiteJ sowohl bei Desktop- als auch bei BlackBerry-Datensystemen zunächst einen Download in eine temporäre Datei durch und ersetzt die lokale Datei erst nach Abschluss des Downloads.

Bei Downloads in BlackBerry-Objektspeicher beginnt UltraLiteJ direkt mit dem Schreiben in den lokalen Speicher. Der Grund hierfür ist, dass es in diesem Fall nicht möglich ist, atomare temporäre Objekte zu erstellen. Alle vorhandenen lokalen Datenbankspeicher mit demselben Namen werden beim Aufruf der `transferFile`-Methode beschädigt.

Ein ausführlicher Ergebnisstatus kann mithilfe der Methoden `getAuthStatus()`, `getAuthValue()`, `getFileAuthCode()`, `isTransferredFile()`, `getStreamErrorCode()` und `getStreamErrorMessage()` abgerufen werden.

downloadFile(FileTransferProgressListener)-Methode

Lädt die Datei herunter, die durch die Eigenschaften dieser Datei festgelegt wird, wobei an den angegebenen Listener Fortschrittsereignisse gesendet werden.

Syntax

```
abstract boolean FileTransfer.downloadFile(  
    FileTransferProgressListener listener  
) throws ULjException
```

Parameter

- **listener** Das Objekt, das Ereignisse zum Dateübertragungsfortschritt empfängt.

Rückgabe

TRUE, wenn der Download erfolgreich ist. Andernfalls wird ein **ULjException**-Objekt erzeugt und die Methode liefert nicht die normale Rückgabe.

Bemerkungen

Fehler können dazu führen, dass keine Daten an den Listener gesendet werden.

Siehe auch

- [FileTransfer.downloadFile-Methode \[UltraLiteJ\] auf Seite 161](#)

getAuthenticationParms-Methode

Gibt Parameter zurück, die für ein angepasstes Benutzerauthentifizierungsskript bestimmt sind.

Syntax

```
abstract String FileTransfer.getAuthenticationParms()
```

Rückgabe

Die Liste der Authentifizierungsparameter oder NULL, wenn keine Parameter angegeben sind.

Siehe auch

- [FileTransfer.setAuthenticationParms-Methode \[UltraLiteJ\] auf Seite 168](#)

getAuthStatus-Methode

Gibt den Autorisierungsstatuscode für den letzten Dateiübertragungsversuch zurück.

Syntax

```
abstract int FileTransfer.getAuthStatus()
```

Rückgabe

Ein AuthStatusCode-Klassenwert.

getAuthValue-Methode

Gibt den Wert zurück, der in Synchronisationsskripten mit angepasster Benutzerauthentifizierung angegeben ist.

Syntax

```
abstract long FileTransfer.getAuthValue()
```

Rückgabe

Eine Ganzzahl, die aus Synchronisationsskripten mit angepasster Benutzerauthentifizierung zurückgegeben wird.

getFileAuthCode-Methode

Gibt den Rückgabewert aus dem Skript `authenticate_file_transfer` für den letzten Dateiübertragungsversuch zurück.

Syntax

```
abstract int FileTransfer.getFileAuthCode()
```

Rückgabe

Eine Ganzzahl, die vom Skript `authenticate_file_transfer` für den letzten Dateiübertragungsversuch zurückgegeben wird.

getLivenessTimeout-Methode

Gibt die Länge des Verfügbarkeits-Timeouts (in Sekunden) zurück.

Syntax

```
abstract int FileTransfer.getLivenessTimeout()
```

Rückgabe

Der Timeout.

Siehe auch

- [FileTransfer.setLivenessTimeout-Methode \[UltraLiteJ\] auf Seite 168](#)

getLocalFileName-Methode

Ermittelt den Namen der lokalen Datei.

Syntax

```
abstract String FileTransfer.getLocalFileName()
```

Rückgabe

Der lokale Dateiname für die heruntergeladene Datei.

Bemerkungen

Bei Datei-Downloads ist dies der Name der heruntergeladenen Datei. Bei Datei-Uploads ist dies der Name der hochzuladenden Datei.

Siehe auch

- [FileTransfer.setLocalFileName-Methode \[UltraLiteJ\] auf Seite 169](#)

getLocalPath-Methode

Gibt an, wo die Datei im lokalen Dateisystem gesucht bzw. gespeichert werden soll.

Syntax

```
abstract String FileTransfer.getLocalPath()
```

Rückgabe

Das lokale Verzeichnis.

Siehe auch

- [FileTransfer.setLocalPath-Methode \[UltraLiteJ\] auf Seite 170](#)

getPassword-Methode

Gibt das MobiLink-Kennwort für den mit der setUsername-Methode angegebenen Benutzer zurück.

Syntax

```
abstract String FileTransfer.getPassword()
```

Rückgabe

Das Kennwort für den MobiLink-Benutzer.

Siehe auch

- [FileTransfer.setPassword-Methode \[UltraLiteJ\] auf Seite 170](#)

getRemoteKey-Methode

Ermittelt den aktuellen entfernten Schlüsselwert.

Syntax

```
abstract String FileTransfer.getRemoteKey()
```

Rückgabe

Der entfernte Schlüsselwert oder NULL, wenn der entfernte Schlüssel nicht festgelegt ist.

Siehe auch

- [FileTransfer.setRemoteKey-Methode \[UltraLiteJ\] auf Seite 171](#)

getServerFileName-Methode

Gibt den Namen der Datei auf dem Server zurück.

Syntax

```
abstract String FileTransfer.getServerFileName()
```

Rückgabe

Der serverseitige Name der Datei.

Bemerkungen

Bei Datei-Downloads ist dies der Name der herunterzuladenden Datei. Bei Datei-Uploads ist dies der Name der hochgeladenen Datei.

Siehe auch

- [FileTransfer.setServerFileName-Methode \[UltraLiteJ\] auf Seite 172](#)

getStreamErrorCode-Methode

Gibt den Fehlercode zurück, der vom Datenstrom gemeldet wurde.

Syntax

```
abstract int FileTransfer.getStreamErrorCode()
```

Rückgabe

0, wenn es keinen Kommunikationsdatenstrom-Fehler gab, ansonsten der Antwortcode vom Server.

Bemerkungen

Der Fehlercode ist der HTTP-Anwortcode.

getStreamErrorMessage-Methode

Gibt den Fehler zurück, der vom Datenstrom selbst gemeldet wurde.

Syntax

```
abstract String FileTransfer.getStreamErrorMessage()
```

Rückgabe

NULL, wenn keine Meldung verfügbar ist, ansonsten die Antwortmeldung.

Bemerkungen

Hierbei handelt es sich um die HTTP-Anwortmeldung.

getStreamParms-Methode

Gibt die Parameter zur Konfiguration des Synchronisationsdatenstroms zurück.

Syntax

```
abstract StreamHTTPParms FileTransfer.getStreamParms()
```

Rückgabe

Ein StreamHTTPParms- oder StreamHTTPSParms-Objekt, das die Parameter für die HTTP- bzw. HTTPS-Datenströme angibt. Das Objekt wird mittels Referenz zurückgegeben.

Bemerkungen

Der Typ des Synchronisationsdatenstroms wird angegeben, wenn das FileTransfer-Objekt erstellt wird.

getUserName-Methode

Gibt den MobiLink-Benutzernamen zurück, der den Client für den MobiLink-Server eindeutig kennzeichnet.

Syntax

```
abstract String FileTransfer.getUserName()
```

Rückgabe

Der MobiLink-Benutzername

Siehe auch

- [FileTransfer.setUserName-Methode \[UltraLiteJ\] auf Seite 173](#)

getVersion-Methode

Gibt das zu verwendende Synchronisationsskript zurück.

Syntax

```
abstract String FileTransfer.getVersion()
```

Rückgabe

Die Skriptversion

Siehe auch

- [FileTransfer.setVersion-Methode \[UltraLiteJ\] auf Seite 173](#)

isResumePartialTransfer-Methode

Ermittelt, ob eine frühere teilweise Übertragung wieder aufgenommen oder verworfen werden soll.

Syntax

```
abstract boolean FileTransfer.isResumePartialTransfer()
```

Rückgabe

TRUE, wenn der Download wieder aufgenommen werden soll, ansonsten FALSE.

Siehe auch

- [FileTransfer.setResumePartialTransfer-Methode \[UltraLiteJ\] auf Seite 171](#)

isTransferredFile-Methode

Prüft, ob die Datei tatsächlich während des letzten Dateiübertragungsversuchs heruntergeladen wurde.

Syntax

```
abstract boolean FileTransfer.isTransferredFile()
```

Rückgabe

TRUE, wenn die Datei übertragen wurde, sonst FALSE.

Bemerkungen

Wenn die Datei beim Aufruf der transferFile()-Methode bereits aktuell ist, gibt diese Methode TRUE zurück, während die isTransferredFile()-Methode FALSE zurückgibt.

Wenn ein Fehler auftritt und die transferFile()-Methode eine Ausnahme ausgibt, gibt die isTransferredFile()-Methode FALSE zurück.

setAuthenticationParms-Methode

Legt Parameter für ein angepasstes Benutzerauthentifizierungsskript (MobiLink authenticate_parameters-Verbindungsereignis) fest.

Syntax

```
abstract void FileTransfer.setAuthenticationParms(  
    String authParms  
) throws ULjException
```

Parameter

- **authParms** Eine durch Kommas getrennte Liste von Authentifizierungsparametern oder die Nullreferenz. Weitere Hinweise zu kommagetrennten Listen finden Sie in der Beschreibung der SyncParms-Klasse.

Bemerkungen

Nur die ersten 255 Zeichenfolgen werden verwendet und keine Zeichenfolge sollte länger als 128 Zeichen sein (längere Zeichenfolgen werden gekürzt, wenn sie an MobiLink gesendet werden).

Siehe auch

- [FileTransfer.getAuthenticationParms-Methode \[UltraLiteJ\] auf Seite 163](#)

setLivenessTimeout-Methode

Legt die Länge des Verfügbarkeits-Timeouts (in Sekunden) fest.

Syntax

```
abstract void FileTransfer.setLivenessTimeout(  
    int timeout  
) throws ULjException
```

Parameter

- **timeout** Der neue Wert des Verfügbarkeits-Timeouts

Bemerkungen

Der Verfügbarkeits-Timeout ist die Zeit, die der Datenbankserver einer entfernten Datenbank erlaubt, inaktiv zu sein. Wenn die entfernte Datenbank 1 Sekunde lang nicht mit dem Server kommuniziert, nimmt der Server an, dass die entfernte Datenbank die Verbindung verloren hat, und beendet die Dateiübertragung. Die entfernte Datenbank sendet automatisch periodische Meldungen an den Datenbankserver, um die Verbindung aufrecht zu halten.

Wenn ein negativer Wert gesetzt wird, wird eine Ausnahmebedingung ausgegeben. Der Wert kann vom MobiLink-Server ohne Ankündigung geändert werden. Diese Änderung tritt auf, wenn der Wert zu hoch oder zu niedrig eingestellt ist.

Der Standardwert ist 100 Sekunden für BlackBerry-/J2SE-Plattformen und 240 Sekunden für Android-Plattformen.

Siehe auch

- [FileTransfer.getLivenessTimeout-Methode \[UltraLiteJ\] auf Seite 164](#)

setLocalFileName-Methode

Gibt den Namen der lokalen Datei an.

Syntax

```
abstract void FileTransfer.setLocalFileName(String localFileName)
```

Parameter

- **localFileName** Eine Zeichenfolge, die den lokalen Dateinamen für die heruntergeladene Datei festlegt. Wenn der Wert eine Nullreferenz ist, wird fileName verwendet. Der Standardwert ist eine Nullreferenz.

Bemerkungen

Bei Datei-Downloads ist dies der Name der heruntergeladenen Datei. Bei Datei-Uploads ist dies der Name der hochzuladenden Datei. Der Dateiname darf keine Laufwerks- oder Pfadinformationen enthalten.

Siehe auch

- [FileTransfer.getLocalFileName-Methode \[UltraLiteJ\] auf Seite 164](#)
- [FileTransfer.setLocalPath-Methode \[UltraLiteJ\] auf Seite 170](#)

setLocalPath-Methode

Gibt an, wo die Datei im lokalen Dateisystem gesucht bzw. gespeichert werden soll.

Syntax

```
abstract void FileTransfer.setLocalPath(String localPath)
```

Parameter

- **localPath** Eine Zeichenfolge, die das lokale Verzeichnis der Datei angibt. Der Standardwert ist eine Nullreferenz.

Bemerkungen

Die Syntax für das lokale Verzeichnis variiert zwischen Plattformen:

- Für einen Desktopcomputer lautet die Syntax "C:\\ulj\\".
- Für ein BlackBerry-Dateisystem lautet die Syntax "file:///SDCard/ulj/".
- Bei einem BlackBerry-Objektspeicher wird diese Option ignoriert.
- Für ein Android-Dateisystem lautet die Syntax "/sdcard/Android/data/your.package.name/files/".

Außerdem variiert das lokale Standardverzeichnis je nach Betriebssystem des Geräts:

- Wenn auf einem Desktopcomputer der localPath-Parameter NULL ist, wird die Datei im aktuellen Verzeichnis gespeichert.
- Für einen BlackBerry-Dateisystemspeicher hat der localPath-Parameter keinen Standardwert und muss explizit festgelegt werden.
- Für einen Android-Dateisystemspeicher hat der localPath-Parameter keinen Standardwert und muss explizit festgelegt werden.

Siehe auch

- [FileTransfer.getLocalPath-Methode \[UltraLiteJ\] auf Seite 164](#)
- [FileTransfer.setLocalFileName-Methode \[UltraLiteJ\] auf Seite 169](#)

setPassword-Methode

Setzt das MobiLink-Kennwort für den mit der setUsername-Methode angegebenen Benutzer.

Syntax

```
abstract void FileTransfer.setPassword(  
    String password  
) throws ULjException
```

Parameter

- **password** Ein Kennwort für den MobiLink-Benutzer.

Bemerkungen

Dieser Benutzername und das Kennwort unterscheiden sich von anderen Datenbankbenutzer-IDs und Kennwörtern. Die Methode wird verwendet, um die Anwendung gegenüber dem MobiLink-Server zu authentifizieren.

Der Standardwert ist eine leere Zeichenfolge, womit kein Kennwort angegeben wird.

Siehe auch

- [FileTransfer.getPassword-Methode \[UltraLiteJ\] auf Seite 165](#)
- [FileTransfer.setUserName-Methode \[UltraLiteJ\] auf Seite 173](#)

setRemoteKey-Methode

Gibt den entfernten Schlüssel an.

Syntax

```
abstract void FileTransfer.setRemoteKey(String remoteKey)
```

Parameter

- **remoteKey** Die entfernte Schlüsselwert oder NULL, um keinen Wert anzugeben.

Bemerkungen

Der entfernte Schlüssel ist ein Parameter, der an das Serverskript `authenticate_file_upload` übergeben wird.

Das Skript kann diesen Parameter verwenden, um den Namen und den Standort der auf dem Server zu speichernden Datei zu ermitteln.

Wenn dieser Wert nicht angegeben ist, wird der `getFileName()`-Wert als entfernter Schlüssel verwendet.

Siehe auch

- [FileTransfer.getRemoteKey-Methode \[UltraLiteJ\] auf Seite 165](#)

setResumePartialTransfer-Methode

Legt fest, ob eine frühere teilweise Übertragung wieder aufgenommen oder verworfen werden soll.

Syntax

```
abstract void FileTransfer.setResumePartialTransfer(boolean resume)
```

Parameter

- **resume** Setzen Sie dies auf TRUE, wenn ein früherer Teil-Download wieder aufgenommen werden soll, oder auf FALSE, wenn er verworfen werden soll.

Bemerkungen

Der Standardwert ist TRUE.

UltraLiteJ kann Dateiübertragungen, die aufgrund von Kommunikationsfehlern oder eines Abbruchs durch den Benutzer fehlgeschlagen sind, mit dem FileTransferProgressListener-Objekt neu starten.

Bei Datei-Downloads verarbeitet UltraLiteJ den Download so, wie er empfangen wird. Wenn ein Download unterbrochen wird, bleibt die Teil-Download-Datei erhalten und kann bei der nächsten Dateiübertragung wieder aufgenommen werden. Wenn die Datei auf dem Server aktualisiert wurde, wird der Teil-Download verworfen und ein neuer Download gestartet.

Bei Datei-Uploads speichert der MobiLink-Server Teil-Upload-Dateien, damit ein nachfolgender Datei-Upload einen früheren wieder aufnehmen kann. Wenn die Datei jedoch auf dem Server aktualisiert wurde, wird der Teil-Upload verworfen und ein neuer Upload gestartet.

Siehe auch

- [FileTransfer.isResumePartialTransfer-Methode \[UltraLiteJ\] auf Seite 167](#)

setServerFileName-Methode

Gibt den Namen der Datei auf dem Server an.

Syntax

```
abstract void FileTransfer.setServerFileName(  
    String fileName  
) throws ULjException
```

Parameter

- **fileName** Eine Zeichenfolge, die den Namen der Datei zurückgibt, der vom MobiLink-Server erkannt wurde.

Bemerkungen

Bei Datei-Downloads ist dies der Name der herunterzuladenden Datei. Bei Datei-Uploads ist dies der Name der hochgeladenen Datei.

Dieser Parameter wird initialisiert, wenn das FileTransfer-Objekt erstellt wird.

MobiLink sucht zunächst im Unterverzeichnis "userName" nach der Datei, gefolgt vom Stammverzeichnis. Das Download-Stammverzeichnis wird mit der MobiLink-Serveroption -ftr angegeben, und Upload-Stammverzeichnis mit der Option -ftru.

Der fileName-Wert darf keine Laufwerks- oder Pfadinformationen enthalten. Andernfalls wird die Datei vom MobiLink-Server nicht gefunden. Beispiel: "myfile.txt" ist gültig, doch "somedir\myfile.txt", "..\myfile.txt" und "c:\myfile.txt" sind jeweils ungültig.

Siehe auch

- [FileTransfer.getServerFileName-Methode \[UltraLiteJ\] auf Seite 165](#)

setUserName-Methode

Setzt den MobiLink-Benutzernamen, der den Client für den MobiLink-Server eindeutig kennzeichnet.

Syntax

```
abstract void FileTransfer.setUserName(  
    String userName  
) throws ULjException
```

Parameter

- **userName** Der MobiLink-Benutzername

Bemerkungen

Der MobiLink-Server verwendet diesen Wert, um die Datei auf der Serverseite zu finden. Der MobiLink-Benutzername und das Kennwort unterscheiden sich von der ID und dem Kennwort des Datenbankbenutzers und dienen der Kennzeichnung und Authentifizierung der Anwendung beim MobiLink-Server.

Dieser Parameter wird initialisiert, wenn das FileTransfer-Objekt erstellt wird.

Siehe auch

- [FileTransfer.getUserName-Methode \[UltraLiteJ\] auf Seite 167](#)
- [FileTransfer.setPassword-Methode \[UltraLiteJ\] auf Seite 170](#)

setVersion-Methode

Setzt das zu verwendende Synchronisationsskript.

Syntax

```
abstract void FileTransfer.setVersion(  
    String version  
) throws ULjException
```

Parameter

- **version** Die Skriptversion

Bemerkungen

Jedes Synchronisationsskript in der konsolidierten Datenbank wird mit einer Versionszeichenfolge markiert. Die Versionszeichenfolge gestattet es einer UltraLiteJ-Anwendung, zwischen einer Reihe von Synchronisationsskripten zu wählen.

Dieser Parameter wird initialisiert, wenn das FileTransfer-Objekt erstellt wird.

Siehe auch

- [FileTransfer.getVersion-Methode \[UltraLiteJ\] auf Seite 167](#)

uploadFile-Methode

Lädt die Datei mit den angegebenen Eigenschaften dieses Objekts hoch.

Überladungsliste

Name	Beschreibung
uploadFile()-Methode	Lädt die Datei mit den angegebenen Eigenschaften dieses Objekts hoch.
uploadFile(FileTransferProgressListener)-Methode	Lädt die Datei hoch, die durch die Eigenschaften dieser Datei festgelegt wird, wobei an den angegebenen Listener Fortschrittsereignisse gesendet werden.

uploadFile()-Methode

Lädt die Datei mit den angegebenen Eigenschaften dieses Objekts hoch.

Syntax

```
abstract boolean FileTransfer.uploadFile() throws ULjException
```

Rückgabe

TRUE, wenn der Upload erfolgreich ist. Andernfalls wird ein ULjException-Objekt erzeugt und die Methode liefert nicht die normale Rückgabe.

Bemerkungen

Die durch die Methoden setLocalFileName und setLocalPath angegebene Datei wird vom MobiLink-Server in die durch die setServerFileName-Methode angegebene Datei hochgeladen. Dabei werden die angegebenen Werte für Datenstrom, Benutzername, Kennwort und Skriptversion verwendet.

Weitere Optionen können mithilfe der Methoden setAuthenticationParms() und setResumePartialTransfer() angegeben werden.

Ein ausführlicher Ergebnisstatus kann mithilfe der Methoden getAuthStatus(), getAuthValue(), getFileAuthCode(), isTransferredFile(), getStreamErrorCode() und getStreamErrorMessage() abgerufen werden.

uploadFile(FileTransferProgressListener)-Methode

Lädt die Datei hoch, die durch die Eigenschaften dieser Datei festgelegt wird, wobei an den angegebenen Listener Fortschrittsereignisse gesendet werden.

Syntax

```
abstract boolean FileTransfer.uploadFile(  
    FileTransferProgressListener listener  
) throws ULjException
```

Parameter

- **listener** Das Objekt, das Ereignisse zum Dateiübertragungsfortschritt empfängt.

Rückgabe

TRUE, wenn der Upload erfolgreich ist. Andernfalls wird ein **ULjException**-Objekt erzeugt und die Methode liefert nicht die normale Rückgabe.

Bemerkungen

Fehler können dazu führen, dass keine Daten an den Listener gesendet werden.

Siehe auch

- [FileTransfer.uploadFile-Methode \[UltraLiteJ\] auf Seite 174](#)

FileTransferProgressData-Schnittstelle

Gibt Überwachungsdaten zum Dateiübertragungsfortschritt zurück.

Syntax

```
public interface FileTransferProgressData
```

Mitglieder

Alle Mitglieder der FileTransferProgressData-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getBytesTransferred-Methode	Gibt die Anzahl der bislang übertragenen Bytes zurück.
getFileSize-Methode	Gibt die Größe der Datei zurück, die übertragen wird.
getResumedAtSize-Methode	Gibt die Position in der Datei zurück, an der die Übetragung wieder aufgenommen wurde.

getBytesTransferred-Methode

Gibt die Anzahl der bislang übertragenen Bytes zurück.

Syntax

```
abstract long FileTransferProgressData.getBytesTransferred( )
```

Rückgabe

Die Anzahl der bislang übertragenen Bytes.

Bemerkungen

Diese Methode zählt die von der aktuellen Dateiübertragungssitzung übertragenen Byte und addiert dazu die übertragenen Byte aus vorherigen unterbrochenen Übertragungen.

Subtrahieren Sie den von der `getResumedAtSize`-Methode zurückgegebenen Wert, um zu ermitteln, wie viele Byte von der aktuellen Sitzung übertragen wurden.

Siehe auch

- [FileTransferProgressData.getResumedAtSize-Methode \[UltraLiteJ\] auf Seite 176](#)

getFileSize-Methode

Gibt die Größe der Datei zurück, die übertragen wird.

Syntax

```
abstract long FileTransferProgressData.getFileSize()
```

Rückgabe

Die Größe der Datei in Byte.

Bemerkungen

Der zurückgegebene Wert bleibt für die Dauer der Dateiübertragungssitzung konstant.

getResumedAtSize-Methode

Gibt die Position in der Datei zurück, an der die Übertragung wieder aufgenommen wurde.

Syntax

```
abstract long FileTransferProgressData.getResumedAtSize()
```

Rückgabe

Die zuvor übertragene Byte-Anzahl.

Bemerkungen

Der zurückgegebene Wert bleibt für die Dauer der Dateiübertragungssitzung konstant.

FileTransferProgressListener-Schnittstelle

Empfängt Ereignisse zum Fortschritt der Dateiübertragung.

Syntax

```
public interface FileTransferProgressListener
```

Mitglieder

Alle Mitglieder der FileTransferProgressListener-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
fileTransferProgressed-Methode	Wird während einer Dateiübertragung aufgerufen, um den Benutzer über den Fortschritt der Übertragung zu informieren.

Bemerkungen

Erstellen Sie eine neue Klasse, die während einer Dateiübertragung Berichte zum Verarbeitungsfortschritt empfangen soll.

Das folgende Beispiel zeigt eine einfache SyncObserver-Schnittstelle, die die FileTransferProgressListener-Schnittstelle implementiert:

```
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-Methode

Wird während einer Dateiübertragung aufgerufen, um den Benutzer über den Fortschritt der Übertragung zu informieren.

Syntax

```
boolean FileTransferProgressListener.fileTransferProgressed(  
    FileTransferProgressData data  
)
```

Parameter

- **data** Ein FileTransferProgressData-Objekt, das die neuesten Fortschrittsdaten zur Dateiübertragung enthält.

Rückgabe

Diese Methode muss TRUE zurückgeben, um die Übertragung abubrechen, oder FALSE, um fortzufahren.

Bemerkungen

Der Listener wird unter den folgenden Bedingungen aufgerufen:

- Vor dem ersten Schreibvorgang auf der Festplatte.
- Nach jedem Schreibvorgang auf der Festplatte oder alle 0,5 Sekunden, je nachdem, welches der spätere Zeitpunkt ist.
- Nachdem der Datei-Download abgeschlossen ist.

Normalerweise wird die Abbruchanforderung von der UltraLiteJ-API akzeptiert. Dies führt dazu, dass ein `ULjException`-Objekt ausgegeben wird, in dem der `errorCode`-Parameter auf die `ULjException.SQLE_INTERRUPTED`-Konstante gesetzt ist.

Wenn jedoch der Download für einen BlackBerry-Objektspeicher abgeschlossen wurde, bricht UltraLiteJ die Übertragung nicht ab.

Während eines `fileTransferProgressed`-Aufrufs dürfen keine Methoden der UltraLiteJ-API aufgerufen werden.

IndexSchema-Schnittstelle

Gibt das Schema für einen Index an und stellt Konstanten bereit, die für das Abfragen von Systemtabellen nützlich sind.

Syntax

```
public interface IndexSchema
```

Mitglieder

Alle Mitglieder der IndexSchema-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
ASCENDING-Variable	Gibt an, dass der Index für eine Spalte in aufsteigender Reihenfolge sortiert wird.
DESCENDING-Variable	Gibt an, dass der Index für eine Spalte in absteigender Reihenfolge sortiert wird.
PERSISTENT-Variable	Gibt an, dass ein Index beständig ist.
PRIMARY_INDEX-Variable	Gibt an, dass ein Index ein Primärschlüssel ist.
UNIQUE_INDEX-Variable	Gibt an, dass ein Index ein eindeutiger Index ist.
UNIQUE_KEY-Variable	Gibt an, dass ein Index ein eindeutiger Schlüssel ist.

Bemerkungen

Diese Schnittstelle enthält nur indexbezogene Konstanten, einschließlich Parametern und der Sortierreihenfolge von Indizes.

Siehe auch

- [TableSchema.SYS_INDEXES-Variable \[UltraLiteJ\] auf Seite 283](#)

ASCENDING-Variable

Gibt an, dass der Index für eine Spalte in aufsteigender Reihenfolge sortiert wird.

Syntax

```
final byte IndexSchema.ASCENDING
```

DESCENDING-Variable

Gibt an, dass der Index für eine Spalte in absteigender Reihenfolge sortiert wird.

Syntax

```
final byte IndexSchema.DESCENDING
```

PERSISTENT-Variable

Gibt an, dass ein Index beständig ist.

Syntax

```
final byte IndexSchema.PERSISTENT
```

Bemerkungen

Dieser Wert kann logisch mit anderen Parametern in der index_flags-Spalte der SYS_TABLES-Systemtabelle kombiniert werden.

Siehe auch

- [TableSchema.SYS_INDEXES-Variable \[UltraLiteJ\] auf Seite 283](#)

PRIMARY_INDEX-Variable

Gibt an, dass ein Index ein Primärschlüssel ist.

Syntax

```
final byte IndexSchema.PRIMARY_INDEX
```

Bemerkungen

Dieser Wert kann logisch mit anderen Parametern in der index_flags-Spalte der SYS_TABLES-Systemtabelle kombiniert werden.

Siehe auch

- [TableSchema.SYS_INDEXES-Variable \[UltraLiteJ\] auf Seite 283](#)

UNIQUE_INDEX-Variable

Gibt an, dass ein Index ein eindeutiger Index ist.

Syntax

```
final byte IndexSchema.UNIQUE_INDEX
```

Bemerkungen

Dieser Wert kann logisch mit anderen Parametern in der index_flags-Spalte der SYS_TABLES-Systemtabelle kombiniert werden.

Siehe auch

- [TableSchema.SYS_INDEXES-Variable \[UltraLiteJ\] auf Seite 283](#)

UNIQUE_KEY-Variable

Gibt an, dass ein Index ein eindeutiger Schlüssel ist.

Syntax

```
final byte IndexSchema.UNIQUE_KEY
```

Bemerkungen

Dieser Wert kann logisch mit anderen Parametern in der index_flags-Spalte der SYS_TABLES-Systemtabelle kombiniert werden.

Siehe auch

- [TableSchema.SYS_INDEXES-Variable \[UltraLiteJ\] auf Seite 283](#)

PreparedStatement-Schnittstelle

Stellt Methoden bereit, um eine SQL-Abfrage zum Generieren eines ResultSet-Objekts auszuführen oder um eine vorbereitete SQL-Anweisung in einer Datenbank auszuführen.

Syntax

```
public interface PreparedStatement
```

Mitglieder

Alle Mitglieder der PreparedStatement-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
close-Methode	Schließt das PreparedStatement, um die ihm zugeordneten Speicherressourcen freizugeben.
execute-Methode	Führt die vorbereitete SQL-Anweisung aus.
executeQuery-Methode	Führt die vorbereitete SQL SELECT-Anweisung aus und gibt ein ResultSet-Objekt zurück.
getBlobOutputStream-Methode	Gibt ein OutputStream-Objekt zurück.
getClobWriter-Methode	Gibt ein Writer-Objekt zurück.
getOrdinal-Methode	Gibt die Ordinalzahl (mit Basis Eins) für den Wert zurück, der vom Namen dargestellt wird.
getParameterCount-Methode [Android]	Ruft die Anzahl der Eingabeparameter für diese Anweisung ab.
getParameterType-Methode [Android]	Ruft den Domänentyp einer Spalte ab.
getPlan-Methode	Gibt eine textbasierte Beschreibung des SQL-Abfrageausführungsplans zurück.
getPlanTree-Methode	Gibt eine textbasierte Beschreibung des SQL-Abfrageausführungsplans zurück, dargestellt als Baumstruktur.
getResultSet-Methode	Gibt das ResultSet-Objekt für eine vorbereitete SQL-Anweisung zurück.
getUpdateCount-Methode	Gibt die Anzahl der Zeilen zurück, die seit der letzten execute-Anweisung eingefügt, aktualisiert oder gelöscht wurden.
hasResultSet-Methode	Ermittelt, ob das PreparedStatement-Objekt ein ResultSet-Objekt enthält.
set-Methode	Setzt einen Wert für die Hostvariable in der SQL-Anweisung.
setNull-Methode	Setzt einen NULL-Wert für die Hostvariable in der SQL-Anweisung.

Bemerkungen

Das folgende Beispiel zeigt, wie Sie ein PreparedStatement-Objekt erstellen, prüfen, ob beim Ausführen einer SELECT-Anweisung ein ResultSet-Objekt erstellt wird, alle ResultSet-Objekte in einer lokalen Variablen speichern und danach das PreparedStatement-Objekt schließen:

```
// 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();
```

Wenn eine Anweisung Ausdrücke enthält, enthält sie möglicherweise eine Hostvariable, in der ein Spaltenname auftreten kann. Hostvariablen werden entweder als ? Zeichen (unbenannte Hostvariablen) oder als :name (benannte Hostvariablen) eingegeben.

Im folgenden Beispiel gibt es zwei Hostvariablen, die mithilfe des PreparedStatement-Objekts festgelegt werden können und die für die betreffende SQL-Anweisung vorbereitet wurden:

```
SELECT * FROM SampleTable WHERE pk > :bound AND pk < ?
```

Siehe auch

- [Connection-Schnittstelle \[UltraLiteJ\] auf Seite 105](#)
- [Connection.prepareStatement-Methode \[UltraLiteJ\] auf Seite 118](#)

close-Methode

Schließt das PreparedStatement, um die ihm zugeordneten Speicherressourcen freizugeben.

Syntax

```
void PreparedStatement.close() throws ULjException
```

Bemerkungen

Es können keine weiteren Methoden an diesem Objekt angewendet werden. Wenn das PreparedStatement-Objekt ein ResultSet-Objekt enthält, werden beide Objekte geschlossen.

execute-Methode

Führt die vorbereitete SQL-Anweisung aus.

Syntax

```
boolean PreparedStatement.execute() throws ULjException
```

Rückgabe

TRUE, wenn die execute-Anweisung erfolgreich ausgeführt wird, ansonsten FALSE.

Siehe auch

- [ResultSet-Schnittstelle \[UltraLiteJ\] auf Seite 197](#)

executeQuery-Methode

Führt die vorbereitete SQL SELECT-Anweisung aus und gibt ein ResultSet-Objekt zurück.

Syntax

```
ResultSet PreparedStatement.executeQuery() throws ULjException
```

Rückgabe

Das ResultSet-Objekt, das das Abfrageergebnis der vorbereiteten SQL SELECT-Anweisung enthält.

Siehe auch

- [ResultSet-Schnittstelle \[UltraLiteJ\] auf Seite 197](#)

getBlobOutputStream-Methode

Gibt ein OutputStream-Objekt zurück.

Überladungsliste

Name	Beschreibung
getBlobOutputStream(int)-Methode	Gibt ein OutputStream-Objekt zurück.
getBlobOutputStream(String)-Methode	Gibt ein OutputStream-Objekt zurück.

getBlobOutputStream(int)-Methode

Gibt ein OutputStream-Objekt zurück.

Syntax

```
java.io.OutputStream PreparedStatement.getBlobOutputStream(  
    int ordinal  
) throws ULjException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Das OutputStream-Objekt für den benannten Wert.

getBlobOutputStream(String)-Methode

Gibt ein OutputStream-Objekt zurück.

Syntax

```
java.io.OutputStream PreparedStatement.getBlobOutputStream(  
    String name  
) throws ULJException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.

Rückgabe

Das OutputStream-Objekt für den benannten Wert.

getClobWriter-Methode

Gibt ein Writer-Objekt zurück.

Überladungsliste

Name	Beschreibung
getClobWriter(int)-Methode	Gibt ein Writer-Objekt zurück.
getClobWriter(String)-Methode	Gibt ein Writer-Objekt zurück.

getClobWriter(int)-Methode

Gibt ein Writer-Objekt zurück.

Syntax

```
java.io.Writer PreparedStatement.getClobWriter(  
    int ordinal  
) throws ULJException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Das Writer-Objekt für den benannten Wert.

getClobWriter(String)-Methode

Gibt ein Writer-Objekt zurück.

Syntax

```
java.io.Writer PreparedStatement.getClobWriter(  
    String name  
) throws ULJException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.

Rückgabe

Das Writer-Objekt für den benannten Wert.

getOrdinal-Methode

Gibt die Ordinalzahl (mit Basis Eins) für den Wert zurück, der vom Namen dargestellt wird.

Syntax

```
int PreparedStatement.getOrdinal(String name) throws ULJException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die Ordinalzahl (mit Basis Eins) für den Wert, der vom Namen dargestellt wird.

getParameterCount-Methode [Android]

Ruft die Anzahl der Eingabeparameter für diese Anweisung ab.

Syntax

```
short PreparedStatement.getParameterCount() throws ULJException
```

Rückgabe

Die Anzahl der Eingabeparameter für diese Anweisung.

getParameterType-Methode [Android]

Ruft den Domänentyp einer Spalte ab.

Syntax

```
short PreparedStatement.getParameterType(  
    int ordinal  
) throws ULjException
```

Parameter

- **ordinal** Die 1-basierte Ordinalzahl des Parameters

Rückgabe

Der Domänentyp des angegebenen Parameters.

getPlan-Methode

Gibt eine textbasierte Beschreibung des SQL-Abfrageausführungsplans zurück.

Syntax

```
String PreparedStatement.getPlan() throws ULjException
```

Rückgabe

Die Zeichenfolgendarstellung des Plans.

Bemerkungen

Diese Methode ist für die Entwicklung gedacht.

Dieser Plan enthält dieselben Informationen, wie sie von der `getPlanTree`-Methode dargestellt werden. Der Unterschied liegt in der Darstellung.

Eine leere Zeichenfolge wird zurückgegeben, wenn es keinen Plan gibt. Pläne sind vorhanden, wenn die vorbereitete Anweisung eine SQL-Abfrage ist.

Der Plan zeigt die Vorgänge, die zum Ausführen der Abfrage verwendet werden, wenn der Plan abgerufen wird, bevor die dazugehörige Abfrage ausgeführt wurde. Außerdem zeigt der Plan die Anzahl der Zeilen, die jeder Vorgang erzeugt, wenn der Plan abgerufen wird, nachdem die dazugehörige Abfrage ausgeführt wurde. Dieser Plan kann verwendet werden, um mehr Informationen über die Ausführung der Abfrage zu gewinnen.

Das Folgende ist ein Beispiel für eine Planstruktur, angegeben als Zeichenfolge. Sie wird auf mehreren Zeilen angezeigt, wobei das Zeichen '|' für die Darstellung der Struktur verwendet wird.

```
SELECT * FROM tab1, tab2 WHERE coll > 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]))
```

Siehe auch

- [PreparedStatement.getPlanTree-Methode \[UltraLiteJ\] auf Seite 187](#)

getPlanTree-Methode

Gibt eine textbasierte Beschreibung des SQL-Abfrageausführungsplans zurück, dargestellt als Baumstruktur.

Syntax

```
String PreparedStatement.getPlanTree() throws ULjException
```

Rückgabe

Die Zeichenfolgendarstellung des Plans, dargestellt als Baumstruktur.

Bemerkungen

Diese Methode ist für die Entwicklung gedacht.

Dieser Plan enthält dieselben Informationen, wie sie von der getPlan-Methode dargestellt werden. Der Unterschied liegt in der Darstellung.

Eine leere Zeichenfolge wird zurückgegeben, wenn es keinen Plan gibt. Pläne sind vorhanden, wenn die vorbereitete Anweisung eine SQL-Abfrage ist.

Der Plan zeigt die Vorgänge, die zum Ausführen der Abfrage verwendet werden, wenn der Plan abgerufen wird, bevor die dazugehörige Abfrage ausgeführt wurde. Außerdem zeigt der Plan die Anzahl der Zeilen, die jeder Vorgang erzeugt, wenn der Plan abgerufen wird, nachdem die dazugehörige Abfrage ausgeführt wurde. Dieser Plan kann verwendet werden, um mehr Informationen über die Ausführung der Abfrage zu gewinnen.

Das Folgende ist ein Beispiel für eine Planstruktur, angegeben als Zeichenfolge. Sie wird auf mehreren Zeilen angezeigt, wobei das Zeichen '|' für die Darstellung der Struktur verwendet wird.

```
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]
```

Siehe auch

- [PreparedStatement.getPlan-Methode \[UltraLiteJ\] auf Seite 186](#)

getResultSet-Methode

Gibt das ResultSet-Objekt für eine vorbereitete SQL-Anweisung zurück.

Syntax

```
ResultSet PreparedStatement.getResultSet() throws ULjException
```

Rückgabe

Das ResultSet-Objekt, das das Abfrageergebnis der vorbereiteten SQL-Anweisung enthält.

Siehe auch

- [ResultSet-Schnittstelle \[UltraLiteJ\] auf Seite 197](#)

getUpdateCount-Methode

Gibt die Anzahl der Zeilen zurück, die seit der letzten execute-Anweisung eingefügt, aktualisiert oder gelöscht wurden.

Syntax

```
int PreparedStatement.getUpdateCount() throws ULjException
```

Rückgabe

-1, wenn eine Anweisung keine Änderungen ausführen kann, andernfalls die Anzahl der geänderten Zeilen.

hasResultSet-Methode

Ermittelt, ob das PreparedStatement-Objekt ein ResultSet-Objekt enthält.

Syntax

```
boolean PreparedStatement.hasResultSet() throws ULjException
```

Rückgabe

TRUE, wenn ein ResultSet-Objekt gefunden wurde, ansonsten FALSE.

Siehe auch

- [ResultSet-Schnittstelle \[UltraLiteJ\] auf Seite 197](#)

set-Methode

Setzt einen Wert für die Hostvariable in der SQL-Anweisung.

Überladungsliste

Name	Beschreibung
set(int, boolean)-Methode	Setzt einen booleschen Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.
set(int, byte[])-Methode	Setzt einen Byte-Array-Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.
set(int, Date)-Methode	Setzt einen java.util.Date-Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.
set(int, DecimalNumber)-Methode	Setzt ein DecimalNumber-Objekt auf die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.
set(int, double)-Methode	Setzt einen double-Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.
set(int, float)-Methode	Setzt einen float-Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.
set(int, int)-Methode	Setzt einen Ganzzahlwert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.
set(int, long)-Methode	Setzt einen Ganzzahlwert (long integer) für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.
set(int, string)-Methode	Setzt einen Zeichenfolgenwert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.
set(int, UUIDValue)-Methode	Setzt einen UUID-Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.
set(string, boolean)-Methode	Setzt einen booleschen Wert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.
set(string, byte[])-Methode	Setzt einen Byte-Array-Wert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.
set(string, date)-Methode	Setzt ein java.util.Date-Objekt auf die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.
set(string, DecimalNumber)-Methode	Setzt ein DecimalNumber-Objekt auf die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Name	Beschreibung
set(string, double)-Methode	Setzt einen double-Wert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.
set(string, float)-Methode	Setzt einen float-Wert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.
set(string, int)-Methode	Setzt einen Ganzzahlwert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.
set(string, long)-Methode	Setzt einen Ganzzahlwert (long integer) für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.
set(string, string)-Methode	Setzt einen Zeichenfolgenwert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.
set(string, UUIDValue)-Methode	Setzt einen UUID-Wert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

set(int, boolean)-Methode

Setzt einen booleschen Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.

Syntax

```
void PreparedStatement.set(  
    int ordinal,  
    boolean value  
) throws ULjException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.
- **value** Der zu setzende Wert.

set(int, byte[])-Methode

Setzt einen Byte-Array-Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.

Syntax

```
void PreparedStatement.set(  
    int ordinal,  
    byte[] value  
) throws ULjException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.
- **value** Der zu setzende Wert.

set(int, Date)-Methode

Setzt einen java.util.Date-Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.

Syntax

```
void PreparedStatement.set(  
    int ordinal,  
    java.util.Date value  
) throws SQLException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.
- **value** Der zu setzende Wert.

set(int, DecimalNumber)-Methode

Setzt ein DecimalNumber-Objekt auf die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.

Syntax

```
void PreparedStatement.set(  
    int ordinal,  
    DecimalNumber value  
) throws SQLException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.
- **value** Der zu setzende DecimalNumber-Wert.

set(int, double)-Methode

Setzt einen double-Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.

Syntax

```
void PreparedStatement.set(  
    int ordinal,  
    double value  
) throws ULJException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.
- **value** Der zu setzende Wert.

set(int, float)-Methode

Setzt einen float-Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.

Syntax

```
void PreparedStatement.set(int ordinal, float value) throws ULJException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.
- **value** Der zu setzende Wert.

set(int, int)-Methode

Setzt einen Ganzzahlwert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.

Syntax

```
void PreparedStatement.set(int ordinal, int value) throws ULJException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.
- **value** Der zu setzende Wert.

set(int, long)-Methode

Setzt einen Ganzzahlwert (long integer) für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.

Syntax

```
void PreparedStatement.set(int ordinal, long value) throws ULJException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.
- **value** Der zu setzende Wert.

set(int, string)-Methode

Setzt einen Zeichenfolgenwert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.

Syntax

```
void PreparedStatement.set(  
    int ordinal,  
    String value  
) throws SQLException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.
- **value** Der zu setzende Wert.

set(int, UUIDValue)-Methode

Setzt einen UUID-Wert für die Hostvariable in der SQL-Anweisung, die durch die Ordinalzahl definiert ist.

Syntax

```
void PreparedStatement.set(  
    int ordinal,  
    UUIDValue value  
) throws SQLException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.
- **value** Der zu setzende Wert.

set(string, boolean)-Methode

Setzt einen booleschen Wert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Syntax

```
void PreparedStatement.set(  
    String name,  
    boolean value  
) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.
- **value** Der zu setzende Wert.

set(string, byte[])-Methode

Setzt einen Byte-Array-Wert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Syntax

```
void PreparedStatement.set(  
    String name,  
    byte[] value  
) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.
- **value** Der zu setzende Wert.

set(string, date)-Methode

Setzt ein java.util.Date-Objekt auf die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Syntax

```
void PreparedStatement.set(  
    String name,  
    java.util.Date value  
) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.
- **value** Der zu setzende Wert.

set(string, DecimalNumber)-Methode

Setzt ein DecimalNumber-Objekt auf die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Syntax

```
void PreparedStatement.set(  
    String name,  
    DecimalNumber value  
) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.
- **value** Der zu setzende DecimalNumber-Wert.

set(string, double)-Methode

Setzt einen double-Wert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Syntax

```
void PreparedStatement.set(  
    String name,  
    double value  
) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.
- **value** Der zu setzende Wert.

set(string, float)-Methode

Setzt einen float-Wert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Syntax

```
void PreparedStatement.set(String name, float value) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.
- **value** Der zu setzende Wert.

set(string, int)-Methode

Setzt einen Ganzzahlwert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Syntax

```
void PreparedStatement.set(String name, int value) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.
- **value** Der zu setzende Wert.

set(string, long)-Methode

Setzt einen Ganzzahlwert (long integer) für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Syntax

```
void PreparedStatement.set(String name, long value) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.
- **value** Der zu setzende Wert.

set(string, string)-Methode

Setzt einen Zeichenfolgenwert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Syntax

```
void PreparedStatement.set(  
    String name,  
    String value  
) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.
- **value** Der zu setzende Wert.

set(string, UUIDValue)-Methode

Setzt einen UUID-Wert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Syntax

```
void PreparedStatement.set(  
    String name,  
    UUIDValue value  
) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.

- **value** Der zu setzende Wert.

setNull-Methode

Setzt einen NULL-Wert für die Hostvariable in der SQL-Anweisung.

Überladungsliste

Name	Beschreibung
setNull(int)-Methode	Setzt einen NULL-Wert für die Hostvariable in der SQL-Anweisung.
setNull(string)-Methode	Setzt einen Nullwert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

setNull(int)-Methode

Setzt einen NULL-Wert für die Hostvariable in der SQL-Anweisung.

Syntax

```
void PreparedStatement.setNull(int ordinal) throws SQLException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Hostvariable darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

setNull(string)-Methode

Setzt einen Nullwert für die Hostvariable in der SQL-Anweisung, die durch den Namen definiert ist.

Syntax

```
void PreparedStatement.setNull(String name) throws SQLException
```

Parameter

- **name** Eine Zeichenfolge, die den Namen der Hostvariable darstellt.

ResultSet-Schnittstelle

Stellt Methoden bereit, um eine Tabelle zeilenweise zu durchlaufen und auf die Spaltendaten zuzugreifen.

Syntax

```
public interface ResultSet
```

Mitglieder

Alle Mitglieder der ResultSet-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
afterLast-Methode [Android]	Verschiebt den Cursor hinter die letzte Zeile.
beforeFirst-Methode [Android]	Verschiebt den Cursor vor die erste Zeile.
close-Methode	Schließt das ResultSet-Objekt, um die ihm zugeordneten Speicherressourcen freizugeben.
first-Methode [Android]	Verschiebt den Cursor in die erste Zeile.
getBlobInputStream-Methode	Gibt für lange Binärdatentypen, einschließlich dateibasierter langer Binärdaten, ein InputStream-Objekt zurück.
getBoolean-Methode	Gibt einen booleschen Wert zurück.
getBytes-Methode	Gibt ein Byte-Array zurück.
getClobReader-Methode	Gibt ein Reader-Objekt zurück.
getDate-Methode	Gibt ein java.util.Date-Objekt zurück.
getDecimalNumber-Methode	Gibt ein DecimalNumber-Objekt zurück.
getDouble-Methode	Gibt einen double-Wert zurück, basierend auf der Spaltennummer.
getFloat-Methode	Gibt einen float-Wert zurück.
getInt-Methode	Gibt einen Ganzzahlwert zurück.
getLong-Methode	Gibt einen Ganzzahlwert (long integer) zurück.
getOrdinal-Methode	Gibt die Ordinalzahl (mit Basis Eins) für den Wert zurück, der von einer Zeichenfolge dargestellt wird.
getResultSetMetadata-Methode	Gibt ein ResultSetMetadata-Objekt zurück, das die Metadaten für das ResultSet-Objekt enthält.
getRowCount-Methode [Android]	Ruft die Anzahl der Zeilen in der Tabelle ab.
getSize-Methode	Ruft die tatsächliche Größe einer Ergebnismengenspalte ab.
getString-Methode	Gibt einen Zeichenfolgenwert zurück.

Name	Beschreibung
getUUIDValue-Methode	Gibt ein UUIDValue-Objekt zurück.
isNull-Methode	Testet, ob der Wert an der Stelle der angegebenen Spaltennummer NULL ist.
last-Methode [Android]	Verschiebt den Cursor in die letzte Zeile.
next-Methode	Ruft die nächste Datenzeile im ResultSet-Objekt ab.
previous-Methode	Ruft die vorherige Datenzeile im ResultSet-Objekt ab.
relative-Methode [Android]	Verschiebt den Cursor von der aktuellen Cursorposition ausgehend um die durch den Offset angegebene Anzahl von Zeilen.

Bemerkungen

Ein ResultSet-Objekt wird generiert, wenn die execute- oder executeQuery-Methode für ein PreparedStatement-Objekt mit einer SQL SELECT-Anweisung aufgerufen wird.

Das folgende Beispiel zeigt, wie Sie eine Zeile mit dem ResultSet-Objekt abrufen und auf Daten in einer angegebenen Spalte zugreifen:

```
// 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();
```

Siehe auch

- [PreparedStatement-Schnittstelle \[UltraLiteJ\] auf Seite 180](#)
- [PreparedStatement.execute-Methode \[UltraLiteJ\] auf Seite 182](#)
- [PreparedStatement.executeQuery-Methode \[UltraLiteJ\] auf Seite 183](#)
- [Connection-Schnittstelle \[UltraLiteJ\] auf Seite 105](#)

afterLast-Methode [Android]

Verschiebt den Cursor hinter die letzte Zeile.

Syntax

```
boolean ResultSet.afterLast() throws ULjException
```

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

beforeFirst-Methode [Android]

Verschiebt den Cursor vor die erste Zeile.

Syntax

```
boolean ResultSet.beforeFirst() throws ULjException
```

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

close-Methode

Schließt das ResultSet-Objekt, um die ihm zugeordneten Speicherressourcen freizugeben.

Syntax

```
void ResultSet.close() throws ULjException
```

Bemerkungen

Bei nachfolgenden Versuchen, Zeilen aus einem geschlossenen ResultSet-Objekt abzurufen, wird einen Fehler ausgegeben.

first-Methode [Android]

Verschiebt den Cursor in die erste Zeile.

Syntax

```
boolean ResultSet.first() throws ULjException
```

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

getBlobInputStream-Methode

Gibt für lange Binärdatentypen, einschließlich dateibasierter langer Binärdaten, ein InputStream-Objekt zurück.

Überladungsliste

Name	Beschreibung
getBlobInputStream(int)-Methode	Gibt für lange Binärdatentypen, einschließlich dateibasierter langer Binärdaten, ein InputStream-Objekt zurück.
getBlobInputStream(String)-Methode	Gibt für lange Binärdatentypen, einschließlich dateibasierter langer Binärdaten, ein InputStream-Objekt zurück.

getBlobInputStream(int)-Methode

Gibt für lange Binärdatentypen, einschließlich dateibasierter langer Binärdaten, ein InputStream-Objekt zurück.

Syntax

```
java.io.InputStream ResultSet.getBlobInputStream(
    int ordinal
) throws SQLException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die Darstellung des benannten Werts als InputStream-Objekt.

getBlobInputStream(String)-Methode

Gibt für lange Binärdatentypen, einschließlich dateibasierter langer Binärdaten, ein InputStream-Objekt zurück.

Syntax

```
java.io.InputStream ResultSet.getBlobInputStream(
    String name
) throws SQLException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die Darstellung des benannten Werts als InputStream-Objekt.

getBoolean-Methode

Gibt einen booleschen Wert zurück.

Überladungsliste

Name	Beschreibung
getBoolean(int)-Methode	Gibt einen booleschen Wert zurück.
getBoolean(string)-Methode	Gibt einen booleschen Wert zurück.

getBoolean(int)-Methode

Gibt einen booleschen Wert zurück.

Syntax

```
boolean ResultSet.getBoolean(int ordinal) throws ULjException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die boolesche Darstellung des benannten Werts.

getBoolean(string)-Methode

Gibt einen booleschen Wert zurück.

Syntax

```
boolean ResultSet.getBoolean(String name) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen im ResultSet-Objekt darstellt.

Rückgabe

Die boolesche Darstellung des benannten Werts.

getBytes-Methode

Gibt ein Byte-Array zurück.

Überladungsliste

Name	Beschreibung
getBytes(int)-Methode	Gibt ein Byte-Array zurück.
getBytes(string)-Methode	Gibt ein Byte-Array zurück.

getBytes(int)-Methode

Gibt ein Byte-Array zurück.

Syntax

```
byte[] ResultSet.getBytes(int ordinal) throws SQLException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die Byte-Array-Darstellung des benannten Werts.

getBytes(string)-Methode

Gibt ein Byte-Array zurück.

Syntax

```
byte[] ResultSet.getBytes(String name) throws SQLException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die Byte-Array-Darstellung des benannten Werts.

getClobReader-Methode

Gibt ein Reader-Objekt zurück.

Überladungsliste

Name	Beschreibung
getClobReader(int)-Methode	Gibt ein Reader-Objekt zurück.
getClobReader(string)-Methode	Gibt ein Reader-Objekt zurück.

getClobReader(int)-Methode

Gibt ein Reader-Objekt zurück.

Syntax

```
java.io.Reader ResultSet.getClobReader(int ordinal) throws ULjException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die Darstellung des benannten Werts als Reader-Objekt.

getClobReader(string)-Methode

Gibt ein Reader-Objekt zurück.

Syntax

```
java.io.Reader ResultSet.getClobReader(String name) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die Darstellung des benannten Werts als Reader-Objekt.

getDate-Methode

Gibt ein java.util.Date-Objekt zurück.

Überladungsliste

Name	Beschreibung
getDate(int)-Methode	Gibt ein java.util.Date-Objekt zurück.

Name	Beschreibung
getDate(string)-Methode	Gibt ein java.util.Date-Objekt zurück.

getDate(int)-Methode

Gibt ein java.util.Date-Objekt zurück.

Syntax

```
java.util.Date ResultSet.getDate(int ordinal) throws SQLException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die Darstellung des benannten Werts als java.util.Date-Objekt.

getDate(string)-Methode

Gibt ein java.util.Date-Objekt zurück.

Syntax

```
java.util.Date ResultSet.getDate(String name) throws SQLException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die Darstellung des benannten Werts als java.util.Date-Objekt.

getDecimalNumber-Methode

Gibt ein DecimalNumber-Objekt zurück.

Überladungsliste

Name	Beschreibung
getDecimalNumber(int)-Methode	Gibt ein DecimalNumber-Objekt zurück.
getDecimalNumber(string)-Methode	Gibt ein DecimalNumber-Objekt zurück.

getDecimalNumber(int)-Methode

Gibt ein DecimalNumber-Objekt zurück.

Syntax

```
DecimalNumber ResultSet.getDecimalNumber(  
    int ordinal  
) throws ULjException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die Darstellung des benannten Werts als DecimalNumber-Objekt.

Siehe auch

- [DecimalNumber-Schnittstelle \[UltraLiteJ\] auf Seite 146](#)

getDecimalNumber(string)-Methode

Gibt ein DecimalNumber-Objekt zurück.

Syntax

```
DecimalNumber ResultSet.getDecimalNumber(  
    String name  
) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die Darstellung des benannten Werts als DecimalNumber-Objekt.

Siehe auch

- [DecimalNumber-Schnittstelle \[UltraLiteJ\] auf Seite 146](#)

getDouble-Methode

Gibt einen double-Wert zurück, basierend auf der Spaltennummer.

Überladungsliste

Name	Beschreibung
getDouble(int)-Methode	Gibt einen double-Wert zurück, basierend auf der Spaltennummer.
getDouble(string)-Methode	Gibt einen double-Wert zurück, basierend auf dem Spaltennamen.

getDouble(int)-Methode

Gibt einen double-Wert zurück, basierend auf der Spaltennummer.

Syntax

```
double ResultSet.getDouble(int ordinal) throws SQLException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die double-Darstellung des benannten Werts.

getDouble(string)-Methode

Gibt einen double-Wert zurück, basierend auf dem Spaltennamen.

Syntax

```
double ResultSet.getDouble(String name) throws SQLException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die double-Darstellung des benannten Werts.

getFloat-Methode

Gibt einen float-Wert zurück.

Überladungsliste

Name	Beschreibung
getFloat(int)-Methode	Gibt einen float-Wert zurück.

Name	Beschreibung
getFloat(string)-Methode	Gibt einen float-Wert zurück.

getFloat(int)-Methode

Gibt einen float-Wert zurück.

Syntax

```
float ResultSet.getFloat(int ordinal) throws ULJException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die float-Darstellung des benannten Werts.

getFloat(string)-Methode

Gibt einen float-Wert zurück.

Syntax

```
float ResultSet.getFloat(String name) throws ULJException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die float-Darstellung des benannten Werts.

getInt-Methode

Gibt einen Ganzzahlwert zurück.

Überladungsliste

Name	Beschreibung
getInt(int)-Methode	Gibt einen Ganzzahlwert zurück.
getInt(string)-Methode	Gibt einen Ganzzahlwert zurück.

getInt(int)-Methode

Gibt einen Ganzzahlwert zurück.

Syntax

```
int ResultSet.getInt(int ordinal) throws SQLException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die Ganzzahldarstellung des benannten Werts.

getInt(string)-Methode

Gibt einen Ganzzahlwert zurück.

Syntax

```
int ResultSet.getInt(String name) throws SQLException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die Ganzzahldarstellung des benannten Werts.

getLong-Methode

Gibt einen Ganzzahlwert (long integer) zurück.

Überladungsliste

Name	Beschreibung
getLong(int)-Methode	Gibt einen Ganzzahlwert (long integer) zurück.
getLong(string)-Methode	Gibt einen Ganzzahlwert (long integer) zurück.

getLong(int)-Methode

Gibt einen Ganzzahlwert (long integer) zurück.

Syntax

```
long ResultSet.getLong(int ordinal) throws SQLException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die Ganzzahldarstellung (long integer) des benannten Werts.

getLong(string)-Methode

Gibt einen Ganzzahlwert (long integer) zurück.

Syntax

```
long ResultSet.getLong(String name) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die Ganzzahldarstellung (long integer) des benannten Werts.

getOrdinal-Methode

Gibt die Ordinalzahl (mit Basis Eins) für den Wert zurück, der von einer Zeichenfolge dargestellt wird.

Syntax

```
int ResultSet.GetOrdinal(String name) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Der Ordinalzahlwert.

getResultSetMetadata-Methode

Gibt ein ResultSetMetadata-Objekt zurück, das die Metadaten für das ResultSet-Objekt enthält.

Syntax

```
ResultSetMetadata ResultSet.getResultSetMetadata() throws ULjException
```

Rückgabe

Das ResultSetMetadata-Objekt.

getRowCount-Methode [Android]

Ruft die Anzahl der Zeilen in der Tabelle ab.

Syntax

```
long ResultSet.getRowCount(long threshold) throws ULJException
```

Parameter

- **threshold** Das Limit für die Anzahl der zu zählenden Zeilen. Der Wert 0 bedeutet keine Beschränkung.

Rückgabe

Die Anzahl der Zeilen in der Tabelle.

Bemerkungen

Diese Methode entspricht dem Ausführen von "SELECT COUNT(*) FROM table".

getSize-Methode

Ruft die tatsächliche Größe einer Ergebnismengenspalte ab.

Überladungsliste

Name	Beschreibung
getSize(int)-Methode	Ruft die tatsächliche Größe einer Ergebnismengenspalte ab.
getSize(string)-Methode	Ruft die tatsächliche Größe einer Ergebnismengenspalte ab.

getSize(int)-Methode

Ruft die tatsächliche Größe einer Ergebnismengenspalte ab.

Syntax

```
int ResultSet.getSize(int ordinal) throws ULJException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die tatsächliche Größe einer Ergebnismengenspalte.

getSize(string)-Methode

Ruft die tatsächliche Größe einer Ergebnismengenspalte ab.

Syntax

```
int ResultSet.getSize(String name) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die tatsächliche Größe einer Ergebnismengenspalte.

getString-Methode

Gibt einen Zeichenfolgenwert zurück.

Überladungsliste

Name	Beschreibung
getString(int)-Methode	Gibt einen Zeichenfolgenwert zurück.
getString(string)-Methode	Gibt einen Zeichenfolgenwert zurück.

getString(int)-Methode

Gibt einen Zeichenfolgenwert zurück.

Syntax

```
String ResultSet.getString(int ordinal) throws ULjException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die Zeichenfolgendarstellung des benannten Werts.

getString(string)-Methode

Gibt einen Zeichenfolgenwert zurück.

Syntax

```
String ResultSet.getString(String name) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die Zeichenfolgendarstellung des benannten Werts.

getUUIDValue-Methode

Gibt ein UUIDValue-Objekt zurück.

Überladungsliste

Name	Beschreibung
getUUIDValue(int)-Methode	Gibt ein UUIDValue-Objekt zurück.
getUUIDValue(string)-Methode	Gibt ein UUIDValue-Objekt zurück.

getUUIDValue(int)-Methode

Gibt ein UUIDValue-Objekt zurück.

Syntax

```
UUIDValue ResultSet.getUUIDValue(int ordinal) throws ULjException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

Die Darstellung des benannten Werts als UUIDValue-Objekt.

Siehe auch

- [UUIDValue-Schnittstelle \[UltraLiteJ\] auf Seite 292](#)

getUUIDValue(string)-Methode

Gibt ein UUIDValue-Objekt zurück.

Syntax

```
UUIDValue ResultSet.getUUIDValue(String name) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

Die Darstellung des benannten Werts als UUIDValue-Objekt.

isNull-Methode

Testet, ob der Wert an der Stelle der angegebenen Spaltennummer NULL ist.

Überladungsliste

Name	Beschreibung
isNull(int)-Methode	Testet, ob der Wert an der Stelle der angegebenen Spaltennummer NULL ist.
isNull(String)-Methode	Testet, ob der Wert an der Stelle des angegebenen Spaltennamens NULL ist.

isNull(int)-Methode

Testet, ob der Wert an der Stelle der angegebenen Spaltennummer NULL ist.

Syntax

```
boolean ResultSet.isNull(int ordinal) throws ULjException
```

Parameter

- **ordinal** Eine Ganzzahl mit Basis Eins, die die Spaltennummer darstellt, wie sie in der SQL-Anweisung eingeordnet ist.

Rückgabe

TRUE, wenn der Wert Null ist, ansonsten FALSE.

isNull(String)-Methode

Testet, ob der Wert an der Stelle des angegebenen Spaltennamens NULL ist.

Syntax

```
boolean ResultSet.isNull(String name) throws ULjException
```

Parameter

- **name** Eine Zeichenfolge, die den Tabellenspaltennamen darstellt.

Rückgabe

TRUE, wenn der Wert Null ist, ansonsten FALSE.

last-Methode [Android]

Verschiebt den Cursor in die letzte Zeile.

Syntax

```
boolean ResultSet.last() throws ULjException
```

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

next-Methode

Ruft die nächste Datenzeile im ResultSet-Objekt ab.

Syntax

```
boolean ResultSet.next() throws ULjException
```

Rückgabe

TRUE, wenn die nächste Zeile erfolgreich abgerufen wird, ansonsten FALSE.

Siehe auch

- [ResultSetMetadata-Schnittstelle \[UltraLiteJ\] auf Seite 216](#)

previous-Methode

Ruft die vorherige Datenzeile im ResultSet-Objekt ab.

Syntax

```
boolean ResultSet.previous() throws ULjException
```

Rückgabe

TRUE, wenn die vorherige Zeile erfolgreich abgerufen wird, ansonsten FALSE.

relative-Methode [Android]

Verschiebt den Cursor von der aktuellen Cursorposition ausgehend um die durch den Offset angegebene Anzahl von Zeilen.

Syntax

```
boolean ResultSet.relative(int offset) throws ULjException
```

Parameter

- **offset** Die Anzahl Zeilen, um die die Position verschoben wird.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

ResultSetMetadata-Schnittstelle

Ist einem ResultSet-Objekt zugeordnet und enthält eine Methode, um Spalteninformationen bereitzustellen.

Syntax

```
public interface ResultSetMetadata
```

Mitglieder

Alle Mitglieder der ResultSetMetadata-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getAliasName-Methode	Gibt den Aliasnamen für eine Spalte zurück.
getColumnCount-Methode	Gibt die Gesamtzahl der Spalten im ResultSet-Objekt zurück.
getCorrelationName-Methode	Gibt den Korrelationsnamen für eine Spalte zurück.
getDomainName-Methode	Gibt den Namen der Domäne zurück.
getDomainPrecision-Methode	Gibt die Gesamtstellenzahl des Domänenwerts zurück.
getDomainScale-Methode	Gibt die Dezimalstellen des Domänenwerts zurück.
getDomainSize-Methode	Gibt die Größe des Domänenwerts zurück.
getDomainType-Methode	Gibt den Typ der Domäne zurück.
getQualifiedName-Methode	Gibt den qualifizierten Namen für eine Spalte zurück.
getTableColumnName-Methode	Gibt den Spaltennamen in der Tabelle oder abgeleiteten Tabelle zurück.
getTableName-Methode	Gibt den Tabellennamen für eine Spalte zurück.
getWrittenName-Methode	Gibt den geschriebenen Namen für eine Spalte zurück.

Bemerkungen

Diese Schnittstelle wird mit der ResultSet.getResultSetMetadata-Methode aufgerufen.

Wenn eine Spalte in der Auswahlliste des ResultSet-Objekts ein einfacher Name oder zusammengesetzter Name (Tabellenname.Spaltenname, Korrelationsname.Spaltenname) ist, können die folgenden Informationen über diesen Namen extrahiert werden, sofern sie vorhanden sind:

- Aliasname.
- Korrelationsname
- Qualifizierte Version des Namens.
- Tabellenname
- Geschriebener Name.

Für jede Spalte in der Select-Liste des ResultSet-Objekts können die folgenden Informationen über die Domäne der betreffenden Spalte abgerufen werden:

- Spaltentyp: eine Ganzzahl aus der Domain-Schnittstelle.
- Name der Domäne.
- Größe der Domäne, für VARCHAR- und BINARY-Domänen.
- Dezimalstellen und Gesamtstellenzahl, für NUMERIC-Domänen.

Siehe auch

- [Domain-Schnittstelle \[UltraLiteJ\] auf Seite 149](#)
- [ResultSet-Schnittstelle \[UltraLiteJ\] auf Seite 197](#)
- [ResultSet.getResultSetMetadata-Methode \[UltraLiteJ\] auf Seite 210](#)

getAliasName-Methode

Gibt den Aliasnamen für eine Spalte zurück.

Syntax

```
String ResultSetMetadata.getAliasName(int column_no) throws ULjException
```

Parameter

- **column_no** Die Nummer (auf Basis 1) der Spalte in der Auswahlliste.

Rückgabe

NULL, falls es keinen Aliasnamen für die Spalte gibt, andernfalls der Aliasname für eine Spalte.

Bemerkungen

Der Aliasname ([AS] Name) kann angegeben werden, um eine Spalte zu referenzieren.

getColumnCount-Methode

Gibt die Gesamtzahl der Spalten im ResultSet-Objekt zurück.

Syntax

```
int ResultSetMetadata.getColumnCount() throws ULjException
```

Rückgabe

Die Spaltenanzahl.

getCorrelationName-Methode

Gibt den Korrelationsnamen für eine Spalte zurück.

Syntax

```
String ResultSetMetadata.getCorrelationName(  
    int column_no  
) throws ULjException
```

Parameter

- **column_no** Die Nummer (auf Basis 1) der Spalte in der Auswahlliste.

Rückgabe

NULL, falls es keinen Korrelationsnamen für die Spalte gibt, andernfalls der Korrelationsname für eine Spalte.

Bemerkungen

Der Korrelationsname ([AS] Korrelationsname), der in der FROM-Klausel angegeben wird, um einen Tabellenausdruck anzugeben, z.B. eine abgeleitete Tabelle.

getDomainName-Methode

Gibt den Namen der Domäne zurück.

Syntax

```
String ResultSetMetadata.getDomainName(  
    int column_no  
) throws ULjException
```

Parameter

- **column_no** Die Nummer (auf Basis 1) der Spalte in der Auswahlliste.

Rückgabe

Der Domänenname

getDomainPrecision-Methode

Gibt die Gesamtstellenzahl des Domänenwerts zurück.

Syntax

```
int ResultSetMetadata.getDomainPrecision(  
    int column_no  
) throws ULjException
```

Parameter

- **column_no** Die Nummer (auf Basis 1) der Spalte in der Auswahlliste.

Rückgabe

Die Gesamtstellenzahl.

getDomainScale-Methode

Gibt die Dezimalstellen des Domänenwerts zurück.

Syntax

```
int ResultSetMetadata.getDomainScale(int column_no) throws ULjException
```

Parameter

- **column_no** Die Nummer (auf Basis 1) der Spalte in der Auswahlliste.

Rückgabe

Die Anzahl der Dezimalstellen.

getDomainSize-Methode

Gibt die Größe des Domänenwerts zurück.

Syntax

```
int ResultSetMetadata.getDomainSize(int column_no) throws ULjException
```

Parameter

- **column_no** Die Nummer (auf Basis 1) der Spalte in der Auswahlliste.

Rückgabe

Die Größe.

getDomainType-Methode

Gibt den Typ der Domäne zurück.

Syntax

```
short ResultSetMetadata.getDomainType(int column_no) throws ULjException
```

Parameter

- **column_no** Die Nummer (auf Basis 1) der Spalte in der Auswahlliste.

Rückgabe

Der Domänentyp, als Ganzzahl ausgedrückt.

getQualifiedName-Methode

Gibt den qualifizierten Namen für eine Spalte zurück.

Syntax

```
String ResultSetMetadata.getQualifiedName(  
    int column_no  
) throws ULjException
```

Parameter

- **column_no** Die Nummer (auf Basis 1) der Spalte in der Auswahlliste.

Rückgabe

NULL, falls es keinen qualifizierten Namen für die Spalte gibt, andernfalls der qualifizierte Name für eine Spalte.

Bemerkungen

Wenn die ResultSet-Spalte eine Spalte in einer Tabelle referenziert, ist der zurückgegebene Name ein zusammengesetzter Name, bestehend aus einem Korrelationsnamen (oder einem Tabellennamen, wenn der Korrelationsname nicht angegeben wurde), gefolgt vom Namen der Spalte in der Tabelle.

Wenn die ResultSet-Spalte sich nicht auf eine Spalte in einer Tabelle bezieht und ein Alias angegeben wurde, wird der Aliasname zurückgegeben.

getTableColumnName-Methode

Gibt den Spaltennamen in der Tabelle oder abgeleiteten Tabelle zurück.

Syntax

```
String ResultSetMetadata.getTableColumnName(  
    int column_no  
) throws ULjException
```

Parameter

- **column_no** Die Nummer (auf Basis 1) der Spalte in der Auswahlliste.

Rückgabe

NULL, falls es keinen Tabellennamen für die Spalte gibt, andernfalls der Tabellename für eine Spalte.

getTableName-Methode

Gibt den Tabellennamen für eine Spalte zurück.

Syntax

```
String ResultSetMetadata.getTableName(int column_no) throws ULjException
```

Parameter

- **column_no** Die Nummer (auf Basis 1) der Spalte in der Auswahlliste.

Rückgabe

NULL, falls es keinen Tabellennamen für die Spalte gibt, andernfalls der Tabellename für eine Spalte.

Bemerkungen

Die Tabelle ist der Name der Tabelle, die die ResultSet-Spalte referenziert (möglicherweise durch einen Korrelationsnamen).

getWrittenName-Methode

Gibt den geschriebenen Namen für eine Spalte zurück.

Syntax

```
String ResultSetMetadata.getWrittenName(  
    int column_no  
) throws ULjException
```

Parameter

- **column_no** Die Nummer (auf Basis 1) der Spalte in der Auswahlliste.

Rückgabe

NULL, falls es keinen geschriebenen Namen für die Spalte gibt, andernfalls der geschriebene Name für eine Spalte.

Bemerkungen

Der geschriebene Name ist der einfache oder zusammengesetzte Name, den die Select-Liste als die angegebene Spalte enthält.

SISListener-Schnittstelle [BlackBerry]

Wartet auf serverinitiierte Synchronisationsmeldungen.

Syntax

```
public interface SISListener
```

Mitglieder

Alle Mitglieder der SISListener-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
startListening-Methode	Erstellt und startet einen Listening-Thread.
stopListening-Methode	Stoppt den Listening-Thread.

Bemerkungen

Die Anwendung erstellt mit der geeigneten DatabaseManager.createSISHTTPListener-Methode eine Instanz der SISListener-Schnittstelle.

Siehe auch

- [DatabaseManager.createSISHTTPListener-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 144](#)

startListening-Methode

Erstellt und startet einen Listening-Thread.

Syntax

```
void SISListener.startListening()
```

stopListening-Methode

Stoppt den Listening-Thread.

Syntax

```
void SISListener.stopListening()
```

SISRequestHandler-Schnittstelle [BlackBerry]

Verarbeitet serverinitiierte Synchronisationsanforderungen.

Syntax

```
public interface SISRequestHandler
```

Mitglieder

Alle Mitglieder der SISRequestHandler-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
onError-Methode	Verarbeitet während des Abhörens auftretende Fehler in der serverinitiierten Synchronisation.
onRequest-Methode	Verarbeitet Anforderungen zur serverinitiierten Synchronisation auf Worker-Threads.

Siehe auch

- [SISListener-Schnittstelle \[BlackBerry\] \[UltraLiteJ\] auf Seite 221](#)

onError-Methode

Verarbeitet während des Abhörens auftretende Fehler in der serverinitiierten Synchronisation.

Syntax

```
void SISRequestHandler.onError(String text)
```

Parameter

- **text** Die Zeichenfolgendarstellung der Ausnahmebedingung.

Bemerkungen

Um das Warten zu stoppen, rufen Sie explizit die stopListening-Methode in der SISListener-Schnittstelle auf.

onRequest-Methode

Verarbeitet Anforderungen zur serverinitiierten Synchronisation auf Worker-Threads.

Syntax

```
void SISRequestHandler.onRequest(String text)
```

Parameter

- **text** Die von der Anforderung gesendete Zeichenfolge

SQLInfo-Schnittstelle [Android]

Zeigt Informationen zu einer ausgeführten SQL-Anweisung an.

Syntax

```
public interface SQLInfo
```

Mitglieder

Alle Mitglieder der SQLInfo-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getMessage-Methode	Gibt die Meldung zurück, die dem SQL-Code zugeordnet ist.
getParameter-Methode	Gibt den angegebenen Parameter zurück.
getParameterCount-Methode	Gibt die Anzahl der Parameter der Nachricht zurück.
getSQLCode-Methode	Ruft den Fehlercode (SQLCODE) für die ausgeführte SQL-Anweisung ab.
getSQLCount-Methode	Gibt einen Wert zurück, der von dem Ergebnis von der ausgeführten SQL-Anweisung abhängt.

getMessage-Methode

Gibt die Meldung zurück, die dem SQL-Code zugeordnet ist.

Syntax

```
String SQLInfo.getMessage()
```

getParameter-Methode

Gibt den angegebenen Parameter zurück.

Syntax

```
String SQLInfo.getParameter(short param_no)
```

Parameter

- **param_no** Eine Parameternummer auf Basis eins.

Rückgabe

Der Parameter.

getParameterCount-Methode

Gibt die Anzahl der Parameter der Nachricht zurück.

Syntax

```
short SQLInfo.getParameterCount()
```

Rückgabe

Die Anzahl der Parameter.

getSQLCode-Methode

Ruft den Fehlercode (SQLCODE) für die ausgeführte SQL-Anweisung ab.

Syntax

```
int SQLInfo.getSQLCode()
```

Rückgabe

Der SQLCODE-Wert.

getSQLCount-Methode

Gibt einen Wert zurück, der von dem Ergebnis von der ausgeführten SQL-Anweisung abhängt.

Syntax

```
int SQLInfo.getSQLCount()
```

Rückgabe

Die Anzahl der durch die Anweisung betroffenen Zeilen, nachdem eine INSERT-, UPDATE- oder DELETE-Anweisung ausgeführt wurde. Der Rückgabewert ist der Offset für die verbundene dynamische SQL-Anweisung, die dem Fehler entspricht, wenn ein SQLE_SYNTAX_ERROR-Fehler auftritt.

StreamHTTPParams-Schnittstelle

Stellt die HTTP-Datenstromparameter dar, die festlegen, wie mit einem MobiLink-Server unter Verwendung von HTTP kommuniziert werden soll.

Syntax

```
public interface StreamHTTPParams
```

Abgeleitete Klassen

- [StreamHTTPSPParams-Schnittstelle \[UltraLiteJ\] auf Seite 238](#)

Mitglieder

Alle Mitglieder der StreamHTTPParams-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
addCustomHTTPHeader-Methode [BlackBerry]	Fügt einen Nachrichtenheader in jede HTTP-Anforderung ein.

Name	Beschreibung
getCustomHTTPHeaders-Methode [BlackBerry]	Gibt ein Hash-Tabellenobjekt zurück, das die HTTP-Header enthält, die mit der addCustomHTTPHeader-Methode angegeben wurden.
getE2eePublicKey-Methode [Blackberry]	Gibt den Namen der Datei zurück, die den öffentlichen Schlüssel für die Ende-zu-Ende-Verschlüsselung enthält.
getE2eeType-Methode [BlackBerry]	Gibt die Ende-zu-Ende-Verschlüsselung zurück, die gerade benutzt wird.
getExtraParameters-Methode [Android]	Ruft die zusätzlichen Netzwerkprotokolloptionen für den MobiLink-Client ab.
getHost-Methode	Gibt den Hostnamen des MobiLink-Servers zurück.
getHTTPPassword-Methode [BlackBerry]	Gibt das HTTP-Kennwort zurück.
getHTTPUserId-Methode [BlackBerry]	Gibt die HTTP Benutzer-ID zurück.
getOutputBufferSize-Methode	Gibt die Größe (in Byte) des Ausgabepuffers zurück, in dem Daten gespeichert werden, bevor sie an den MobiLink-Server gesendet werden.
getPort-Methode	Gibt die Portnummer zurück, die zum Verbinden mit dem MobiLink-Server verwendet wird.
getURLSuffix-Methode	Gibt das URL-Suffix des MobiLink-Servers zurück.
isRestartable-Methode	Ermittelt, ob die neu startbare HTTP-Verbindung verwendet wird.
setE2eePublicKey-Methode [Android]	Legt den Namen der Datei fest, die den öffentlichen Schlüssel für die Ende-zu-Ende-Verschlüsselung enthält.
setE2eeType-Methode [BlackBerry]	Legt den Typ der Ende-zu-Ende-Verschlüsselung fest, die verwendet werden soll.
setExtraParameters-Methode [Android]	Legt zusätzliche Netzwerkprotokolloptionen für den MobiLink-Client fest.
setHost-Methode	Setzt den Hostnamen des MobiLink-Servers.
setHTTPUserIdAndPassword-Methode [BlackBerry]	Legt die Benutzer-ID und das Kennwort fest, die für Basic HTTP-Authentifizierung verwendet werden sollen, wie in RFC 2617 beschrieben.

Name	Beschreibung
setOutputBufferSize-Methode	Setzt die Größe (in Byte) des Ausgabepuffers, in dem Daten gespeichert werden, bevor sie an den MobiLink-Server gesendet werden.
setPort-Methode	Setzt die Portnummer, die zum Verbinden mit dem MobiLink-Server verwendet wird.
setRestartable-Methode	Aktiviert bzw. deaktiviert die neu startbare HTTP-Verbindung.
setURLSuffix-Methode	Legt das URL-Suffix zum Verbinden mit dem MobiLink-Server fest.
setZlibCompression-Methode	Aktiviert bzw. deaktiviert die ZLIB-Komprimierung.
setZlibDownloadWindowSize-Methode	Legt die Größe des Download-Fensters für die ZLIB-Komprimierung fest.
setZlibUploadWindowSize-Methode	Legt die Größe des Upload-Fensters für die ZLIB-Komprimierung fest.
zlibCompressionEnabled-Methode	Ermittelt, ob die ZLIB-Komprimierung aktiviert ist.
E2EE_RSA-Variable [BlackBerry]	Legt die Ende-zu-Ende-Verschlüsselung auf RSA-Basis bei der Übergabe an die setE2eeType-Methode fest.

Bemerkungen

Das folgende Beispiel setzt die Datenstromparameter zur Kommunikation mit einem MobiLink -Server auf einem Host namens "MyMLHost". Der Server wurde mit den folgenden Parametern gestartet: "-x http(port=1234)":

```
SyncParams syncParams = myConnection.createSyncParams(
    SyncParams.HTTP_STREAM,
    "MyUniqueMLUserID",
    "MyMLScriptVersion"
);
StreamHTTPParams httpParams = syncParams.getStreamParams();
httpParams.setHost("MyMLHost");
httpParams.setPort(1234);
```

Instanzen, die diese Schnittstelle implementieren, werden durch die Methode SyncParams.getStreamParams zurückgegeben.

Siehe auch

- [SyncParams-Klasse \[UltraLiteJ\] auf Seite 250](#)
- [SyncParams.getStreamParams-Methode \[UltraLiteJ\] auf Seite 256](#)

addCustomHTTPHeader-Methode [BlackBerry]

Fügt einen Nachrichtenheader in jede HTTP-Anforderung ein.

Syntax

```
void StreamHTTPParms.addCustomHTTPHeader(String name, String value)
```

Parameter

- **name** Der Header-Name
- **value** Der Header-Wert

Bemerkungen

Wenn diese Methode mehr als einmal mit demselben name-Parameter aufgerufen wird, werden die Werte miteinander verkettet in einer durch Kommas getrennten Liste angegeben.

Die folgenden Standard-Header können mit dieser Methode nicht geändert werden:

- Connection
- Content-Length
- User-Agent
- Content-Type

Andere Header können mit Java VM geändert werden.

Geben Sie benutzerangepasste Cookies an, indem Sie diese Methode mit Cookies als Header-Name aufrufen.

Verwenden Sie für Android-Smartphones die setExtraParameters-Methode, um einen custom_header-Parameter anzugeben.

Siehe auch

- [StreamHTTPParms.setExtraParameters-Methode \[Android\] \[UltraLiteJ\] auf Seite 233](#)

getCustomHTTPHeaders-Methode [BlackBerry]

Gibt ein Hash-Tabellenobjekt zurück, das die HTTP-Header enthält, die mit der addCustomHTTPHeader-Methode angegeben wurden.

Syntax

```
java.util.Hashtable StreamHTTPParms.getCustomHTTPHeaders( )
```

Rückgabe

Eine Hashtabelle, die die HTTP-Header enthält, die durch die addCustomHTTPHeader-Methode angegeben werden. Der Schlüssel ist der Header-Name und der Wert ist der Header-Wert.

Siehe auch

- [StreamHTTPParams.addCustomHTTPHeader-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 228](#)

getE2eePublicKey-Methode [Blackberry]

Gibt den Namen der Datei zurück, die den öffentlichen Schlüssel für die Ende-zu-Ende-Verschlüsselung enthält.

Syntax

```
String StreamHTTPParams.getE2eePublicKey()
```

Rückgabe

Der Name der Datei, die den öffentlichen Schlüssel für die Ende-zu-Ende-Verschlüsselung enthält.

Siehe auch

- [StreamHTTPParams.setE2eePublicKey-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 232](#)

getE2eeType-Methode [BlackBerry]

Gibt die Ende-zu-Ende-Verschlüsselung zurück, die gerade benutzt wird.

Syntax

```
short StreamHTTPParams.getE2eeType()
```

Rückgabe

Der Ende-zu-Ende-Verschlüsselungstypcode.

getExtraParameters-Methode [Android]

Ruft die zusätzlichen Netzwerkprotokolloptionen für den MobiLink-Client ab.

Syntax

```
String StreamHTTPParams.getExtraParameters()
```

Rückgabe

Die festgelegten zusätzlichen Protokolloptionen.

getHost-Methode

Gibt den Hostnamen des MobiLink-Servers zurück.

Syntax

```
String StreamHTTPParams.getHost()
```

Rückgabe

Der Name des Hosts

Siehe auch

- [StreamHTTPParams.setHost-Methode \[UltraLiteJ\] auf Seite 233](#)
- [StreamHTTPParams.getPort-Methode \[UltraLiteJ\] auf Seite 231](#)
- [StreamHTTPParams.setPort-Methode \[UltraLiteJ\] auf Seite 235](#)

getHTTPPassword-Methode [BlackBerry]

Gibt das HTTP-Kennwort zurück.

Syntax

```
String StreamHTTPParams.getHTTPPassword()
```

Rückgabe

Das Kennwort, das zuvor von der setHTTPUserIdAndPassword-Methode gesetzt wurde.

Siehe auch

- [StreamHTTPParams.setHTTPUserIdAndPassword-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 234](#)

getHTTPUserId-Methode [BlackBerry]

Gibt die HTTP Benutzer-ID zurück.

Syntax

```
String StreamHTTPParams.getHTTPUserId()
```

Rückgabe

Die Benutzer-ID, die zuvor von der setHTTPUserIdAndPassword Methode gesetzt wurde.

Siehe auch

- [StreamHTTPParams.setHTTPUserIdAndPassword-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 234](#)

getOutputBufferSize-Methode

Gibt die Größe (in Byte) des Ausgabepuffers zurück, in dem Daten gespeichert werden, bevor sie an den MobiLink-Server gesendet werden.

Syntax

```
int StreamHTTPParams.getOutputBufferSize()
```

Rückgabe

Die Ganzzahl, die die Puffergröße enthält.

Bemerkungen

Eine Erhöhung dieses Werts vermindert möglicherweise die Anzahl der Netzwerkbereinigungen, die zum Senden eines großen Uploads benötigt werden, allerdings bei erhöhter Speicherbeanspruchung. In HTTP sendet jede Bereinigung einen großen (ca. 250 Byte) HTTP-Header. Eine Verminderung der Anzahl der Bereinigungen kann die Bandbreitennutzung verringern.

Siehe auch

- [StreamHTTPParams.setOutputBufferSize-Methode \[UltraLiteJ\] auf Seite 234](#)

getPort-Methode

Gibt die Portnummer zurück, die zum Verbinden mit dem MobiLink-Server verwendet wird.

Syntax

```
int StreamHTTPParams.getPort()
```

Rückgabe

Die Portnummer des MobiLink-Servers.

Siehe auch

- [StreamHTTPParams.setPort-Methode \[UltraLiteJ\] auf Seite 235](#)

getURLSuffix-Methode

Gibt das URL-Suffix des MobiLink-Servers zurück.

Syntax

```
String StreamHTTPParams.getURLSuffix()
```

Rückgabe

Die Zeichenfolge, die das URL-Suffix enthält.

Siehe auch

- [StreamHTTPParams.setURLSuffix-Methode \[UltraLiteJ\] auf Seite 236](#)

isRestartable-Methode

Ermittelt, ob die neu startbare HTTP-Verbindung verwendet wird.

Syntax

```
boolean StreamHTTPParams.isRestartable()
```

Rückgabe

TRUE, wenn die neu startbare HTTP-Verbindung aktiviert ist, ansonsten FALSE.

Siehe auch

- [StreamHTTPParams.setRestartable-Methode \[UltraLiteJ\] auf Seite 235](#)

setE2eePublicKey-Methode [Android]

Legt den Namen der Datei fest, die den öffentlichen Schlüssel für die Ende-zu-Ende-Verschlüsselung enthält.

Syntax

```
void StreamHTTPParams.setE2eePublicKey(String public_key)
```

Parameter

- **public_key** Der Name der für die Verschlüsselung verwendeten Datei mit dem öffentlichen RSA-Schlüssel. Dieser Name muss DER-kodiert sein.

Bemerkungen

Der Standardwert ist NULL und bedeutet, dass keine Ende-zu-Ende-Verschlüsselung verwendet wird.

Diese Methode entspricht der e2ee_public_key-Protokolloption.

Der öffentliche Schlüssel kann auf einer SD-Karte oder auf dem Geräteobjektspeicher gespeichert werden.

Wenn Sie eine SD-Karte verwenden, sollte der public_key-Parameter folgende Form haben:

file://path

path ist der absolute Pfad für die Datei auf der Karte. Beispiel: *file:///SDCard/ulj/public_key.der* ist ein gültiger public_key-Parameter.

Wenn Sie einen Objektspeicher benutzen, verwenden Sie das UltraLite Java Edition-Datenbankübertragungs-Dienstprogramm für den Download der Datei vom MobiLink-Server. Der Schlüssel muss eine *der*-Dateierweiterung haben.

Siehe auch

- [StreamHTTPParams.getE2eePublicKey-Methode \[Blackberry\] \[UltraLiteJ\] auf Seite 229](#)
- „e2ee_public_key“ [*MobiLink - Clientadministration*]
- „UltraLite Java Edition-Dienstprogramm für die Datenbankübertragung“ [*UltraLite - Datenbankverwaltung*]

setE2eeType-Methode [BlackBerry]

Legt den Typ der Ende-zu-Ende-Verschlüsselung fest, die verwendet werden soll.

Syntax

```
void StreamHTTPParams.setE2eeType(short type)
```

Parameter

- **type** Übergeben Sie die StreamHTTPParams.E2EE_RSA-Konstante. StreamHTTPParams.E2EE_RSA ist die Standardeinstellung.

Siehe auch

- [StreamHTTPParams.E2EE_RSA-Variable \[BlackBerry\] \[UltraLiteJ\] auf Seite 238](#)

setExtraParameters-Methode [Android]

Legt zusätzliche Netzwerkprotokolloptionen für den MobiLink-Client fest.

Syntax

```
void StreamHTTPParams.setExtraParameters(String parms)
```

Parameter

- **parms** Eine Liste von durch Semikola getrennten Protokolloptionen.

Bemerkungen

Diese Optionen werden an die Liste der Einstellungen angehängt, die aus den Methoden dieser Klasse resultieren.

Durch diese Methode festgelegte Optionen heben die entsprechenden durch andere Methoden festgelegten Optionen auf. Wenn beispielsweise "host=abc" in den zusätzlichen Parametern enthalten ist und die setHost("xyz")-Methode aufgerufen wird, lautet die host-Option "abc".

setHost-Methode

Setzt den Hostnamen des MobiLink-Servers.

Syntax

```
void StreamHTTPParams.setHost(String v)
```

Parameter

- **v** Der Name des Hosts

Bemerkungen

Der Standardwert ist NULL, womit ein "localhost" angezeigt wird.

Siehe auch

- [StreamHTTPParams.getHost-Methode \[UltraLiteJ\] auf Seite 229](#)
- [StreamHTTPParams.getPort-Methode \[UltraLiteJ\] auf Seite 231](#)
- [StreamHTTPParams.setPort-Methode \[UltraLiteJ\] auf Seite 235](#)

setHTTPUserIdAndPassword-Methode [BlackBerry]

Legt die Benutzer-ID und das Kennwort fest, die für Basic HTTP-Authentifizierung verwendet werden sollen, wie in RFC 2617 beschrieben.

Syntax

```
void StreamHTTPParams.setHTTPUserIdAndPassword(  
    String userid,  
    String password  
)
```

Parameter

- **userid** Die zu verwendende Benutzer-ID.
- **password** Das zu verwendende Kennwort.

Bemerkungen

Bei der Basic-HTTP-Authentifizierung werden Kennwörter in HTTP-Headern in lesbarer Form gesendet. Sie können allerdings HTTPS zur Verschlüsselung der Header verwenden, um das Kennwort zu schützen.

Siehe auch

- [RFC 2617: HTTP-Authentifizierung](#)

setOutputBufferSize-Methode

Setzt die Größe (in Byte) des Ausgabepuffers, in dem Daten gespeichert werden, bevor sie an den MobiLink-Server gesendet werden.

Syntax

```
void StreamHTTPParams.setOutputBufferSize(int size)
```

Parameter

- **size** Die neue Puffergröße.

Bemerkungen

Der Standardwert ist 4096. Gültige Werte liegen zwischen 512 und 32768. Eine Erhöhung dieses Werts bewirkt möglicherweise, dass die Java-Laufzeitumgebung im HTTP-Blockmodus sendet, der vom MobiLink-Server nicht verarbeitet werden kann.

Wenn der MobiLink-Server den Fehler "Unbekannte Übertragungskodierung" meldet, versuchen Sie, diesen Wert zu vermindern.

Siehe auch

- [StreamHTTPParams.getOutputStreamBufferSize-Methode \[UltraLiteJ\] auf Seite 230](#)

setPort-Methode

Setzt die Portnummer, die zum Verbinden mit dem MobiLink-Server verwendet wird.

Syntax

```
void StreamHTTPParams.setPort(int v)
```

Parameter

- **v** Eine Portnummer von 1 bis 65535. Nicht im gültigen Wertbereich liegende Werte werden auf den Standardwert zurückgesetzt.

Bemerkungen

Der Standard-Port ist 80 für HTTP-Synchronisationen und 443 für HTTPS Synchronisationen.

Siehe auch

- [StreamHTTPParams.getPort-Methode \[UltraLiteJ\] auf Seite 231](#)

setRestartable-Methode

Aktiviert bzw. deaktiviert die neu startbare HTTP-Verbindung.

Syntax

```
void StreamHTTPParams.setRestartable(boolean isRestartable)
```

Parameter

- **isRestartable** Setzen Sie dies auf TRUE, um die neu startbare HTTP-Verbindung zu aktivieren. Der Standardwert lautet FALSE.

Bemerkungen

Ist die neu startbare HTTP-Verbindung aktiviert, kann UltraLiteJ Netzwerkunterbrechungen tolerieren, sodass Synchronisationen in unzuverlässigen Netzwerken nicht so häufig fehlschlagen.

Damit die neu startbare HTTP-Verbindung verwendet werden kann, müssen sowohl UltraLiteJ als auch der MobiLink-Server CR#690250 übernommen haben.

Siehe auch

- [StreamHTTPParams.isRestartable-Methode \[UltraLiteJ\] auf Seite 231](#)

setURLSuffix-Methode

Legt das URL-Suffix zum Verbinden mit dem MobiLink-Server fest.

Syntax

```
void StreamHTTPParams.setURLSuffix(String v)
```

Parameter

- **v** Die URL-Suffixzeichenfolge.

Bemerkungen

UltraLiteJ erstellt URLs im folgenden Format:

```
[http|https]://host-name:port-number/url-suffix
```

Standardmäßig ist für *url-suffix* "Mobilink/" eingestellt. Sie können für das URL-Suffix den Standardwert einstellen, indem Sie **v** auf NULL setzen.

Der folgende Code zeigt, wie Sie ein URL-Suffix angeben, das ein BlackBerry-Smartphone anweist, Verbindungen nur über WLAN herzustellen:

```
myHTTPParams.setURLSuffix(";deviceside=true;interface=wifi");
```

Der folgende Code zeigt, wie Sie eine URL angeben, die ein BlackBerry-Smartphone anweist, die Synchronisation über das HTTPS-Protokoll mit einem BlackBerry Enterprise Server (BES) durchzuführen. Dies kann für einige BlackBerry-Smartphones erforderlich sein:

```
myHTTPParams.setURLSuffix(";EndToEndRequired");  
End-to-end encryption is required when the host (MobiLink or relay server)  
uses a certificate that is not trusted by the BES (the certificate's chain  
may not be trusted or the hostname in the certificate does not match the  
hostname). When end-to-end encryption is required, the certificate needs to  
be installed and trusted on the device.
```

Siehe auch

- [StreamHTTPParams.getURLSuffix-Methode \[UltraLiteJ\] auf Seite 231](#)

setZlibCompression-Methode

Aktiviert bzw. deaktiviert die ZLIB-Komprimierung.

Syntax

```
void StreamHTTPParams.setZlibCompression(boolean enable)
```

Parameter

- **enable** Setzen Sie dies auf TRUE, um die ZLIB-Komprimierung zu aktivieren, oder auf FALSE, um die ZLIB-Komprimierung zu deaktivieren.

Bemerkungen

Standardmäßig ist die ZLIB-Komprimierung deaktiviert.

Diese Methode entspricht der compression=zlib-Protokolloption.

Siehe auch

- „compression“ [[MobiLink - Clientadministration](#)]

setZlibDownloadWindowSize-Methode

Legt die Größe des Download-Fensters für die ZLIB-Komprimierung fest.

Syntax

```
void StreamHTTPParams.setZlibDownloadWindowSize(int size)
```

Parameter

- **size** Die Größenangabe für das Komprimierungsfenster. Dieser Parameter ist der Logarithmus zur Basis 2 der Fenstergröße (Größe des Verlaufspuffers). Für Android geben Sie einen gültigen Bereich zwischen 9 und 15 einschließlich an. Für BlackBerry geben Sie einen gültigen Bereich zwischen 10 und 15 einschließlich an.

Bemerkungen

Diese Methode entspricht der zlib_download_window_size-Protokolloption.

Siehe auch

- „zlib_download_window_size“ [[MobiLink - Clientadministration](#)]

setZlibUploadWindowSize-Methode

Legt die Größe des Upload-Fensters für die ZLIB-Komprimierung fest.

Syntax

```
void StreamHTTPParams.setZlibUploadWindowSize(int size)
```

Parameter

- **size** Die Größenangabe für das Komprimierungsfenster. Dieser Parameter ist der Logarithmus zur Basis 2 der Fenstergröße (Größe des Verlaufspuffers). Für Android geben Sie einen gültigen Bereich zwischen 9 und 15 einschließlich an. Für BlackBerry geben Sie einen gültigen Bereich zwischen 8 und 15 einschließlich an.

Bemerkungen

Diese Methode entspricht der zlib_upload_window_size-Protokolloption.

Siehe auch

- „zlib_upload_window_size“ [[MobiLink - Clientadministration](#)]

zlibCompressionEnabled-Methode

Ermittelt, ob die ZLIB-Komprimierung aktiviert ist.

Syntax

```
boolean StreamHTTPParams.zlibCompressionEnabled()
```

Rückgabe

TRUE bei Aktivierung, ansonsten FALSE.

Siehe auch

- [StreamHTTPParams.setZlibCompression-Methode \[UltraLiteJ\] auf Seite 236](#)

E2EE_RSA-Variable [BlackBerry]

Legt die Ende-zu-Ende-Verschlüsselung auf RSA-Basis bei der Übergabe an die setE2eeType-Methode fest.

Syntax

```
final short StreamHTTPParams.E2EE_RSA
```

Siehe auch

- [StreamHTTPParams.setE2eeType-Methode \[BlackBerry\] \[UltraLiteJ\] auf Seite 233](#)

StreamHTTPSParms-Schnittstelle

Stellt die HTTPS-Datenstromparameter dar, die festlegen, wie unter Verwendung sicherer HTTPS-Verbindungen mit einem MobiLink-Server kommuniziert werden soll.

Syntax

```
public interface StreamHTTPSParms
```

Basisklassen

- [StreamHTTPParams-Schnittstelle \[UltraLiteJ\] auf Seite 225](#)

Mitglieder

Alle Mitglieder der StreamHTTPSParms-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
addCustomHTTPHeader-Methode [BlackBerry]	Fügt einen Nachrichtenheader in jede HTTP-Anforderung ein.

Name	Beschreibung
getCertificateCompany-Methode	Gibt den Namen der Zertifizierungsgesellschaft für die Verifizierung von sicheren Verbindungen zurück.
getCertificateName-Methode	Gibt den Zertifikatsnamen für die Verifizierung von sicheren Verbindungen zurück.
getCertificateUnit-Methode	Gibt den Namen der Zertifizierungseinheit für die Verifizierung von sicheren Verbindungen zurück.
getCustomHTTPHeaders-Methode [BlackBerry]	Gibt ein Hash-Tabellenobjekt zurück, das die HTTP-Header enthält, die mit der addCustomHTTPHeader-Methode angegeben wurden.
getE2eePublicKey-Methode [Blackberry]	Gibt den Namen der Datei zurück, die den öffentlichen Schlüssel für die Ende-zu-Ende-Verschlüsselung enthält.
getE2eeType-Methode [BlackBerry]	Gibt die Ende-zu-Ende-Verschlüsselung zurück, die gerade benutzt wird.
getExtraParameters-Methode [Android]	Ruft die zusätzlichen Netzwerkprotokolloptionen für den MobiLink-Client ab.
getHost-Methode	Gibt den Hostnamen des MobiLink-Servers zurück.
getHTTPPassword-Methode [BlackBerry]	Gibt das HTTP-Kennwort zurück.
getHTTPUserId-Methode [BlackBerry]	Gibt die HTTP Benutzer-ID zurück.
getOutputBufferSize-Methode	Gibt die Größe (in Byte) des Ausgabepuffers zurück, in dem Daten gespeichert werden, bevor sie an den MobiLink-Server gesendet werden.
getPort-Methode	Gibt die Portnummer zurück, die zum Verbinden mit dem MobiLink-Server verwendet wird.
getTrustedCertificates-Methode	Gibt den Namen der Datei zurück, die eine Liste von vertrauenswürdigen Stammzertifikaten enthält, die bei der sicheren Synchronisation verwendet werden.
getURLSuffix-Methode	Gibt das URL-Suffix des MobiLink-Servers zurück.
isRestartable-Methode	Ermittelt, ob die neu startbare HTTP-Verbindung verwendet wird.
setCertificateCompany-Methode	Setzt den Namen der Zertifizierungsgesellschaft für die Verifizierung von sicheren Verbindungen.

Name	Beschreibung
setCertificateName-Methode	Setzt den Zertifikatsnamen für die Verifizierung von sicheren Verbindungen.
setCertificateUnit-Methode	Setzt den Namen der Zertifizierungseinheit für die Verifizierung von sicheren Verbindungen.
setE2eePublicKey-Methode [Android]	Legt den Namen der Datei fest, die den öffentlichen Schlüssel für die Ende-zu-Ende-Verschlüsselung enthält.
setE2eeType-Methode [BlackBerry]	Legt den Typ der Ende-zu-Ende-Verschlüsselung fest, die verwendet werden soll.
setExtraParameters-Methode [Android]	Legt zusätzliche Netzwerkprotokolloptionen für den MobiLink-Client fest.
setHost-Methode	Setzt den Hostnamen des MobiLink-Servers.
setHTTPUserIdAndPassword-Methode [BlackBerry]	Legt die Benutzer-ID und das Kennwort fest, die für Basic HTTP-Authentifizierung verwendet werden sollen, wie in RFC 2617 beschrieben.
setOutputBufferSize-Methode	Setzt die Größe (in Byte) des Ausgabepuffers, in dem Daten gespeichert werden, bevor sie an den MobiLink-Server gesendet werden.
setPort-Methode	Setzt die Portnummer, die zum Verbinden mit dem MobiLink-Server verwendet wird.
setRestartable-Methode	Aktiviert bzw. deaktiviert die neu startbare HTTP-Verbindung.
setTrustedCertificates-Methode	Setzt eine Datei, die eine Liste von vertrauenswürdigen Stammzertifikaten für die sichere Synchronisation enthält.
setURLSuffix-Methode	Legt das URL-Suffix zum Verbinden mit dem MobiLink-Server fest.
setZlibCompression-Methode	Aktiviert bzw. deaktiviert die ZLIB-Komprimierung.
setZlibDownloadWindowSize-Methode	Legt die Größe des Download-Fensters für die ZLIB-Komprimierung fest.
setZlibUploadWindowSize-Methode	Legt die Größe des Upload-Fensters für die ZLIB-Komprimierung fest.
zlibCompressionEnabled-Methode	Ermittelt, ob die ZLIB-Komprimierung aktiviert ist.

Name	Beschreibung
E2EE_RSA-Variable [Black-Berry]	Legt die Ende-zu-Ende-Verschlüsselung auf RSA-Basis bei der Übergabe an die setE2eeType-Methode fest.

Bemerkungen

Das folgende Beispiel setzt die Datenstromparameter zur Kommunikation mit einem MobiLink -Server auf einem Host namens "MyMLHost". Der Server wurde mit den folgenden Parametern gestartet: "-x https(port=1234;certificate=RSAServer.crt;certificate_password=x)"

```
SyncParms syncParms = myConnection.createSyncParms(
    SyncParms.HTTPS_STREAM,
    "MyUniqueMLUserID",
    "MyMLScriptVersion"
);
StreamHTTPSParms httpsParms =
    (StreamHTTPSParms) syncParms.getStreamParms();
httpsParms.setHost("MyMLHost");
httpsParms.setPort(1234);
```

Das obenstehende Beispiel nimmt an, dass das Zertifikat in RSAServer.crt mit einem vertrauenswürdigen Stammzertifikat verkettet ist, das bereits auf dem Client-Host oder Gerät installiert ist.

Für J2SE können Sie das erforderliche vertrauenswürdige Stammzertifikat bereitstellen, indem Sie eine der folgenden Methoden verwenden:

1. Installieren Sie das vertrauenswürdige Stammzertifikat im lib/security/cacerts-Schlüsselspeicher des JRE.
2. Erstellen Sie einen eigenen Schlüsselspeicher, indem Sie das Java-Dienstprogramm keytool verwenden und die Java-Systemeigenschaft javax.net.ssl.trustStore auf den gewünschten Speicherort einstellen. (Setzen Sie die javax.net.ssl.trustStorePassword-Methode auf einen entsprechenden Wert.)
3. Verwenden Sie den setTrustedCertificates(String)-Parameter, um auf die bereitgestellte Zertifikatdatei zu zeigen.

Um die Sicherheit zu erhöhen, sollten Sie mithilfe der Methode setCertificateName, setCertificateCompany bzw. setCertificateUnit die Validierung des MobiLink-Serverzertifikats aktivieren.

Instanzen, die diese Schnittstelle implementieren, werden von der Methode SyncParms.getStreamParms zurückgegeben, wenn das SyncParms-Objekt für die HTTPS-Synchronisation erstellt wird.

Siehe auch

- [SyncParms-Klasse \[UltraLiteJ\] auf Seite 250](#)
- [SyncParms.getStreamParms-Methode \[UltraLiteJ\] auf Seite 256](#)
- [StreamHTTPSParms.setCertificateCompany-Methode \[UltraLiteJ\] auf Seite 243](#)
- [StreamHTTPSParms.setCertificateName-Methode \[UltraLiteJ\] auf Seite 243](#)
- [StreamHTTPSParms.setCertificateUnit-Methode \[UltraLiteJ\] auf Seite 243](#)
- [StreamHTTPSParms.setTrustedCertificates-Methode \[UltraLiteJ\] auf Seite 244](#)

getCertificateCompany-Methode

Gibt den Namen der Zertifizierungsgesellschaft für die Verifizierung von sicheren Verbindungen zurück.

Syntax

```
String StreamHTTPSParms.getCertificateCompany()
```

Rückgabe

Der Name der Zertifizierungsgesellschaft

getCertificateName-Methode

Gibt den Zertifikatsnamen für die Verifizierung von sicheren Verbindungen zurück.

Syntax

```
String StreamHTTPSParms.getCertificateName()
```

Rückgabe

Der Zertifikatsname.

getCertificateUnit-Methode

Gibt den Namen der Zertifizierungseinheit für die Verifizierung von sicheren Verbindungen zurück.

Syntax

```
String StreamHTTPSParms.getCertificateUnit()
```

Rückgabe

Der Name der Organisationseinheit

getTrustedCertificates-Methode

Gibt den Namen der Datei zurück, die eine Liste von vertrauenswürdigen Stammzertifikaten enthält, die bei der sicheren Synchronisation verwendet werden.

Syntax

```
String StreamHTTPSParms.getTrustedCertificates()
```

Rückgabe

Der Dateiname der Datei mit vertrauenswürdigen Stammzertifikaten.

Siehe auch

- [StreamHTTPSParms.setTrustedCertificates-Methode \[UltraLiteJ\] auf Seite 244](#)

setCertificateCompany-Methode

Setzt den Namen der Zertifizierungsgesellschaft für die Verifizierung von sicheren Verbindungen.

Syntax

```
void StreamHTTPSParms.setCertificateCompany(String val)
```

Parameter

- **val** Der Name der Firma.

Bemerkungen

Der Standardwert ist NULL, womit angegeben wird, dass der Firmenname im Zertifikat nicht verifiziert wird.

setCertificateName-Methode

Setzt den Zertifikatsnamen für die Verifizierung von sicheren Verbindungen.

Syntax

```
void StreamHTTPSParms.setCertificateName(String val)
```

Parameter

- **val** Der Zertifikatsname.

Bemerkungen

Der Standardwert ist NULL, womit angegeben wird, dass der Zertifikatsname im Zertifikat nicht verifiziert wird.

setCertificateUnit-Methode

Setzt den Namen der Zertifizierungseinheit für die Verifizierung von sicheren Verbindungen.

Syntax

```
void StreamHTTPSParms.setCertificateUnit(String val)
```

Parameter

- **val** Der Name der Organisationseinheit.

Bemerkungen

Der Standardwert ist NULL, womit angegeben wird, dass der Name der Organisationseinheit im Zertifikat nicht verifiziert wird

setTrustedCertificates-Methode

Setzt eine Datei, die eine Liste von vertrauenswürdigen Stammzertifikaten für die sichere Synchronisation enthält.

Syntax

```
void StreamHTTPSParms.setTrustedCertificates(  
    String filename  
) throws ULjException
```

Parameter

- **filename** Der Dateiname des vertrauenswürdigen Stammzertifikats.

Bemerkungen

Diese Methode unterstützt jedes X.509-Format, das die Java-Laufzeitumgebung auf der Plattform zulässt. J2SE verwendet den JKS KeyStore-Typ zum Speichern von Stammzertifikaten, Android-Smartphones verwenden BKS KeyStore.

Diese Methode kann nur auf der J2SE-Plattform und auf Android-Smartphones verwendet werden.

Zertifikate werden gemäß den folgenden Prioritätsregeln verwendet:

1. Wenn diese Methode aufgerufen wird, werden die Zertifikate aus der angegebenen Datei verwendet.
2. Wenn diese Methode nicht aufgerufen wird und in der Datenbank mit den Dienstprogrammen ulinit oder ulload Zertifikate festgelegt wurden, werden diese Zertifikate verwendet.
3. Wenn Zertifikate weder mit dieser Methode noch mit dem Dienstprogramm ulinit oder ulload angegeben wurden und Sie Android verwenden, werden Zertifikate aus dem vertrauenswürdigen Zertifikatsspeicher des Betriebssystems gelesen. Der Zertifikatsspeicher wird von Webbrowsern bei der Verbindung mit sicheren Webservern über HTTPS verwendet.

Siehe auch

- [StreamHTTPSParms.getTrustedCertificates-Methode \[UltraLiteJ\] auf Seite 242](#)

SyncObserver-Schnittstelle

Empfängt Informationen zum Synchronisationsfortschritt

Syntax

```
public interface SyncObserver
```

Mitglieder

Alle Mitglieder der SyncObserver-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
syncProgress-Methode	Informiert den Benutzer über den Verarbeitungsfortschritt.

Bemerkungen

Erstellen Sie eine neue Klasse, die die Synchronisation ausführt, und implementieren Sie sie mithilfe der `SyncParams.setSyncObserver-Methode` so, dass sie Berichte zum Synchronisationsfortschritt empfängt.

Das folgende Beispiel zeigt eine einfache Implementierung des SyncObserver-Objekts:

```
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.
}
```

Die oben gezeigte Klasse kann mit dem folgenden Methodenaufwurf aktiviert werden:

```
SyncParams.setSyncObserver(new MyObserver());
```

Siehe auch

- [SyncParams-Klasse \[UltraLiteJ\] auf Seite 250](#)
- [SyncParams.setSyncObserver-Methode \[UltraLiteJ\] auf Seite 265](#)

syncProgress-Methode

Informiert den Benutzer über den Verarbeitungsfortschritt.

Syntax

```
boolean SyncObserver.syncProgress(int state, SyncResult data)
```

Parameter

- **state** Eine der SyncObserver.States-Konstanten, die den aktuellen Status der Synchronisation darstellen.
- **data** Ein SyncResult-Objekt, das die letzten Synchronisationsergebnisse enthält.

Rückgabe

Gibt TRUE zurück, um die Synchronisation abubrechen, ansonsten FALSE, um die Synchronisation fortzusetzen.

Bemerkungen

Diese Methode wird während der Synchronisation aufgerufen.

Die verschiedenen Status, die als Pakete signalisiert werden, werden empfangen und gesendet. Da mehrere Tabellen in einem einzigen Paket hoch- oder heruntergeladen werden können, überspringen Aufrufe dieser Methode möglicherweise eine Reihe von Zuständen.

Hinweis
Mit Ausnahme der SyncResult-Methoden sollten keine anderen Methoden der UltraLiteJ-API während eines syncProgress-Aufrufs aufgerufen werden.

Siehe auch

- [SyncObserver.States-Schnittstelle \[UltraLiteJ\]](#) auf Seite 246
- [SyncParms.setSyncObserver-Methode \[UltraLiteJ\]](#) auf Seite 265
- [SyncResult-Klasse \[UltraLiteJ\]](#) auf Seite 268

SyncObserver.States-Schnittstelle

Definiert den Synchronisationsstatus, der an einen Observer signalisiert werden kann.

Syntax

```
public interface SyncObserver.States
```

Mitglieder

Alle Mitglieder der SyncObserver.States-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
COMMITTING_DOWNLOAD-Variable	Gibt an, dass sie heruntergeladenen Zeilen in der Datenbank festgeschrieben werden.
CONNECTING-Variable	Gibt an, dass eine Synchronisation gestartet wird.
DISCONNECTING-Variable	Gibt an, dass die Verbindung des Synchronisationsdatenstroms getrennt wird.
DONE-Variable	Gibt an, dass die Synchronisation abgeschlossen ist.
ERROR-Variable	Gibt an, dass die Synchronisation zwar abgeschlossen, dabei aber ein Fehler aufgetreten ist.
FINISHING_UPLOAD-Variable	Gibt an, dass der Upload gerade abgeschlossen wird.
RECEIVING_DATA-Variable	Gibt an, dass Schemainformationen oder Zeilendaten empfangen werden.
RECEIVING_TABLE-Variable	Gibt an, dass eine neue Tabelle heruntergeladen wird.

Name	Beschreibung
RECEIVING_UPLOAD_ACK-Variable	Gibt an, dass eine Uploadbestätigung heruntergeladen wird.
ROLLING_BACK_DOWNLOAD-Variable	Gibt an, dass sie heruntergeladenen Zeilen in der Datenbank festgeschrieben werden.
SENDING_DATA-Variable	Gibt an, dass Schemainformationen oder Zeilendaten gesendet werden.
SENDING_DOWNLOAD_ACK-Variable	Gibt an, dass eine Bestätigung eines abgeschlossenen Downloads gesendet wird.
SENDING_HEADER-Variable	Gibt an, dass der Synchronisationsdatenstrom geöffnet wurde und der Header demnächst gesendet wird.
SENDING_TABLE-Variable	Gibt an, dass eine neue Tabelle hochgeladen wird.
STARTING-Variable	Gibt an, dass eine Synchronisation gestartet wird.

Siehe auch

- [SyncParms.setSyncObserver-Methode \[UltraLiteJ\] auf Seite 265](#)
- [SyncObserver-Schnittstelle \[UltraLiteJ\] auf Seite 244](#)

COMMITTING_DOWNLOAD-Variable

Gibt an, dass sie heruntergeladenen Zeilen in der Datenbank festgeschrieben werden.

Syntax

```
final int SyncObserver.States.COMMITTING_DOWNLOAD
```

CONNECTING-Variable

Gibt an, dass eine Synchronisation gestartet wird.

Syntax

```
final int SyncObserver.States.CONNECTING
```

Bemerkungen

Bisher haben keine Aktionen stattgefunden.

DISCONNECTING-Variable

Gibt an, dass die Verbindung des Synchronisationsdatenstroms getrennt wird.

Syntax

```
final int SyncObserver.States.DISCONNECTING
```

DONE-Variable

Gibt an, dass die Synchronisation abgeschlossen ist.

Syntax

```
final int SyncObserver.States.DONE
```

Bemerkungen

Es werden keine weiteren Status gemeldet.

ERROR-Variable

Gibt an, dass die Synchronisation zwar abgeschlossen, dabei aber ein Fehler aufgetreten ist.

Syntax

```
final int SyncObserver.States.ERROR
```

FINISHING_UPLOAD-Variable

Gibt an, dass der Upload gerade abgeschlossen wird.

Syntax

```
final int SyncObserver.States.FINISHING_UPLOAD
```

RECEIVING_DATA-Variable

Gibt an, dass Schemainformationen oder Zeilendaten empfangen werden.

Syntax

```
final int SyncObserver.States.RECEIVING_DATA
```

RECEIVING_TABLE-Variable

Gibt an, dass eine neue Tabelle heruntergeladen wird.

Syntax

```
final int SyncObserver.States.RECEIVING_TABLE
```

RECEIVING_UPLOAD_ACK-Variable

Gibt an, dass eine Uploadbestätigung heruntergeladen wird.

Syntax

```
final int SyncObserver.States.RECEIVING_UPLOAD_ACK
```

ROLLING_BACK_DOWNLOAD-Variable

Gibt an, dass sie heruntergeladenen Zeilen in der Datenbank festgeschrieben werden.

Syntax

```
final int SyncObserver.States.ROLLING_BACK_DOWNLOAD
```

Bemerkungen

Gibt an, dass die Synchronisation den Download zurücksetzt, weil während des Downloads ein Fehler aufgetreten ist.

SENDING_DATA-Variable

Gibt an, dass Schemainformationen oder Zeilendaten gesendet werden.

Syntax

```
final int SyncObserver.States.SENDING_DATA
```

SENDING_DOWNLOAD_ACK-Variable

Gibt an, dass eine Bestätigung eines abgeschlossenen Downloads gesendet wird.

Syntax

```
final int SyncObserver.States.SENDING_DOWNLOAD_ACK
```

SENDING_HEADER-Variable

Gibt an, dass der Synchronisationsdatenstrom geöffnet wurde und der Header demnächst gesendet wird.

Syntax

```
final int SyncObserver.States.SENDING_HEADER
```

Bemerkungen

Gibt an, dass der Synchronisationsdatenstrom geöffnet wurde und der Header demnächst gesendet wird.

SENDING_TABLE-Variable

Gibt an, dass eine neue Tabelle hochgeladen wird.

Syntax

```
final int SyncObserver.States.SENDING_TABLE
```

STARTING-Variable

Gibt an, dass eine Synchronisation gestartet wird.

Syntax

```
final int SyncObserver.States.STARTING
```

Bemerkungen

Bisher haben keine Aktionen stattgefunden.

SyncParms-Klasse

Enthält die Parameter, die während des Datenbank-Synchronisationsvorgangs verwendet werden.

Syntax

```
public class SyncParms
```

Mitglieder

Alle Mitglieder der SyncParms-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getAcknowledgeDownload-Methode	Ermittelt, ob der Client Downloadbestätigungen sendet.
getAdditionalParms-Methode [Android]	Gibt die zusätzlichen Synchronisationsparameter zurück.
getAuthenticationParms-Methode	Gibt Parameter zurück, die für ein angepasstes Benutzerauthentifizierungsskript bestimmt sind.
getKeepPartialDownload-Methode [Android]	Legt fest, ob Teil-Downloads aktiviert sind.

Name	Beschreibung
getLivenessTimeout-Methode	Gibt die Länge des Verfügbarkeits-Timeouts (in Sekunden) zurück.
getNewPassword-Methode	Gibt das neue MobiLink-Kennwort für den mit der setUsername-Methode angegebenen Benutzer zurück.
getPassword-Methode	Gibt das MobiLink-Kennwort für den mit der setUsername-Methode angegebenen Benutzer zurück.
getPublications-Methode	Gibt die zu synchronisierenden Publikationen zurück.
getResumePartialDownload-Methode [Android]	Legt fest, ob Teil-Downloads wieder aufgenommen werden.
getStreamParms-Methode	Gibt die Parameter zur Konfiguration des Synchronisationsdatenstroms zurück.
getSyncObserver-Methode	Gibt das derzeit angegebene SyncObserver-Objekt zurück.
getSyncResult-Methode	Gibt das SyncResult-Objekt zurück, das den Status der Synchronisation enthält.
getTableOrder-Methode	Gibt die Reihenfolge zurück, in der Tabellen in die konsolidierte Datenbank übertragen werden sollen.
getUserName-Methode	Gibt den MobiLink-Benutzernamen zurück, der den Client für den MobiLink-Server eindeutig kennzeichnet.
getVersion-Methode	Gibt die zu verwendende Skriptversion zurück.
isDownloadOnly-Methode	Ermittelt, ob die Synchronisation ein reiner Download ist.
isPingOnly-Methode	Ermittelt, ob der Client den MobiLink-Server anpingt oder eine Synchronisation durchführt.
isUploadOnly-Methode	Ermittelt, ob die Synchronisation ein reiner Upload ist.
setAcknowledgeDownload-Methode	Gibt an, ob der Client Downloadbestätigungen senden soll.
setAdditionalParms-Methode [Android]	Legt zusätzliche Synchronisationsparameter als semikolontrennte Liste von Paaren der Form Name=Wert fest.
setAuthenticationParms-Methode	Gibt Parameter für ein angepasstes Benutzerauthentifizierungsskript (MobiLink authenticate_parameters-Verbindungsereignis) an.
setDownloadOnly-Methode	Setzt die Synchronisation als reinen Download.

Name	Beschreibung
setKeepPartialDownload-Methode [Android]	Legt fest, ob Teil-Downloads während der Synchronisation zulässig sind.
setLivenessTimeout-Methode	Legt die Länge des Verfügbarkeits-Timeouts (in Sekunden) fest.
setNewPassword-Methode	Legt ein neues MobiLink-Kennwort für den mit der setUsername-Methode angegebenen Benutzer fest.
setPassword-Methode	Setzt das MobiLink-Kennwort für den mit der setUsername-Methode angegebenen Benutzer.
setPingOnly-Methode	Legt fest, dass der Client den MobiLink-Server anpingt, statt eine Synchronisation durchzuführen.
setPublications-Methode	Setzt die zu synchronisierenden Publikationen.
setResumePartialDownload-Methode [Android]	Legt fest, ob ein früherer Teil-Download wieder aufgenommen oder entfernt werden soll.
setSyncObserver-Methode	Setzt ein SyncObserver-Objekt, um den Verarbeitungsfortschritt der Synchronisation zu überwachen.
setTableOrder-Methode	Setzt die Reihenfolge, in der Tabellen in die konsolidierte Datenbank übertragen werden sollen.
setUploadOnly-Methode	Setzt die Synchronisation als reinen Upload.
setUserName-Methode	Setzt den MobiLink-Benutzernamen, der den Client für den MobiLink-Server eindeutig kennzeichnet.
setVersion-Methode	Setzt das zu verwendende Synchronisationsskript.
HTTP_STREAM-Variable	Erstellt ein SyncParms-Objekt für HTTP-Synchronisationen.
HTTPS_STREAM-Variable	Erstellt ein SyncParms-Objekt für sichere HTTPS-Synchronisationen.

Bemerkungen

Diese Schnittstelle wird mit der Connection.createSyncParms-Methode aufgerufen.

Sie können jeweils nur einen Synchronisationsbefehl zur gleichen Zeit setzen. Diese Befehle werden unter Verwendung der setDownloadOnly-, setPingOnly- und setUploadOnly-Methoden angegeben. Indem Sie eine dieser Methoden auf TRUE setzen, setzen Sie die anderen Methoden auf FALSE.

Die Parameter UserName und Version müssen gesetzt sein. Der UserName muss für jede Client-Datenbank eindeutig sein.

Der Kommunikationsdatenstrom wird konfiguriert, indem die `getStreamParms`-Methode, basierend auf dem Typ des `SyncParms`-Objekts, verwendet wird. Der folgende Code z.B. bereitet eine HTTP-Synchronisation vor und führt sie aus:

```
SyncParms syncParms = myConnection.createSyncParms(
    SyncParms.HTTP_STREAM,
    "MyUniqueMLUserID",
    "MyMLScriptVersion"
);
syncParms.setPassword( "ThePWDforMyUniqueMLUserID" );
syncParms.getStreamParms().setHost( "MyMLHost" );
myConnection.synchronize( syncParms );
```

Kommagetrennte Liste

Die Parameter `AuthenticationParms`, `Publications` und `TableOrder` werden unter Verwendung eines Zeichenfolgenwerts angegeben, der eine durch Kommas getrennte Werteliste enthält. Werte in der Liste können in Apostrophe oder Anführungszeichen gesetzt sein, es gibt aber keine Escapezeichen. Führende und nachgestellte Leerzeichen in Werten werden ignoriert, außer sie sind in Anführungszeichen gesetzt. Beispiel: Der folgende Code gibt erst **Table A**, dann **Table B,D** und dann **Table C** an:

```
syncParms.setTableOrder( "'Table A',\"Table B,D\",Table C );
```

Siehe auch

- [SyncParms.getStreamParms-Methode \[UltraLiteJ\] auf Seite 256](#)
- [SyncParms.setUserName-Methode \[UltraLiteJ\] auf Seite 267](#)
- [Connection.createSyncParms-Methode \[UltraLiteJ\] auf Seite 112](#)
- [StreamHTTPParms-Schnittstelle \[UltraLiteJ\] auf Seite 225](#)
- [StreamHTTPSPParms-Schnittstelle \[UltraLiteJ\] auf Seite 238](#)

getAcknowledgeDownload-Methode

Ermittelt, ob der Client Downloadbestätigungen sendet.

Syntax

```
abstract boolean SyncParms.getAcknowledgeDownload()
```

Rückgabe

TRUE, wenn der Client Downloadbestätigungen sendet, ansonsten FALSE.

Siehe auch

- [SyncParms.setAcknowledgeDownload-Methode \[UltraLiteJ\] auf Seite 259](#)

getAdditionalParms-Methode [Android]

Gibt die zusätzlichen Synchronisationsparameter zurück.

Syntax

```
abstract String SyncParms.getAdditionalParms()
```

Rückgabe

Die Liste der zusätzlichen Parameter oder NULL, wenn keine Parameter angegeben sind.

Siehe auch

- [SyncParms.setAdditionalParms-Methode \[Android\] \[UltraLiteJ\] auf Seite 259](#)

getAuthenticationParms-Methode

Gibt Parameter zurück, die für ein angepasstes Benutzerauthentifizierungsskript bestimmt sind.

Syntax

```
abstract String SyncParms.getAuthenticationParms()
```

Rückgabe

Die Liste der Authentifizierungsparameter oder NULL, wenn keine Parameter angegeben sind.

Siehe auch

- [SyncParms.setAuthenticationParms-Methode \[UltraLiteJ\] auf Seite 260](#)

getKeepPartialDownload-Methode [Android]

Legt fest, ob Teil-Downloads aktiviert sind.

Syntax

```
abstract boolean SyncParms.getKeepPartialDownload()
```

Rückgabe

TRUE, wenn Teil-Downloads aktiviert sind, ansonsten FALSE.

Siehe auch

- [SyncParms.setKeepPartialDownload-Methode \[Android\] \[UltraLiteJ\] auf Seite 261](#)

getLivenessTimeout-Methode

Gibt die Länge des Verfügbarkeits-Timeouts (in Sekunden) zurück.

Syntax

```
abstract int SyncParms.getLivenessTimeout()
```

Rückgabe

Der Timeout.

Siehe auch

- [SyncParms.setLivenessTimeout-Methode \[UltraLiteJ\] auf Seite 262](#)

getNewPassword-Methode

Gibt das neue MobiLink-Kennwort für den mit der setUsername-Methode angegebenen Benutzer zurück.

Syntax

```
abstract String SyncParms.getNewPassword()
```

Rückgabe

Das neue Kennwort, das nach der nächsten Synchronisation gesetzt wird.

Siehe auch

- [SyncParms.setUsername-Methode \[UltraLiteJ\] auf Seite 267](#)
- [SyncParms.setNewPassword-Methode \[UltraLiteJ\] auf Seite 263](#)

getPassword-Methode

Gibt das MobiLink-Kennwort für den mit der setUsername-Methode angegebenen Benutzer zurück.

Syntax

```
abstract String SyncParms.getPassword()
```

Rückgabe

Das Kennwort für den MobiLink-Benutzer.

Siehe auch

- [SyncParms.setPassword-Methode \[UltraLiteJ\] auf Seite 263](#)

getPublications-Methode

Gibt die zu synchronisierenden Publikationen zurück.

Syntax

```
abstract String SyncParms.getPublications()
```

Rückgabe

Die Gruppe von zu synchronisierenden Publikationen.

Siehe auch

- [SyncParms.setPublications-Methode \[UltraLiteJ\] auf Seite 264](#)

getResumePartialDownload-Methode [Android]

Legt fest, ob Teil-Downloads wieder aufgenommen werden.

Syntax

```
abstract boolean SyncParams.getResumePartialDownload()
```

Rückgabe

TRUE, wenn Teil-Downloads wieder aufgenommen werden sollen, ansonsten FALSE.

Siehe auch

- [SyncParams.setResumePartialDownload-Methode \[Android\] \[UltraLiteJ\] auf Seite 265](#)

getStreamParams-Methode

Gibt die Parameter zur Konfiguration des Synchronisationsdatenstroms zurück.

Syntax

```
abstract StreamHTTPParams SyncParams.getStreamParams()
```

Rückgabe

Ein StreamHTTPParams- oder StreamHTTPSPParams-Objekt, das die Parameter für die HTTP- bzw. HTTPS-Synchronisationsdatenströme angibt. Das Objekt wird mittels Referenz zurückgegeben.

Bemerkungen

Der Typ des Synchronisationsdatenstroms wird angegeben, wenn das SyncParams-Objekt erstellt wird.

Siehe auch

- [Connection.createSyncParams-Methode \[UltraLiteJ\] auf Seite 112](#)
- [StreamHTTPParams-Schnittstelle \[UltraLiteJ\] auf Seite 225](#)
- [StreamHTTPSPParams-Schnittstelle \[UltraLiteJ\] auf Seite 238](#)

getSyncObserver-Methode

Gibt das derzeit angegebene SyncObserver-Objekt zurück.

Syntax

```
abstract SyncObserver SyncParams.getSyncObserver()
```

Rückgabe

Das SyncObserver-Objekt oder NULL, wenn kein Observer angegeben wurde.

Siehe auch

- [SyncParams.setSyncObserver-Methode \[UltraLiteJ\] auf Seite 265](#)

getSyncResult-Methode

Gibt das SyncResult-Objekt zurück, das den Status der Synchronisation enthält.

Syntax

```
abstract SyncResult SyncParms.getSyncResult()
```

Rückgabe

Das SyncResult-Objekt, das das Ergebnis des letzten Aufrufs der Connection.synchronize-Methode darstellt.

Bemerkungen

Das folgende Beispiel zeigt, wie Sie die Ergebnismenge des letzten Aufrufs der Connection.synchronize-Methode abrufen:

```
conn.synchronize( mySyncParms );
SyncResult result = mySyncParms.getSyncResult();
display(
    "*** Synchronized *** sent=" + result.getSentRowCount()
    + ", received=" + result.getReceivedRowCount()
);
```

Hinweis

Diese Methode gibt nicht das Ergebnis der letzten SYNCHRONIZE SQL-Anweisung zurück. Um das SyncResult-Objekt für die letzte SYNCHRONIZE SQL-Anweisung abzurufen, wenden Sie die getSyncResult-Methode auf das übergebene Connection-Objekt an.

Siehe auch

- [SyncResult-Klasse \[UltraLiteJ\] auf Seite 268](#)
- [Connection.getSyncResult-Methode \[UltraLiteJ\] auf Seite 117](#)

getTableOrder-Methode

Gibt die Reihenfolge zurück, in der Tabellen in die konsolidierte Datenbank übertragen werden sollen.

Syntax

```
abstract String SyncParms.getTableOrder()
```

Rückgabe

Eine kommasetrennte Liste von Tabellennamen, andernfalls NULL, wenn keine Tabellenfolge angegeben wurde. Weitere Hinweise zu kommasetrennten Listen finden Sie in der Klassenbeschreibung.

Siehe auch

- [SyncParms.setTableOrder-Methode \[UltraLiteJ\] auf Seite 266](#)

getUserName-Methode

Gibt den MobiLink-Benutzernamen zurück, der den Client für den MobiLink-Server eindeutig kennzeichnet.

Syntax

```
abstract String SyncParams.getUserName()
```

Rückgabe

Der MobiLink-Benutzername

Siehe auch

- [SyncParams.setUserName-Methode \[UltraLiteJ\] auf Seite 267](#)

getVersion-Methode

Gibt die zu verwendende Skriptversion zurück.

Syntax

```
abstract String SyncParams.getVersion()
```

Rückgabe

Die Skriptversion

Siehe auch

- [SyncParams.setVersion-Methode \[UltraLiteJ\] auf Seite 267](#)

isDownloadOnly-Methode

Ermittelt, ob die Synchronisation ein reiner Download ist.

Syntax

```
abstract boolean SyncParams.isDownloadOnly()
```

Rückgabe

TRUE, wenn Uploads deaktiviert sind, ansonsten FALSE.

Siehe auch

- [SyncParams.setDownloadOnly-Methode \[UltraLiteJ\] auf Seite 261](#)

isPingOnly-Methode

Ermittelt, ob der Client den MobiLink-Server anpingt oder eine Synchronisation durchführt.

Syntax

```
abstract boolean SyncParms.isPingOnly()
```

Rückgabe

TRUE, wenn der Client den Server nur anpingt, ansonsten FALSE.

Siehe auch

- [SyncParms.setPingOnly-Methode \[UltraLiteJ\] auf Seite 264](#)

isUploadOnly-Methode

Ermittelt, ob die Synchronisation ein reiner Upload ist.

Syntax

```
abstract boolean SyncParms.isUploadOnly()
```

Rückgabe

TRUE, wenn Downloads deaktiviert sind, ansonsten FALSE.

Siehe auch

- [SyncParms.setUploadOnly-Methode \[UltraLiteJ\] auf Seite 266](#)

setAcknowledgeDownload-Methode

Gibt an, ob der Client Downloadbestätigungen senden soll.

Syntax

```
abstract void SyncParms.setAcknowledgeDownload(boolean ack)
```

Parameter

- **ack** Setzen Sie dies auf TRUE, damit der Client einen Download bestätigt, ansonsten auf FALSE.

Bemerkungen

Der Standardwert ist FALSE.

Siehe auch

- [SyncParms.getAcknowledgeDownload-Methode \[UltraLiteJ\] auf Seite 253](#)

setAdditionalParms-Methode [Android]

Legt zusätzliche Synchronisationsparameter als semikolongetrennte Liste von Paaren der Form Name=Wert fest.

Syntax

```
abstract void SyncParams.setAdditionalParams(String v) throws ULjException
```

Parameter

- **v** Eine Zeichenfolge in der Form einer durch Semikola getrennten Liste von Name=Wert-Paaren.

Bemerkungen

Verwenden Sie diese Methode, um mehrere zusätzliche Synchronisationsparameter anzugeben, die mit vorhandenen Methoden der SyncParams-Klasse nicht angegeben werden können.

Das folgende Beispiel zeigt, wie die Parameter AllowDownloadDupRows, CheckpointStore und DisableConcurrency für ein SyncParams-Objekt festgelegt werden:

```
SyncParams params;  
...  
params.setAdditionalParams(  
    "AllowDownloadDupRows=1;CheckpointStore=1;DisableConcurrency=1" );
```

Siehe auch

- [SyncParams.getAdditionalParams-Methode \[Android\] \[UltraLiteJ\] auf Seite 253](#)

setAuthenticationParams-Methode

Legt Parameter für ein angepasstes Benutzerauthentifizierungsskript (MobiLink authenticate_parameters-Verbindungsereignis) fest.

Syntax

```
abstract void SyncParams.setAuthenticationParams(  
    String v  
) throws ULjException
```

Parameter

- **v** Eine durch Kommas getrennte Liste von Authentifizierungsparametern oder die Nullreferenz. Weitere Hinweise zu kommasetrennten Listen finden Sie in der Klassenbeschreibung.

Bemerkungen

Nur die ersten 255 Zeichenfolgen werden verwendet und keine Zeichenfolge darf die Grenze des MobiLink-Servers für Authentifizierungsparameter überschreiten (derzeit 4000 Byte UTF8).

Zeichenfolgen, die länger sind als 21K Zeichen, werden beim Senden an MobiLink gekürzt und Zeichenfolgen, die die Grenze des Servers für Authentifizierungsparameter überschreiten, verursachen einen serverseitigen Synchronisationsfehler.

Siehe auch

- [SyncParams.getAuthenticationParams-Methode \[UltraLiteJ\] auf Seite 254](#)

setDownloadOnly-Methode

Setzt die Synchronisation als reinen Download.

Syntax

```
abstract void SyncParms.setDownloadOnly(boolean v)
```

Parameter

- **v** Setzen Sie dies auf TRUE, um Uploads zu deaktivieren, ansonsten auf FALSE.

Bemerkungen

Der Standardwert ist FALSE. Durch Angeben von TRUE werden automatisch die Methoden setPingOnly und setUploadOnly aufgerufen und auf FALSE gesetzt.

Siehe auch

- [SyncParms.isDownloadOnly-Methode \[UltraLiteJ\] auf Seite 258](#)
- [SyncParms.setPingOnly-Methode \[UltraLiteJ\] auf Seite 264](#)
- [SyncParms.setUploadOnly-Methode \[UltraLiteJ\] auf Seite 266](#)

setKeepPartialDownload-Methode [Android]

Legt fest, ob Teil-Downloads während der Synchronisation zulässig sind.

Syntax

```
abstract void SyncParms.setKeepPartialDownload(
    boolean c
) throws ULjException
```

Parameter

- **c** Setzen Sie diese Option auf TRUE, um Teil-Downloads zu aktivieren.

Bemerkungen

Die Standardeinstellung ist FALSE. Setzen Sie diese Option auf TRUE, um Teil-Downloads während der Synchronisation zu aktivieren und zu speichern. Setzen Sie die Option auf FALSE, um Teil-Downloads zu deaktivieren und zurückzusetzen, wenn ein Fehler auftritt.

UltraLite hat die Möglichkeit, Teil-Downloads wieder aufzunehmen, die aufgrund von Kommunikationsfehlern oder Abbruch durch den Benutzer über das SyncObserver-Objekt fehlschlagen. UltraLite verarbeitet den Download so, wie er empfangen wird. Wenn ein Download unterbrochen wird, bleibt die Teil-Download-Transaktion in der Datenbank und kann bei der nächsten Synchronisation wieder aufgenommen werden.

Um festzulegen, dass UltraLite Teil-Downloads speichern soll, setzen Sie die Option auf TRUE. Andernfalls wird der Download zurückgesetzt, wenn ein Fehler auftritt.

Falls ein Teil-Download aufbewahrt wurde, gibt die SyncResult.getPartialDownloadRetained-Methode TRUE zurück, wenn die Connection.synchronize-Methode beendet wird.

Wenn der KeepPartialDownload-Synchronisationsparameter auf TRUE gesetzt ist, können Sie einen Teil-Download wieder aufnehmen. Um einen Teil-Download wieder aufzunehmen, rufen Sie die Connection.synchronize-Methode auf, wobei die setResumePartialDownload-Methode auf TRUE gesetzt sein muss.

Es wird empfohlen, den Wert TRUE für den KeepPartialDownload-Parameter für den Fall eines weiteren Kommunikationsfehlers beizubehalten. Wird ein Download übersprungen, so wird kein Upload ausgeführt.

Der Download, den Sie während eines wiederaufgenommenen Downloads empfangen, ist so alt wie der Download, der ursprünglich begonnen wurde. Wenn Sie die aktuellen Daten benötigen, können Sie unmittelbar nach Abschluss des wiederaufgenommenen Downloads einen weiteren Download ausführen.

Bei der Wiederaufnahme eines Downloads sind zahlreiche Synchronisationsparameter, die von der SyncParms-Klasse angegeben werden, nicht relevant. Der Publications-Parameter wird z.B. nicht verwendet. Sie erhalten die angeforderten Publikationen während des ursprünglichen Downloads. Nur die setResumePartialDownload-Methode und die setUsername-Methode müssen verwendet werden. Die setKeepPartialDownload-Methode kann benutzt werden, falls gewünscht.

Wenn ein Teil-Download existiert und nicht mehr benötigt wird, können Sie Connection.rollbackPartialDownload aufrufen, um die fehlgeschlagene Download-Transaktion zurückzusetzen. Wenn Sie versuchen, erneut zu synchronisieren, ohne den ResumePartialDownload-Parameter festzulegen, wird der Teil-Download zurückgesetzt, bevor die nächste Synchronisation beginnt.

Siehe auch

- [SyncParms.getKeepPartialDownload-Methode \[Android\] \[UltraLiteJ\] auf Seite 254](#)
- [SyncParms.setResumePartialDownload-Methode \[Android\] \[UltraLiteJ\] auf Seite 265](#)
- [SyncParms.setUsername-Methode \[UltraLiteJ\] auf Seite 267](#)
- [„Wiederaufnahme fehlgeschlagener Downloads“ \[MobiLink - Serveradministration\]](#)

setLivenessTimeout-Methode

Legt die Länge des Verfügbarkeits-Timeouts (in Sekunden) fest.

Syntax

```
abstract void SyncParms.setLivenessTimeout(  
    int seconds  
) throws ULjException
```

Parameter

- **seconds** Der neue Wert des Verfügbarkeits-Timeouts

Bemerkungen

Der Verfügbarkeits-Timeout ist die Zeit, die der Datenbankserver einer entfernten Datenbank erlaubt, inaktiv zu sein. Wenn die entfernte Datenbank nicht 1 Sekunde mit dem Datenbankserver kommuniziert, nimmt der Server an, dass die Verbindung zur entfernten Datenbank unterbrochen wurde, und beendet die

Synchronisation. Die entfernte Datenbank sendet automatisch periodische Meldungen an den Datenbankserver, um die Verbindung aufrecht zu halten.

Wenn ein negativer Wert gesetzt wird, wird eine Ausnahmebedingung ausgegeben. Der Wert kann vom MobiLink-Server ohne Ankündigung geändert werden. Diese Änderung tritt auf, wenn der Wert zu hoch oder zu niedrig eingestellt ist.

Der Standardwert ist 100 Sekunden für BlackBerry-/J2SE-Plattformen und 240 Sekunden für Android-Plattformen.

Siehe auch

- [SyncParams.getLivenessTimeout-Methode \[UltraLiteJ\] auf Seite 254](#)

setNewPassword-Methode

Legt ein neues MobiLink-Kennwort für den mit der setUsername-Methode angegebenen Benutzer fest.

Syntax

```
abstract void SyncParams.setNewPassword(String v)
```

Parameter

- **v** Ein neues Kennwort für einen MobiLink-Benutzer.

Bemerkungen

Das neue Kennwort wird nach der nächsten Synchronisation wirksam.

Der Standardwert ist NULL, was anzeigt, dass das Kennwort nicht ersetzt wird.

Siehe auch

- [SyncParams.getNewPassword-Methode \[UltraLiteJ\] auf Seite 255](#)
- [SyncParams.setPassword-Methode \[UltraLiteJ\] auf Seite 263](#)
- [SyncParams.setUsername-Methode \[UltraLiteJ\] auf Seite 267](#)

setPassword-Methode

Setzt das MobiLink-Kennwort für den mit der setUsername-Methode angegebenen Benutzer.

Syntax

```
abstract void SyncParams.setPassword(String v) throws ULJException
```

Parameter

- **v** Ein Kennwort für den MobiLink-Benutzer.

Bemerkungen

Dieser Benutzername und das Kennwort unterscheiden sich von anderen Datenbankbenutzer-IDs und Kennwörtern. Die Methode wird verwendet, um die Anwendung gegenüber dem MobiLink-Server zu authentifizieren.

Der Standardwert ist eine leere Zeichenfolge, womit kein Kennwort angegeben wird.

Siehe auch

- [SyncParms.getPassword-Methode \[UltraLiteJ\] auf Seite 255](#)
- [SyncParms.setNewPassword-Methode \[UltraLiteJ\] auf Seite 263](#)
- [SyncParms.setUserName-Methode \[UltraLiteJ\] auf Seite 267](#)

setPingOnly-Methode

Legt fest, dass der Client den MobiLink-Server anpingt, statt eine Synchronisation durchzuführen.

Syntax

```
abstract void SyncParms.setPingOnly(boolean v)
```

Parameter

- **v** Setzen Sie dies auf TRUE, um den Server nur anzupingen, oder auf FALSE, um eine Synchronisation durchzuführen.

Bemerkungen

Der Standardwert ist FALSE. Durch Angeben von TRUE werden automatisch die Methoden setDownloadOnly und setUploadOnly aufgerufen und auf FALSE gesetzt.

Siehe auch

- [SyncParms.isPingOnly-Methode \[UltraLiteJ\] auf Seite 258](#)
- [SyncParms.setDownloadOnly-Methode \[UltraLiteJ\] auf Seite 261](#)
- [SyncParms.setUploadOnly-Methode \[UltraLiteJ\] auf Seite 266](#)

setPublications-Methode

Setzt die zu synchronisierenden Publikationen.

Syntax

```
abstract void SyncParms.setPublications(String pubs) throws ULjException
```

Parameter

- **pubs** Eine durch Kommas getrennte Liste von Publikationsnamen. Weitere Hinweise zu kommagetrennten Listen finden Sie in der Klassenbeschreibung.

Bemerkungen

Der Standardwert ist auf die `Connection.SYNC_ALL`-Konstante gesetzt, die verwendet wird, um die Synchronisation aller Tabellen in der Datenbank zu bezeichnen. Um alle Publikationen zu synchronisieren, setzen Sie diese Methode auf die `Connection.SYNC_ALL_PUBS`-Konstante.

Siehe auch

- [SyncParams.getPublications-Methode \[UltraLiteJ\] auf Seite 255](#)
- [Connection.SYNC_ALL-Variable \[UltraLiteJ\] auf Seite 131](#)
- [Connection.SYNC_ALL_PUBS-Variable \[UltraLiteJ\] auf Seite 132](#)

setResumePartialDownload-Methode [Android]

Legt fest, ob ein früherer Teil-Download wieder aufgenommen oder entfernt werden soll.

Syntax

```
abstract void SyncParams.setResumePartialDownload(  
    boolean c  
) throws ULjException
```

Parameter

- **c** Setzen Sie dies auf TRUE, um einen früheren Teil-Download wieder aufzunehmen.

Ausnahmen

- **ULjException-Klasse** `SQLE_SYNC_INFO_INVALID` wird von der `Connection.synchronize`-Methode generiert, wenn mehr als einer der folgenden Synchronisationsparameter (`DownloadOnly`, `PingOnly`, `ResumePartialDownload` oder `UploadOnly`) auf TRUE gesetzt ist.

Bemerkungen

Setzen Sie dies auf TRUE, wenn ein früherer Teil-Download wieder aufgenommen werden soll, oder auf FALSE, wenn er verworfen werden soll. Der Standardwert ist FALSE.

Siehe auch

- [SyncParams.getResumePartialDownload-Methode \[Android\] \[UltraLiteJ\] auf Seite 256](#)

setSyncObserver-Methode

Setzt ein `SyncObserver`-Objekt, um den Verarbeitungsfortschritt der Synchronisation zu überwachen.

Syntax

```
abstract void SyncParams.setSyncObserver(SyncObserver so)
```

Parameter

- **so** Ein `SyncObserver`-Objekt.

Bemerkungen

Der Standardwert ist NULL, womit kein Observer angegeben wird.

Siehe auch

- [SyncObserver-Schnittstelle \[UltraLiteJ\] auf Seite 244](#)

setTableOrder-Methode

Setzt die Reihenfolge, in der Tabellen in die konsolidierte Datenbank übertragen werden sollen.

Syntax

```
abstract void SyncParams.setTableOrder(String v) throws ULJException
```

Parameter

- **v** Eine durch Kommas getrennte Liste von Tabellennamen in der Reihenfolge, in der sie synchronisiert werden sollen, oder NULL, womit keine Tabellenfolge angegeben wird. Weitere Hinweise zu kommagetrennten Listen finden Sie in der Klassenbeschreibung.

Bemerkungen

Die Primärtabelle sollte zuerst aufgelistet sein, zusammen mit allen Tabellen, die Fremdschlüsselbeziehungen in der konsolidierten Datenbank enthalten.

Alle durch den Publications-Parameter für die Synchronisation ausgewählten Tabellen werden synchronisiert, unabhängig davon, ob sie im TableOrder-Parameter angegeben sind. Nicht angegebene Tabellen werden in der Reihenfolge der Fremdschlüsselbeziehungen in der Client-Datenbank synchronisiert. Sie werden nach den angegebenen Tabellen synchronisiert.

Der Standardwert ist eine Nullreferenz, die die Standardreihenfolge der Tabellen nicht außer Kraft setzt.

Siehe auch

- [SyncParams.getTableOrder-Methode \[UltraLiteJ\] auf Seite 257](#)
- [SyncParams.setPublications-Methode \[UltraLiteJ\] auf Seite 264](#)

setUploadOnly-Methode

Setzt die Synchronisation als reinen Upload.

Syntax

```
abstract void SyncParams.setUploadOnly(boolean v)
```

Parameter

- **v** Setzen Sie dies auf TRUE, um Downloads zu deaktivieren, ansonsten auf FALSE.

Bemerkungen

Der Standardwert ist FALSE. Durch Angeben von TRUE werden automatisch die Methoden `setDownloadOnly` und `setPingOnly` aufgerufen und auf FALSE gesetzt.

Siehe auch

- [SyncParams.isUploadOnly-Methode \[UltraLiteJ\] auf Seite 259](#)
- [SyncParams.setDownloadOnly-Methode \[UltraLiteJ\] auf Seite 261](#)
- [SyncParams.setPingOnly-Methode \[UltraLiteJ\] auf Seite 264](#)

setUserName-Methode

Setzt den MobiLink-Benutzernamen, der den Client für den MobiLink-Server eindeutig kennzeichnet.

Syntax

```
abstract void SyncParams.setUserName(String v) throws ULjException
```

Parameter

- **v** Der MobiLink-Benutzername

Bemerkungen

Dieser Wert wird verwendet, um Folgendes festzulegen:

- Download-Inhalt
- Aufzeichnung des Synchronisationsstatus
- Wiederherstellung nach Unterbrechungen während der Synchronisation

Dieser Benutzername und das Kennwort unterscheiden sich von anderen Datenbankbenutzer-IDs und Kennwörtern. Die Methode wird verwendet, um die Anwendung gegenüber dem MobiLink-Server zu authentifizieren.

Dieser Parameter wird initialisiert, wenn das SyncParams-Objekt erstellt wird.

Siehe auch

- [SyncParams.getUserName-Methode \[UltraLiteJ\] auf Seite 258](#)
- [SyncParams.setPassword-Methode \[UltraLiteJ\] auf Seite 263](#)
- [SyncParams.setNewPassword-Methode \[UltraLiteJ\] auf Seite 263](#)
- [Connection.createSyncParams-Methode \[UltraLiteJ\] auf Seite 112](#)

setVersion-Methode

Setzt das zu verwendende Synchronisationsskript.

Syntax

```
abstract void SyncParams.setVersion(String v) throws ULjException
```

Parameter

- **v** Die Skriptversion

Bemerkungen

Jedes Synchronisationsskript in der konsolidierten Datenbank wird mit einer Versionszeichenfolge markiert. Wenn es z.B. zwei verschiedene `download_cursor`-Skripts gibt, wird jedes durch unterschiedliche Versionszeichenfolgen gekennzeichnet. Die Versionszeichenfolge gestattet es einer Anwendung, zwischen einer Reihe von Synchronisationsskripten zu wählen.

Dieser Parameter wird initialisiert, wenn das `SyncParms`-Objekt erstellt wird.

Siehe auch

- [SyncParms.getVersion-Methode \[UltraLiteJ\] auf Seite 258](#)
- [Connection.createSyncParms-Methode \[UltraLiteJ\] auf Seite 112](#)

HTTP_STREAM-Variable

Erstellt ein `SyncParms`-Objekt für HTTP-Synchronisationen.

Syntax

```
final int SyncParms.HTTP_STREAM
```

Siehe auch

- [Connection.createSyncParms-Methode \[UltraLiteJ\] auf Seite 112](#)

HTTPS_STREAM-Variable

Erstellt ein `SyncParms`-Objekt für sichere HTTPS-Synchronisationen.

Syntax

```
final int SyncParms.HTTPS_STREAM
```

Siehe auch

- [Connection.createSyncParms-Methode \[UltraLiteJ\] auf Seite 112](#)

SyncResult-Klasse

Meldet statusbezogene Informationen zu einer angegebenen Datenbanksynchronisation.

Syntax

```
public class SyncResult
```

Mitglieder

Alle Mitglieder der SyncResult-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getAuthMessage-Methode	Gibt die Autorisierungsnachricht des letzten Synchronisationsversuchs zurück, wie von den Synchronisationsskripten der Benutzerauthentifizierung angegeben.
getAuthStatus-Methode	Gibt den Autorisierungsstatuscode des letzten Synchronisationsversuchs zurück.
getAuthValue-Methode	Gibt den Wert zurück, der in Synchronisationsskripten mit angepasster Benutzerauthentifizierung angegeben ist.
getCurrentTableName-Methode	Gibt den Namen der Tabelle zurück, die derzeit synchronisiert wird.
getIgnoredRows-Methode	Ermittelt, ob ausgelesene Zeilen bei der letzten Synchronisation ignoriert wurden.
getPartialDownloadRetained-Methode [Android]	Prüft, ob ein Teil-Download bei der letzten Synchronisation bewahrt wurde.
getReceivedByteCount-Methode	Gibt die Anzahl der Bytes zurück, die während einer Datensynchronisation empfangen wurden.
getReceivedDeletes-Methode	Gibt die Anzahl der empfangenen Zeilen zurück, die gelöscht wurden.
getReceivedIgnoredDeletes-Methode	Gibt die Anzahl der empfangenen gelöschten Zeilen zurück, die ignoriert wurden.
getReceivedIgnoredUpdates-Methode	Gibt die Anzahl der empfangenen aktualisierten Zeilen zurück, die ignoriert wurden.
getReceivedInserts-Methode	Gibt die Anzahl der empfangenen Zeilen zurück, die eingefügt wurden.
getReceivedRowCount-Methode	Gibt die Anzahl der empfangenen Zeilen zurück.
getReceivedTruncateDeletes-Methode	Gibt die Anzahl der Zeilen zurück, die durch einen empfangenen Kürzungsvorgang gelöscht wurden.
getReceivedUpdates-Methode	Gibt die Anzahl der empfangenen Zeilen zurück, die als Aktualisierungen übernommen wurden.
getSentByteCount-Methode	Gibt die Anzahl der Bytes zurück, die während einer Datensynchronisation gesendet wurden.

Name	Beschreibung
getSentDeletes-Methode	Gibt die Anzahl der gelöschten Zeilen zurück, die gesendet wurden.
getSentInserts-Methode	Gibt die Anzahl der eingefügten Zeilen zurück, die gesendet wurden.
getSentUpdates-Methode	Gibt die Anzahl der aktualisierten Zeilen zurück, die gesendet wurden.
getStreamErrorCode-Methode	Gibt den Fehler zurück, der vom Datenstrom selbst gemeldet wurde.
getStreamErrorMessage-Methode	Gibt den Fehler zurück, der vom Datenstrom selbst gemeldet wurde.
getSyncedTableCount-Methode	Gibt die Anzahl der bisher synchronisierten Tabellen zurück.
getTotalDownloadRowCount-Methode	Gibt die Gesamtzahl der Zeilen zurück, die im Download empfangen wurden.
getTotalTableCount-Methode	Gibt die Anzahl der zu synchronisierenden Tabellen zurück.
isUploadOK-Methode	Ermittelt, ob die letzte Upload-Synchronisation erfolgreich war.

Siehe auch

- [SyncParms.getSyncResult-Methode \[UltraLiteJ\] auf Seite 257](#)

getAuthMessage-Methode

Gibt die Autorisierungsnachricht des letzten Synchronisationsversuchs zurück, wie von den Synchronisationsskripten der Benutzerauthentifizierung angegeben.

Syntax

```
abstract String SyncResult.getAuthMessage()
```

Rückgabe

Eine Zeichenfolge mit Informationen über die Authentifizierung der letzten Synchronisation.

Bemerkungen

Leere Nachrichten werden als NULL zurückgegeben.

Für jeden Autorisierungsstatuscode kann eine Authentifizierungsnachricht zurückgegeben werden. Weitere Hinweise finden Sie bei den Details zum benannten MobiLink-Systemparameter `authentication_message`.

getAuthStatus-Methode

Gibt den Autorisierungsstatuscode des letzten Synchronisationsversuchs zurück.

Syntax

```
abstract int SyncResult.getAuthStatus()
```

Rückgabe

Ein AuthStatusCode-Wert.

getAuthValue-Methode

Gibt den Wert zurück, der in Synchronisationsskripten mit angepasster Benutzerauthentifizierung angegeben ist.

Syntax

```
abstract int SyncResult.getAuthValue()
```

Rückgabe

Eine Ganzzahl, die aus Synchronisationsskripten mit angepasster Benutzerauthentifizierung zurückgegeben wird.

getCurrentTableName-Methode

Gibt den Namen der Tabelle zurück, die derzeit synchronisiert wird.

Syntax

```
abstract String SyncResult.getCurrentTableName()
```

Rückgabe

Der Tabellenname.

getIgnoredRows-Methode

Ermittelt, ob ausgelesene Zeilen bei der letzten Synchronisation ignoriert wurden.

Syntax

```
abstract boolean SyncResult.getIgnoredRows()
```

Rückgabe

TRUE, wenn hochgeladene Zeilen bei der letzten Synchronisation ignoriert wurden, oder FALSE, wenn keine Zeilen ignoriert wurden.

getPartialDownloadRetained-Methode [Android]

Prüft, ob ein Teil-Download bei der letzten Synchronisation bewahrt wurde.

Syntax

```
abstract boolean SyncResult.getPartialDownloadRetained()
```

Rückgabe

TRUE, wenn ein Download unterbrochen wurde und der Teil-Download bewahrt wurde. FALSE, wenn der Download nicht unterbrochen oder der Teil-Download zurückgesetzt wurde.

getReceivedByteCount-Methode

Gibt die Anzahl der Bytes zurück, die während einer Datensynchronisation empfangen wurden.

Syntax

```
abstract long SyncResult.getReceivedByteCount()
```

Rückgabe

Die Anzahl der Bytes.

getReceivedDeletes-Methode

Gibt die Anzahl der empfangenen Zeilen zurück, die gelöscht wurden.

Syntax

```
abstract long SyncResult.getReceivedDeletes()
```

Rückgabe

Die Anzahl der heruntergeladenen Zeilen, die als Löschungen angewendet wurden.

Siehe auch

- [SyncResult.getReceivedIgnoredDeletes-Methode \[UltraLiteJ\] auf Seite 272](#)
- [SyncResult.getReceivedIgnoredUpdates-Methode \[UltraLiteJ\] auf Seite 273](#)
- [SyncResult.getReceivedInserts-Methode \[UltraLiteJ\] auf Seite 273](#)
- [SyncResult.getReceivedTruncateDeletes-Methode \[UltraLiteJ\] auf Seite 274](#)
- [SyncResult.getReceivedUpdates-Methode \[UltraLiteJ\] auf Seite 275](#)

getReceivedIgnoredDeletes-Methode

Gibt die Anzahl der empfangenen gelöschten Zeilen zurück, die ignoriert wurden.

Syntax

```
abstract long SyncResult.getReceivedIgnoredDeletes()
```

Rückgabe

Gesamtanzahl der gelöschten Download-Zeilen, die ignoriert wurden.

Siehe auch

- [SyncResult.getReceivedIgnoredUpdates-Methode \[UltraLiteJ\] auf Seite 273](#)
- [SyncResult.getReceivedDeletes-Methode \[UltraLiteJ\] auf Seite 272](#)
- [SyncResult.getReceivedInserts-Methode \[UltraLiteJ\] auf Seite 273](#)
- [SyncResult.getReceivedTruncateDeletes-Methode \[UltraLiteJ\] auf Seite 274](#)
- [SyncResult.getReceivedUpdates-Methode \[UltraLiteJ\] auf Seite 275](#)

getReceivedIgnoredUpdates-Methode

Gibt die Anzahl der empfangenen aktualisierten Zeilen zurück, die ignoriert wurden.

Syntax

```
abstract long SyncResult.getReceivedIgnoredUpdates()
```

Rückgabe

Gesamtanzahl der heruntergeladenen aktualisierten Zeilen, die ignoriert wurden. Für UltraLite Java Edition-Datenbanken ist dieser Wert immer "0".

Bemerkungen

Empfangene aktualisierte Zeilen werden nur dann ignoriert, wenn doppelte Primärschlüssel im Download erlaubt sind (was nur auf Android möglich ist). Doppelte heruntergeladene aktualisierte Zeilen führen sonst zu einer fehlgeschlagenen Synchronisation.

Siehe auch

- [SyncResult.getReceivedIgnoredDeletes-Methode \[UltraLiteJ\] auf Seite 272](#)
- [SyncResult.getReceivedDeletes-Methode \[UltraLiteJ\] auf Seite 272](#)
- [SyncResult.getReceivedInserts-Methode \[UltraLiteJ\] auf Seite 273](#)
- [SyncResult.getReceivedTruncateDeletes-Methode \[UltraLiteJ\] auf Seite 274](#)
- [SyncResult.getReceivedUpdates-Methode \[UltraLiteJ\] auf Seite 275](#)

getReceivedInserts-Methode

Gibt die Anzahl der empfangenen Zeilen zurück, die eingefügt wurden.

Syntax

```
abstract long SyncResult.getReceivedInserts()
```

Rückgabe

Die Anzahl der Zeilen, die als Einfügungen angewendet werden.

Siehe auch

- [SyncResult.getReceivedDeletes-Methode \[UltraLiteJ\] auf Seite 272](#)
- [SyncResult.getReceivedIgnoredDeletes-Methode \[UltraLiteJ\] auf Seite 272](#)
- [SyncResult.getReceivedInserts-Methode \[UltraLiteJ\] auf Seite 273](#)
- [SyncResult.getReceivedTruncateDeletes-Methode \[UltraLiteJ\] auf Seite 274](#)
- [SyncResult.getReceivedUpdates-Methode \[UltraLiteJ\] auf Seite 275](#)

getReceivedRowCount-Methode

Gibt die Anzahl der empfangenen Zeilen zurück.

Syntax

```
abstract long SyncResult.getReceivedRowCount()
```

Rückgabe

Die Anzahl der empfangenen Zeilen.

Bemerkungen

Diese Anzahl umfasst Zeilen, die eventuell ignoriert werden, wenn der Download übernommen wird.

Siehe auch

- [SyncResult.getTotalDownloadRowCount-Methode \[UltraLiteJ\] auf Seite 277](#)

getReceivedTruncateDeletes-Methode

Gibt die Anzahl der Zeilen zurück, die durch einen empfangenen Kürzungsvorgang gelöscht wurden.

Syntax

```
abstract long SyncResult.getReceivedTruncateDeletes()
```

Rückgabe

Die Anzahl der gekürzten Zeilen.

Bemerkungen

Jeder heruntergeladene Kürzungsvorgang erscheint als einzelne Zeile im von der getReceivedRowCount-Methode gezählten Zeilen-Download, aber möglicherweise werden null bis viele Zeilen gekürzt, wie durch die Methode gezählt.

Siehe auch

- [SyncResult.getReceivedDeletes-Methode \[UltraLiteJ\] auf Seite 272](#)
- [SyncResult.getReceivedIgnoredDeletes-Methode \[UltraLiteJ\] auf Seite 272](#)
- [SyncResult.getReceivedIgnoredUpdates-Methode \[UltraLiteJ\] auf Seite 273](#)
- [SyncResult.getReceivedInserts-Methode \[UltraLiteJ\] auf Seite 273](#)
- [SyncResult.getReceivedUpdates-Methode \[UltraLiteJ\] auf Seite 275](#)

getReceivedUpdates-Methode

Gibt die Anzahl der empfangenen Zeilen zurück, die als Aktualisierungen übernommen wurden.

Syntax

```
abstract long SyncResult.getReceivedUpdates()
```

Rückgabe

Die Anzahl der Zeilen, die als Aktualisierungen angewendet wurden.

Siehe auch

- [SyncResult.getReceivedDeletes-Methode \[UltraLiteJ\] auf Seite 272](#)
- [SyncResult.getReceivedIgnoredDeletes-Methode \[UltraLiteJ\] auf Seite 272](#)
- [SyncResult.getReceivedIgnoredUpdates-Methode \[UltraLiteJ\] auf Seite 273](#)
- [SyncResult.getReceivedInserts-Methode \[UltraLiteJ\] auf Seite 273](#)
- [SyncResult.getReceivedTruncateDeletes-Methode \[UltraLiteJ\] auf Seite 274](#)

getSentByteCount-Methode

Gibt die Anzahl der Bytes zurück, die während einer Datensynchronisation gesendet wurden.

Syntax

```
abstract long SyncResult.getSentByteCount()
```

Rückgabe

Die Anzahl der gesendeten Bytes.

getSentDeletes-Methode

Gibt die Anzahl der gelöschten Zeilen zurück, die gesendet wurden.

Syntax

```
abstract long SyncResult.getSentDeletes()
```

Rückgabe

Die Anzahl der gesendeten gelöschten Zeilen.

Bemerkungen

Die Anzahl von Einfügungen, Aktualisierungen und Löschungen können unterschiedlich sein zur Anzahl der bei den synchronisierten Tabellen durchgeführten Vorgänge, da alle Vorgänge in einer gegebenen Zeile zu einem Vorgang zusammengefasst werden.

Siehe auch

- [SyncResult.getSentInserts-Methode \[UltraLiteJ\] auf Seite 276](#)
- [SyncResult.getSentUpdates-Methode \[UltraLiteJ\] auf Seite 276](#)

getSentInserts-Methode

Gibt die Anzahl der eingefügten Zeilen zurück, die gesendet wurden.

Syntax

```
abstract long SyncResult.getSentInserts()
```

Rückgabe

Die Anzahl der eingefügten Zeilen, die gesendet wurden.

Bemerkungen

Die Anzahl von Einfügungen, Aktualisierungen und Löschungen können unterschiedlich sein zur Anzahl der bei den synchronisierten Tabellen durchgeführten Vorgänge, da alle Vorgänge in einer gegebenen Zeile zu einem Vorgang zusammengefasst werden.

Siehe auch

- [SyncResult.getSentDeletes-Methode \[UltraLiteJ\] auf Seite 275](#)
- [SyncResult.getSentUpdates-Methode \[UltraLiteJ\] auf Seite 276](#)

getSentUpdates-Methode

Gibt die Anzahl der aktualisierten Zeilen zurück, die gesendet wurden.

Syntax

```
abstract long SyncResult.getSentUpdates()
```

Rückgabe

Die Anzahl der aktualisierten Zeilen, die gesendet wurden.

Bemerkungen

Die Anzahl von Einfügungen, Aktualisierungen und Löschungen können unterschiedlich sein zur Anzahl der bei den synchronisierten Tabellen durchgeführten Vorgänge, da alle Vorgänge in einer gegebenen Zeile zu einem Vorgang zusammengefasst werden.

Siehe auch

- [SyncResult.getSentDeletes-Methode \[UltraLiteJ\] auf Seite 275](#)
- [SyncResult.getSentInserts-Methode \[UltraLiteJ\] auf Seite 276](#)

getStreamErrorCode-Methode

Gibt den Fehler zurück, der vom Datenstrom selbst gemeldet wurde.

Syntax

```
abstract int SyncResult.getStreamErrorCode()
```

Rückgabe

0, wenn es keinen Kommunikationsdatenstrom-Fehler gab, ansonsten der Antwortcode vom Server.

Bemerkungen

Diese Methode gibt den HTTP-Anwortcode zurück.

getStreamErrorMessage-Methode

Gibt den Fehler zurück, der vom Datenstrom selbst gemeldet wurde.

Syntax

```
abstract String SyncResult.getStreamErrorMessage()
```

Rückgabe

NULL, wenn keine Meldung verfügbar ist, andernfalls die Antwortmeldung.

Bemerkungen

Diese Methode gibt die HTTP-Anwortmeldung zurück.

getSyncedTableCount-Methode

Gibt die Anzahl der bisher synchronisierten Tabellen zurück.

Syntax

```
abstract int SyncResult.getSyncedTableCount()
```

Rückgabe

Die Anzahl der synchronisierten Tabellen.

getTotalDownloadRowCount-Methode

Gibt die Gesamtzahl der Zeilen zurück, die im Download empfangen wurden.

Syntax

```
abstract long SyncResult.getTotalDownloadRowCount()
```

Rückgabe

Die Anzahl der im Download zu empfangenden Zeilen. Diese Anzahl enthält alle Zeilen, die nicht zutreffen, wie z.B. Löschvorgänge für Zeilen, die nicht auf dem Client vorhanden sind.

Bemerkungen

Diese Anzahl enthält mehrfach vorkommende Zeilen, die ignoriert werden. Dieser Wert wird nicht eingestellt, bevor der SyncObserver.State.RECEIVING_TABLE-Status für die erste Tabelle erreicht ist.

getTotalTableCount-Methode

Gibt die Anzahl der zu synchronisierenden Tabellen zurück.

Syntax

```
abstract int SyncResult.getTotalTableCount()
```

Rückgabe

Die Anzahl der zu synchronisierenden Tabellen.

isUploadOK-Methode

Ermittelt, ob die letzte Upload-Synchronisation erfolgreich war.

Syntax

```
abstract boolean SyncResult.isUploadOK()
```

Rückgabe

TRUE, wenn die letzte Upload-Synchronisation erfolgreich war, ansonsten FALSE

SyncResult.AuthStatusCode-Schnittstelle

Listet die Autorisierungscodes auf, die vom MobiLink-Server zurückgegeben werden.

Syntax

```
public interface SyncResult.AuthStatusCode
```

Mitglieder

Alle Mitglieder der SyncResult.AuthStatusCode-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
EXPIRED-Variable	Benutzer-ID oder Kennwort ist abgelaufen.
IN_USE-Variable	Benutzer-ID wird bereits verwendet.

Name	Beschreibung
INVALID-Variable	Ungültige Benutzer-ID bzw. ungültiges Kennwort.
UNKNOWN-Variable	Autorisierungsstatus ist unbekannt.
VALID-Variable	Benutzer-ID und Kennwort waren zur Zeit der Synchronisation gültig.
VALID_BUT_EXPIRES_SOON-Variable	Benutzer-ID und Kennwort waren zur Zeit der Synchronisation gültig, laufen aber bald ab.

Siehe auch

- [SyncResult.getAuthStatus-Methode \[UltraLiteJ\] auf Seite 271](#)

EXPIRED-Variable

Benutzer-ID oder Kennwort ist abgelaufen.

Syntax

```
final int SyncResult.AuthStatusCode.EXPIRED
```

Bemerkungen

Die Autorisierung schlägt fehl.

IN_USE-Variable

Benutzer-ID wird bereits verwendet.

Syntax

```
final int SyncResult.AuthStatusCode.IN_USE
```

Bemerkungen

Die Autorisierung schlägt fehl.

INVALID-Variable

Ungültige Benutzer-ID bzw. ungültiges Kennwort.

Syntax

```
final int SyncResult.AuthStatusCode.INVALID
```

Bemerkungen

Die Autorisierung schlägt fehl.

UNKNOWN-Variable

Autorisierungsstatus ist unbekannt.

Syntax

```
final int SyncResult.AuthStatusCode.UNKNOWN
```

Bemerkungen

Dieser Code gibt an, dass die Synchronisation nicht durchgeführt wurde.

VALID-Variable

Benutzer-ID und Kennwort waren zur Zeit der Synchronisation gültig.

Syntax

```
final int SyncResult.AuthStatusCode.VALID
```

VALID_BUT_EXPIRES_SOON-Variable

Benutzer-ID und Kennwort waren zur Zeit der Synchronisation gültig, laufen aber bald ab.

Syntax

```
final int SyncResult.AuthStatusCode.VALID_BUT_EXPIRES_SOON
```

TableSchema-Schnittstelle

Gibt das Schema einer Tabelle an und stellt Konstante bereit, die die Namen von Systemtabellen definieren.

Syntax

```
public interface TableSchema
```

Mitglieder

Alle Mitglieder der TableSchema-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
SYS_ARTICLES-Variable	Enthält den Namen der Systemtabelle, die Informationen zu Publikationsartikeln enthält.

Name	Beschreibung
SYS_COLUMNS-Variable	Enthält den Namen der Systemtabelle, die Informationen zu den Tabellenspalten in der Datenbank enthält.
SYS_FKEY_COLUMNS-Variable	Enthält den Namen der Systemtabelle, die Informationen zu Fremdschlüsselspalten enthält.
SYS_FOREIGN_KEYS-Variable	Enthält den Namen der Systemtabelle, die Informationen zu Fremdschlüsseln in der Datenbank enthält.
SYS_INDEX_COLUMNS-Variable	Enthält den Namen der Systemtabelle, die Informationen zu den Indexspalten in der Datenbank enthält.
SYS_INDEXES-Variable	Enthält den Namen der Systemtabelle, die Informationen zu den Tabellenindizes in der Datenbank enthält.
SYS_INTERNAL-Variable	Enthält den Namen der Systemtabelle, die interne Informationen enthält.
SYS_PRIMARY_INDEX-Variable	Enthält den Namen des Primärschlüsselindexes von Systemtabellen.
SYS_PUBLICATIONS-Variable	Enthält den Namen der Systemtabelle, die Informationen zu Datenbankpublikationen enthält.
SYS_TABLES-Variable	Enthält den Namen der Systemtabelle, die Informationen zu den Tabellen in der Datenbank enthält.
SYS_ULDATA-Variable	Enthält den Namen der Systemtabelle, die Informationen zu Systemwerten enthält.
SYS_ULDATA_INTERNAL-Variable	Enthält den Typ für interne Systemdaten.
SYS_ULDATA_OPTION-Variable	Enthält den Typ für optionenbezogene Systemdaten.
SYS_ULDATA_PROPERTY-Variable	Enthält den Typ für eigenschaftsbezogene Systemdaten.
TABLE_IS_DOWNLOAD_ONLY-Variable	Gibt an, dass eine Tabelle eine reine Download-Tabelle ist (d.h. eine Tabelle, die nach dem Synchronisieren hochgeladen wird).
TABLE_IS_NOSYNC-Variable	Gibt an, dass eine Tabelle nicht synchronisiert wird.
TABLE_IS_SYSTEM-Variable	Gibt an, dass eine Tabelle eine Systemtabelle ist.

Bemerkungen

Diese Schnittstelle enthält nur tabellenbezogene Konstanten. Dazu gehören Systemtabellennamen, Tabellenparameter und Typen von Daten in der **sysuldata**-Systemtabelle.

SYS_ARTICLES-Variable

Enthält den Namen der Systemtabelle, die Informationen zu Publikationsartikeln enthält.

Syntax

```
final String TableSchema.SYS_ARTICLES
```

SYS_COLUMNS-Variable

Enthält den Namen der Systemtabelle, die Informationen zu den Tabellenspalten in der Datenbank enthält.

Syntax

```
final String TableSchema.SYS_COLUMNS
```

SYS_FKEY_COLUMNS-Variable

Enthält den Namen der Systemtabelle, die Informationen zu Fremdschlüsselspalten enthält.

Syntax

```
final String TableSchema.SYS_FKEY_COLUMNS
```

SYS_FOREIGN_KEYS-Variable

Enthält den Namen der Systemtabelle, die Informationen zu Fremdschlüsseln in der Datenbank enthält.

Syntax

```
final String TableSchema.SYS_FOREIGN_KEYS
```

SYS_INDEX_COLUMNS-Variable

Enthält den Namen der Systemtabelle, die Informationen zu den Indexspalten in der Datenbank enthält.

Syntax

```
final String TableSchema.SYS_INDEX_COLUMNS
```

SYS_INDEXES-Variable

Enthält den Namen der Systemtabelle, die Informationen zu den Tabellenindizes in der Datenbank enthält.

Syntax

```
final String TableSchema.SYS_INDEXES
```

SYS_INTERNAL-Variable

Enthält den Namen der Systemtabelle, die interne Informationen enthält.

Syntax

```
final String TableSchema.SYS_INTERNAL
```

SYS_PRIMARY_INDEX-Variable

Enthält den Namen des Primärschlüsselindexes von Systemtabellen.

Syntax

```
final String TableSchema.SYS_PRIMARY_INDEX
```

SYS_PUBLICATIONS-Variable

Enthält den Namen der Systemtabelle, die Informationen zu Datenbankpublikationen enthält.

Syntax

```
final String TableSchema.SYS_PUBLICATIONS
```

SYS_TABLES-Variable

Enthält den Namen der Systemtabelle, die Informationen zu den Tabellen in der Datenbank enthält.

Syntax

```
final String TableSchema.SYS_TABLES
```

SYS_ULDATA-Variable

Enthält den Namen der Systemtabelle, die Informationen zu Systemwerten enthält.

Syntax

```
final String TableSchema.SYS_ULDATA
```

SYS_ULDATA_INTERNAL-Variable

Enthält den Typ für interne Systemdaten.

Syntax

```
final String TableSchema.SYS_ULDATA_INTERNAL
```

SYS_ULDATA_OPTION-Variable

Enthält den Typ für optionenbezogene Systemdaten.

Syntax

```
final String TableSchema.SYS_ULDATA_OPTION
```

SYS_ULDATA_PROPERTY-Variable

Enthält den Typ für eigenschaftenbezogene Systemdaten.

Syntax

```
final String TableSchema.SYS_ULDATA_PROPERTY
```

TABLE_IS_DOWNLOAD_ONLY-Variable

Gibt an, dass eine Tabelle eine reine Download-Tabelle ist (d.h. eine Tabelle, die nach dem Synchronisieren hochgeladen wird).

Syntax

```
final short TableSchema.TABLE_IS_DOWNLOAD_ONLY
```

Bemerkungen

Dieser Wert kann logisch mit anderen Parametern in der table_flags-Spalte der SYS_TABLES-Tabelle kombiniert werden, mit Ausnahme des TABLE_IS_NOSYNC-Parameters.

TABLE_IS_NOSYNC-Variable

Gibt an, dass eine Tabelle nicht synchronisiert wird.

Syntax

```
final short TableSchema.TABLE_IS_NOSYNC
```

Bemerkungen

Dieser Wert kann logisch mit anderen Parametern in der table_flags-Spalte der SYS_TABLES-Tabelle kombiniert werden, mit Ausnahme des TABLE_IS_DOWNLOAD_ONLY-Parameters.

TABLE_IS_SYSTEM-Variable

Gibt an, dass eine Tabelle eine Systemtabelle ist.

Syntax

```
final short TableSchema.TABLE_IS_SYSTEM
```

Bemerkungen

Dieser Wert kann logisch mit anderen Parametern in der table_flags-Spalte der Tabelle SYS_TABLES kombiniert werden.

ULjEvent-Schnittstelle [Android]

Stellt ein UltraLiteJ-API-Systemereignis dar.

Syntax

```
public interface ULjEvent
```

Mitglieder

Alle Mitglieder der ULjEvent-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getParameter-Methode	Gibt einen benannten Parameter für das Ereignis zurück.
getType-Methode	Gibt den Ereignistyp zurück.
COMMIT_EVENT-Variable	Gibt den Ereignistyp "Commit" zurück.
SYNC_COMPLETE_EVENT-Variable	Gibt den Ereignistyp "Synchronisation abgeschlossen" zurück.
TABLE_MODIFIED_EVENT-Variable	Gibt den Ereignistyp "Tabelle geändert" zurück.

getParameter-Methode

Gibt einen benannten Parameter für das Ereignis zurück.

Syntax

```
String ULjEvent.getParameter(String name) throws ULjException
```

Parameter

- **name** Der Ereignisname, für den der Wert abgerufen werden soll.

getType-Methode

Gibt den Ereignistyp zurück.

Syntax

```
short ULjEvent.getType()
```

COMMIT_EVENT-Variable

Gibt den Ereignistyp "Commit" zurück.

Syntax

```
final short ULjEvent.COMMIT_EVENT
```

SYNC_COMPLETE_EVENT-Variable

Gibt den Ereignistyp "Synchronisation abgeschlossen" zurück.

Syntax

```
final short ULjEvent.SYNC_COMPLETE_EVENT
```

TABLE_MODIFIED_EVENT-Variable

Gibt den Ereignistyp "Tabelle geändert" zurück.

Syntax

```
final short ULjEvent.TABLE_MODIFIED_EVENT
```

ULjException-Klasse

Ersetzt die von der Datenbank ausgegebenen Ausnahmebedingungen.

Syntax

```
public class ULjException
```

Mitglieder

Alle Mitglieder der ULjException-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getCausingException-Methode	Gibt das ULjException-Objekt zurück, wenn eine Ausnahmebedingung verursacht wird.

Name	Beschreibung
getErrorCode-Methode	Gibt den Fehlercode zurück, der der Ausnahmebedingung zugeordnet ist.
getParameter-Methode [Android]	Gibt den angegebenen Fehlerparameter zurück.
getParameterCount-Methode [Android]	Gibt die Anzahl der Fehlerparameter zurück.
getSqlOffset-Methode	Gibt den Fehler-Ausgangspunkt in der SQL-Zeichenfolge zurück.

getCausingException-Methode

Gibt das ULjException-Objekt zurück, wenn eine Ausnahmebedingung verursacht wird.

Syntax

```
abstract ULjException ULjException.getCausingException()
```

Rückgabe

NULL, wenn keine verursachende Ausnahmebedingung vorhanden ist, ansonsten das ULjException-Objekt.

getErrorCode-Methode

Gibt den Fehlercode zurück, der der Ausnahmebedingung zugeordnet ist.

Syntax

```
abstract int ULjException.getErrorCode()
```

Rückgabe

Der Fehlercode.

getParameter-Methode [Android]

Gibt den angegebenen Fehlerparameter zurück.

Syntax

```
abstract String ULjException.getParameter(short param_no)
```

Parameter

- **param_no** Eine Parameternummer auf Basis eins.

Rückgabe

Der Fehlerparameter.

getParameterCount-Methode [Android]

Gibt die Anzahl der Fehlerparameter zurück.

Syntax

```
abstract short ULjException.getParameterCount()
```

Rückgabe

Die Anzahl der Fehlerparameter.

getSqlOffset-Methode

Gibt den Fehler-Ausgangspunkt in der SQL-Zeichenfolge zurück.

Syntax

```
abstract int ULjException.getSqlOffset()
```

Rückgabe

-1, wenn es keine der Fehlermeldung zugeordnete SQL-Zeichenfolge gibt, andernfalls den auf Null basierenden Offset innerhalb dieser Zeichenfolge, an dem der Fehler aufgetreten ist.

Unsigned64-Klasse

Implementiert 64-Bit-Binärwerte ohne Vorzeichen.

Syntax

```
public class Unsigned64
```

Mitglieder

Alle Mitglieder der Unsigned64-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
add-Methode	Addiert zwei Werte und setzt sich selbst auf das Ergebnis.
compare-Methode	Vergleicht zwei long-Werte.
divide-Methode	Dividiert zwei Werte und setzt sich selbst auf das Ergebnis.
multiply-Methode	Multipliziert zwei Werte und setzt sich selbst auf das Ergebnis.

Name	Beschreibung
remainder-Methode	Gibt den Rest zurück, wenn ein Wert durch einen anderen dividiert wird.
subtract-Methode	Subtrahiert zwei Werte und setzt sich selbst auf das Ergebnis.

Bemerkungen

Die mit dieser Klasse verfolgte Absicht besteht darin, Werte als Ganzzahlen (long integer) beizubehalten, aber mit den statische Methoden in dieser Klasse zu interpretieren.

Diese Klasse kann nicht instanziiert werden.

add-Methode

Addiert zwei Werte und setzt sich selbst auf das Ergebnis.

Syntax

```
final long Unsigned64.add(long v1, long v2)
```

Parameter

- **v1** Der erste Operand.
- **v2** Der zweite Operand.

Rückgabe

Die Summe der Operanden.

compare-Methode

Vergleicht zwei long-Werte.

Überladungsliste

Name	Beschreibung
compare(int, int)-Methode	Vergleicht zwei Ganzzahlwerte.
compare(long, long)-Methode	Vergleicht zwei long-Werte.

compare(int, int)-Methode

Vergleicht zwei Ganzzahlwerte.

Syntax

```
final byte Unsigned64.compare(int v1, int v2)
```

Parameter

- **v1** Der erste Wert, der verglichen werden soll.
- **v2** Der zweite Wert, der verglichen werden soll.

Rückgabe

-1, wenn v2 größer ist als v1, 0, wenn v1 gleich v2 ist, und 1, wenn v2 kleiner ist als v1.

compare(long, long)-Methode

Vergleicht zwei long-Werte.

Syntax

```
final byte Unsigned64.compare(long v1, long v2)
```

Parameter

- **v1** Der erste Wert, der verglichen werden soll.
- **v2** Der zweite Wert, der verglichen werden soll.

Rückgabe

-1, wenn v2 größer ist als v1, 0, wenn v1 gleich v2 ist, und 1, wenn v2 kleiner ist als v1.

divide-Methode

Dividiert zwei Werte und setzt sich selbst auf das Ergebnis.

Syntax

```
final long Unsigned64.divide(long v1, long v2)
```

Parameter

- **v1** Der erste Operand.
- **v2** Der zweite Operand.

Rückgabe

Der erste Operand dividiert durch den zweiten Operanden.

multiply-Methode

Multipliziert zwei Werte und setzt sich selbst auf das Ergebnis.

Syntax

```
final long Unsigned64.multiply(long v1, long v2)
```

Parameter

- **v1** Der erste Operand.
- **v2** Der zweite Operand.

Rückgabe

Das Produkt aus v1 und v2.

remainder-Methode

Gibt den Rest zurück, wenn ein Wert durch einen anderen dividiert wird.

Überladungsliste

Name	Beschreibung
remainder(long, long)-Methode	Gibt den Rest zurück, wenn ein Wert durch einen anderen dividiert wird.
remainder(long, long, long)-Methode	Gibt den Rest zurück, wenn ein Wert, multipliziert mit einem bestimmten Quotienten, von einem anderen Wert abgezogen wird ($v1 - \text{quot} * v2$).

remainder(long, long)-Methode

Gibt den Rest zurück, wenn ein Wert durch einen anderen dividiert wird.

Syntax

```
final long Unsigned64.remainder(long v1, long v2)
```

Parameter

- **v1** Der Wert, der dividiert werden soll.
- **v2** Der Wert, durch den dividiert werden soll.

Rückgabe

Der Rest, dargestellt als Ganzzahlwert (long integer).

remainder(long, long, long)-Methode

Gibt den Rest zurück, wenn ein Wert, multipliziert mit einem bestimmten Quotienten, von einem anderen Wert abgezogen wird ($v1 - \text{quot} * v2$).

Syntax

```
final long Unsigned64.remainder(long v1, long v2, long quot)
```

Parameter

- **v1** Der Wert, der dividiert werden soll.
- **v2** Der Wert, durch den dividiert werden soll.
- **quot** Der Wert des Quotienten.

Rückgabe

Der Rest, dargestellt als Ganzzahlwert (long integer).

subtract-Methode

Subtrahiert zwei Werte und setzt sich selbst auf das Ergebnis.

Syntax

```
final long Unsigned64.subtract(long v1, long v2)
```

Parameter

- **v1** Der erste Operand.
- **v2** Der zweite Operand.

Rückgabe

Das Ergebnis, wenn v2 von v1 subtrahiert wird.

UUIDValue-Schnittstelle

Beschreibt einen eindeutiges Bezeichnerobjekt (UUID oder universell eindeutiger Bezeichner).

Syntax

```
public interface UUIDValue
```

Mitglieder

Alle Mitglieder der UUIDValue-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getString-Methode	Gibt die Zeichenfolgendarstellung des UUIDValue-Objekts zurück.
isNull-Methode	Ermittelt, ob das UUIDValue-Objekt NULL ist.
set-Methode	Setzt das UUIDValue-Objekt auf einen Zeichenfolgenwert.
setNull-Methode	Setzt das UUIDValue-Objekt auf NULL.

Bemerkungen

Solche Objekte sind nützlich, wenn ein eindeutiger Bezeichner erforderlich ist und der Wert arbiträr sein kann.

Eine UUIDValue-Objekt kann auch von der SQL-Anweisung INSERT erstellt werden, wenn für eine Spalte in einer Tabelle kein Wert angegeben wird und die Spalte mit der Klausel DEFAULT NEWID() erstellt wurde.

Ein Connection-Objekt kann verwendet werden, um ein UUIDValue-Objekt mit der createUUIDValue-Methode zu erstellen.

Siehe auch

- [Connection.createUUIDValue-Methode \[UltraLiteJ\] auf Seite 113](#)

getString-Methode

Gibt die Zeichenfolgendarstellung des UUIDValue-Objekts zurück.

Syntax

```
String UUIDValue.getString() throws ULjException
```

Rückgabe

Der Zeichenfolgenwert.

isNull-Methode

Ermittelt, ob das UUIDValue-Objekt NULL ist.

Syntax

```
boolean UUIDValue.isNull()
```

Rückgabe

TRUE, wenn das Objekt Null ist, ansonsten FALSE.

set-Methode

Setzt das UUIDValue-Objekt auf einen Zeichenfolgenwert.

Syntax

```
void UUIDValue.set(String value) throws ULjException
```

Parameter

- **value** Ein numerischer Wert, als Zeichenfolge dargestellt.

setNull-Methode

Setzt das UUIDValue-Objekt auf NULL.

Syntax

```
void UUIDValue.setNull() throws ULjException
```

ValidateDatabaseProgressData-Schnittstelle [Android]

Berichtet ValidateDatabase-Fortschrittsdaten.

Syntax

```
public interface ValidateDatabaseProgressData
```

Mitglieder

Alle Mitglieder der ValidateDatabaseProgressData-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
getParms-Methode	Gibt ein der Status-ID zugehöriges Parameter-Array zurück.
getStatusId-Methode	Gibt die Status-ID des Validierungsvorgangs zurück.

Siehe auch

- „UltraLite-Dienstprogramm zum Validieren von Datenbanken (ulvalid)“ [*UltraLite - Datenbankverwaltung*]

getParms-Methode

Gibt ein der Status-ID zugehöriges Parameter-Array zurück.

Syntax

```
abstract String[] ValidateDatabaseProgressData.getParms()
```

Rückgabe

Das Array von Parametern als feste Größe. Nicht verwendete Parameter sind NULL.

Siehe auch

- [ValidateDatabaseProgressData.StatusId-Schnittstelle \[Android\] \[UltraLiteJ\] auf Seite 295](#)

getStatusId-Methode

Gibt die Status-ID des Validierungsvorgangs zurück.

Syntax

```
abstract short ValidateDatabaseProgressData.getStatusId()
```

Rückgabe

Eine Status-ID des Typs Konstante.

ValidateDatabaseProgressData.StatusId-Schnittstelle [Android]

Legt mögliche Status-IDs für das UltraLite-Datenbank-Validierungstool fest.

Syntax

```
public interface ValidateDatabaseProgressData.StatusId
```

Mitglieder

Alle Mitglieder der ValidateDatabaseProgressData.StatusID-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
UL_VALID_BAD_ROWID-Variable	Im Index ist ein ungültiger Zeilenbezeichner vorhanden.
UL_VALID_CHECKING_INDEX-Variable	Prüfung eines Indexes.
UL_VALID_CHECKING_PAGE-Variable	Sendet eine periodische Statusmeldung während der Prüfung von Datenbankseiten.
UL_VALID_CHECKING_TABLE-Variable	Prüfung einer Tabelle.
UL_VALID_CONNECT_ERROR-Variable	Fehler beim Verbinden mit der Datenbank.
UL_VALID_CORRUPT_PAGE-Variable	Eine Seite ist beschädigt.
UL_VALID_CORRUPT_PAGE_TABLE-Variable	Seitentabelle ist beschädigt.
UL_VALID_DATABASE_ERROR-Variable	Beim Zugriff auf die Datenbank ist ein Fehler aufgetreten.
UL_VALID_END-Variable	Ende der Validierung.

Name	Beschreibung
UL_VALID_FAILED_CHECKSUM-Variable	Seitenprüfsumme ist fehlgeschlagen.
UL_VALID_INTERRUPTED-Variable	Validierungsvorgang unterbrochen.
UL_VALID_NO_ERROR-Variable	Es ist kein Fehler aufgetreten.
UL_VALID_ROWCOUNT_MISMATCH-Variable	Die Anzahl der Zeilen im Index unterscheidet sich von der Tabellenzeilenanzahl.
UL_VALID_START-Variable	Starten der Validierung.
UL_VALID_STARTUP_ERROR-Variable	Fehler beim Starten der Datenbank bei systemnaheem Zugriff.

UL_VALID_BAD_ROWID-Variable

Im Index ist ein ungültiger Zeilenbezeichner vorhanden.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_BAD_ROWID
```

Bemerkungen

Der erste Parameter, der von der `ValidateDatabaseProgressData.getParms`-Methode zurückgegeben wird, protokolliert den Tabellennamen. Der zweite Parameter protokolliert den Indexnamen.

Siehe auch

- [ValidateDatabaseProgressData.getParms-Methode \[UltraLiteJ\] auf Seite 294](#)

UL_VALID_CHECKING_INDEX-Variable

Prüfung eines Indexes.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CHECKING_INDEX
```

Bemerkungen

Der erste Parameter, der von der `ValidateDatabaseProgressData.getParms`-Methode zurückgegeben wird, speichert den Tabellennamen. Der zweite Parameter speichert den Indexnamen.

Siehe auch

- [ValidateDatabaseProgressData.getParms-Methode \[UltraLiteJ\] auf Seite 294](#)

UL_VALID_CHECKING_PAGE-Variable

Sendet eine periodische Statusmeldung während der Prüfung von Datenbankseiten.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CHECKING_PAGE
```

Bemerkungen

Der erste Parameter, der von der ValidateDatabaseProgressData.getParms-Methode zurückgegeben wird, protokolliert eine mit der Seite verknüpfte Zahl. Die Reihenfolge ist nicht definiert.

Siehe auch

- [ValidateDatabaseProgressData.getParms-Methode \[UltraLiteJ\] auf Seite 294](#)

UL_VALID_CHECKING_TABLE-Variable

Prüfung einer Tabelle.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CHECKING_TABLE
```

Bemerkungen

Der erste Parameter, der von der ValidateDatabaseProgressData.getParms-Methode zurückgegeben wird, protokolliert den Tabellennamen.

Siehe auch

- [ValidateDatabaseProgressData.getParms-Methode \[UltraLiteJ\] auf Seite 294](#)

UL_VALID_CONNECT_ERROR-Variable

Fehler beim Verbinden mit der Datenbank.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CONNECT_ERROR
```

UL_VALID_CORRUPT_PAGE-Variable

Eine Seite ist beschädigt.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CORRUPT_PAGE
```

Bemerkungen

Der erste Parameter, der von der `ValidateDatabaseProgressData.getParms`-Methode zurückgegeben wird, protokolliert eine mit der Seite verknüpfte Zahl.

Siehe auch

- [ValidateDatabaseProgressData.getParms-Methode \[UltraLiteJ\] auf Seite 294](#)

UL_VALID_CORRUPT_PAGE_TABLE-Variable

Seitentabelle ist beschädigt.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CORRUPT_PAGE_TABLE
```

UL_VALID_DATABASE_ERROR-Variable

Beim Zugriff auf die Datenbank ist ein Fehler aufgetreten.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_DATABASE_ERROR
```

Bemerkungen

Weitere Informationen finden Sie im SQLCODE.

Siehe auch

- „SQL Anywhere-Fehlermeldungen - sortiert nach SQLCODE“ [[Fehlermeldungen](#)]

UL_VALID_END-Variable

Ende der Validierung.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_END
```

Bemerkungen

Der erste Parameter, der von der `ValidateDatabaseProgressData.getParms`-Methode zurückgegeben wird, protokolliert den sich ergebenden SQLCODE, der Erfolg oder Fehlschlag anzeigt.

Siehe auch

- [ValidateDatabaseProgressData.getParms-Methode \[UltraLiteJ\] auf Seite 294](#)
- „SQL Anywhere-Fehlermeldungen - sortiert nach SQLCODE“ [[Fehlermeldungen](#)]

UL_VALID_FAILED_CHECKSUM-Variable

Seitenprüfsumme ist fehlgeschlagen.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_FAILED_CHECKSUM
```

Bemerkungen

Der erste Parameter, der von der ValidateDatabaseProgressData.getParms-Methode zurückgegeben wird, protokolliert eine mit der Seite verknüpfte Zahl.

Siehe auch

- [ValidateDatabaseProgressData.getParms-Methode \[UltraLiteJ\] auf Seite 294](#)

UL_VALID_INTERRUPTED-Variable

Validierungsvorgang unterbrochen.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_INTERRUPTED
```

UL_VALID_NO_ERROR-Variable

Es ist kein Fehler aufgetreten.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_NO_ERROR
```

UL_VALID_ROWCOUNT_MISMATCH-Variable

Die Anzahl der Zeilen im Index unterscheidet sich von der Tabellenzeilenanzahl.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_ROWCOUNT_MISMATCH
```

Bemerkungen

Der erste Parameter, der von der ValidateDatabaseProgressData.getParms-Methode zurückgegeben wird, protokolliert den Tabellennamen. Der zweite Parameter protokolliert den Indexnamen.

Siehe auch

- [ValidateDatabaseProgressData.getParms-Methode \[UltraLiteJ\] auf Seite 294](#)

UL_VALID_START-Variable

Starten der Validierung.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_START
```

UL_VALID_STARTUP_ERROR-Variable

Fehler beim Starten der Datenbank bei systemnahe Zugriff.

Syntax

```
final short ValidateDatabaseProgressData.StatusId.UL_VALID_STARTUP_ERROR
```

ValidateDatabaseProgressListener-Schnittstelle [Android]

Empfängt ValidateDatabase-Fortschrittsereignisse.

Syntax

```
public interface ValidateDatabaseProgressListener
```

Mitglieder

Alle Mitglieder der ValidateDatabaseProgressListener-Schnittstelle, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
validateProgressed-Methode	Wird während eines ValidateDatabase-Vorgangs aufgerufen, um den Benutzer über den Fortschritt der Validierung zu informieren.

validateProgressed-Methode

Wird während eines ValidateDatabase-Vorgangs aufgerufen, um den Benutzer über den Fortschritt der Validierung zu informieren.

Syntax

```
boolean ValidateDatabaseProgressListener.validateProgressed(  
    ValidateDatabaseProgressData data  
)
```

Parameter

- **data** Ein ValidateDatabaseProgressData-Objekt, das die neuesten Fortschrittsdaten zur Validierung enthält.

Rückgabe

TRUE, um den Validierungsprozess abubrechen. Ansonsten wird FALSE zurückgegeben.

Index

A

- add-Methode
 - DecimalNumber-Schnittstelle [UltraLiteJ-API], 146
 - Unsigned64-Klasse [UltraLiteJ-API], 289
- addCustomHTTPHeader-Methode [BlackBerry]
 - StreamHTTPParams-Schnittstelle [UltraLiteJ-API], 228
- AfterLast-Methode
 - UltraLiteJ-Beispiel, 18
- afterLast-Methode [Android]
 - ResultSet-Schnittstelle [UltraLiteJ-API], 200
- Android
 - Codebeispiele, 30
 - CustDB-Synchronisation, 30
 - Datenbank erstellen, 5
 - Datenbankschemata, 19
 - Datenbankspeicher, 5
 - Deployment von Anwendungen durchführen, 27
 - Fehlerbehandlung, 20
 - Info zu UltraLiteJ-Anwendungen, 3
 - mit MobiLink synchronisieren, 21
 - Netzwerkprotokolloptionen, 23
 - praktische Einführung, 41
 - Setup-Hinweise, 4
 - Verbindung mit einer Datenbank herstellen, 5
 - Verbindung mit einer UltraLite-Datenbank trennen, 26
- Anwendungen
 - BlackBerry-Deployment, 49
 - für Android und BlackBerry entwickeln, 3, 27, 29
- Architekturen
 - UltraLiteJ, 81
- ASCENDING-Variable
 - IndexSchema-Schnittstelle [UltraLiteJ-API], 179
- Auswählen, Daten aus Datenbanktabellen
 - UltraLiteJ, 18
- AutoCommit, Modus
 - UltraLiteJ-Entwicklung, 15

B

- BeforeFirst-Methode
 - UltraLiteJ-Beispiel, 18
- beforeFirst-Methode [Android]
 - ResultSet-Schnittstelle [UltraLiteJ-API], 200

Beispielcode

- CreateDb, 32
- CreateSales, 34
- DumpSchema, 37
- Encrypted, 37
- LoadDb, 32
- ReadInnerJoin, 34
- ReadSeq, 33
- Reorg, 36
- SalesReport, 35
- SortTransactions, 35
- Synchronisation, 21, 40
- UltraLiteJ, 30

Beispiele

- CreateDb.java, 32
- CreateSales.java, 34
- Demo.java, 30
- DumpSchema, 37
- Encrypted.java, 37
- LoadDb.java, 32
- ReadInnerJoin.java, 34
- ReadSeq.java, 33
- Reorg.java, 36
- SalesReport.java, 35
- SortTransactions.java, 35
- Synchronisation, 40
- Synchronisation mit UltraLiteJ, 24
- UltraLiteJ, 30

BIG-Variable

- Domain-Schnittstelle [UltraLiteJ-API], 153

BINARY-Variable

- Domain-Schnittstelle [UltraLiteJ-API], 153

BIT-Variable

- Domain-Schnittstelle [UltraLiteJ-API], 153

BlackBerry

- Codebeispiele, 30
- CustDB-Synchronisation, 24
- Datenbank erstellen, 5
- Datenbankschemata, 19
- Datenbankspeicher, 5
- Datensynchronisation, 23
- Deployment von Anwendungen durchführen, 27, 29
- ein Eclipse-Projekt erstellen, 49
- Fehlerbehandlung, 20
- gleichzeitige Synchronisation, 23
- Info zu UltraLiteJ-Anwendungen, 3
- JDE-Komponentenpaket, 49
- mit MobiLink synchronisieren, 21

- Netzwerkprotokolloptionen,23
- praktische Einführung,49
- Setup-Hinweise,4
- Signature-Tool,49
- Synchronisationsfunktion hinzufügen,65
- UltraLiteJ-Anwendung erstellen,49
- Verbindung mit einer Datenbank herstellen,5
- Verbindung mit einer UltraLite Java Edition-Datenbank trennen,26

C

- cancelWaitForEvent-Methode [BlackBerry]
 - Connection-Schnittstelle [UltraLiteJ-API],109
- changeEncryptionKey-Methode
 - Connection-Schnittstelle [UltraLiteJ-API],110
- close-Methode
 - PreparedStatement-Schnittstelle [UltraLiteJ-API],182
 - ResultSet-Schnittstelle [UltraLiteJ-API],200
- COLUMN_DEFAULT_AUTOFILNAME-Variable
 - ColumnSchema-Schnittstelle [UltraLiteJ-API],82
- COLUMN_DEFAULT_AUTOINC-Variable
 - ColumnSchema-Schnittstelle [UltraLiteJ-API],83
- COLUMN_DEFAULT_CONSTANT-Variable
 - ColumnSchema-Schnittstelle [UltraLiteJ-API],83
- COLUMN_DEFAULT_CURRENT_DATE-Variable
 - ColumnSchema-Schnittstelle [UltraLiteJ-API],84
- COLUMN_DEFAULT_CURRENT_TIME-Variable
 - ColumnSchema-Schnittstelle [UltraLiteJ-API],84
- COLUMN_DEFAULT_CURRENT_TIMESTAMP-Variable
 - ColumnSchema-Schnittstelle [UltraLiteJ-API],84
- COLUMN_DEFAULT_CURRENT_UTC_TIMESTAMP-Variable
 - ColumnSchema-Schnittstelle [UltraLiteJ-API],85
- COLUMN_DEFAULT_GLOBAL_AUTOINC-Variable
 - ColumnSchema-Schnittstelle [UltraLiteJ-API],85
- COLUMN_DEFAULT_NONE-Variable
 - ColumnSchema-Schnittstelle [UltraLiteJ-API],86
- COLUMN_DEFAULT_UNIQUE_ID-Variable
 - ColumnSchema-Schnittstelle [UltraLiteJ-API],86
- ColumnSchema-Schnittstelle [UltraLiteJ-API]
 - Beschreibung,81
- COLUMN_DEFAULT_AUTOFILNAME-Variable,82
- COLUMN_DEFAULT_AUTOINC-Variable,83
- COLUMN_DEFAULT_CONSTANT-Variable,83
- COLUMN_DEFAULT_CURRENT_DATE-Variable,84
- COLUMN_DEFAULT_CURRENT_TIME-Variable,84
- COLUMN_DEFAULT_CURRENT_TIMESTAMP-Variable,85
- COLUMN_DEFAULT_GLOBAL_AUTOINC-Variable,85
- COLUMN_DEFAULT_NONE-Variable,86
- COLUMN_DEFAULT_UNIQUE_ID-Variable,86

getRowScoreFlushSize-Methode [BlackBerry],97
 getRowScoreMaximum-Methode [BlackBerry],97
 getUsername-Methode [Android],98
 hasShadowPaging-Methode [BlackBerry],98
 setCacheSize-Methode,98
 setConnectionString-Methode [Android],99
 setCreationString-Methode [Android],99
 setEncryptionKey-Methode,100
 setLazyLoadIndexes-Methode [BlackBerry],100
 setRowScoreFlushSize-Methode [BlackBerry],101
 setRowScoreMaximum-Methode [BlackBerry],101
 setUsername-Methode [Android],102
 Configuration-Objekte
 UltraLiteJ,5
 Configuration-Schnittstelle [UltraLiteJ-API]
 Beschreibung,102
 getDatabaseName-Methode,103
 getPageSize-Methode,103
 setDatabaseName-Methode,103
 setPageSize-Methode,104
 setPassword-Methode,104
 connect-Methode
 DatabaseManager-Klasse [UltraLiteJ-API],139
 CONNECTED-Variable
 Connection-Schnittstelle [UltraLiteJ-API],124
 CONNECTING-Variable
 SyncObserver.States-Schnittstelle [UltraLiteJ-API],247
 Connection-Objekte
 UltraLiteJ,5
 Connection-Schnittstelle [UltraLiteJ-API]
 Beschreibung,105
 cancelWaitForEvent-Methode [BlackBerry],109
 changeEncryptionKey-Methode,110
 commit-Methode,110
 CONNECTED-Variable,124
 createDecimalNumber-Methode,110
 createSyncParms-Methode,112
 createUUIDValue-Methode,113
 dropDatabase-Methode,113
 emergencyShutdown-Methode [BlackBerry],114
 getDatabaseId-Methode [BlackBerry],114
 getDatabaseInfo-Methode,114
 getDatabaseProperty-Methode,114
 getLastDownloadTime-Methode,115
 getLastIdentity-Methode,115
 getLastWarning-Methode,116
 getOption-Methode [BlackBerry],116
 getState-Methode,117
 getSyncObserver-Methode,117
 getSyncResult-Methode,117
 isSynchronizationDeleteDisabled-Methode [BlackBerry],118
 NOT_CONNECTED-Variable,125
 OPTION_BLOB_FILE_BASE_DIR-Variable [BlackBerry],125
 OPTION_DATABASE_ID-Variable [BlackBerry],125
 OPTION_DATE_FORMAT-Variable,125
 OPTION_DATE_ORDER-Variable,126
 OPTION_MAX_HASH_SIZE-Variable,126
 OPTION_ML_REMOTE_ID-Variable [BlackBerry],127
 OPTION_ML_SERVER_VERSION-Variable [BlackBerry],127
 OPTION_NEAREST_CENTURY-Variable,127
 OPTION_PRECISION-Variable,128
 OPTION_SCALE-Variable,128
 OPTION_TIME_FORMAT-Variable,129
 OPTION_TIMESTAMP_FORMAT-Variable,129
 OPTION_TIMESTAMP_INCREMENT-Variable,130
 OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT-Variable,130
 prepareStatement-Methode,118
 PROPERTY_DATABASE_NAME-Variable,130
 PROPERTY_PAGE_SIZE-Variable,131
 registerForEvent-Methode [BlackBerry],119
 release-Methode,119
 resetLastDownloadTime-Methode,119
 rollback-Methode,120
 rollbackPartialDownload-Methode [Android],120
 setDatabaseId-Methode,120
 setOption-Methode,121
 setSyncObserver-Methode,122
 SYNC_ALL-Variable,131
 SYNC_ALL_DB_PUB_NAME-Variable,131
 SYNC_ALL_PUBS-Variable,132
 synchronize-Methode,122
 ULVF_DATABASE-Variable [Android],132
 ULVF_EXPRESS-Variable [Android],132
 ULVF_FULL_VALIDATE-Variable [Android],133
 ULVF_INDEX-Variable [Android],133
 ULVF_TABLE-Variable [Android],133
 unregisterForEvent-Methode [Android],123

- validateDatabase-Methode [Android],123
- waitForEvent-Methode [Android],124
- createConfigurationFile, Methode
 - DatabaseManager, Klasse [UltraLiteJ-API],140
- createConfigurationFileAndroid-Methode
 - DatabaseManager-Klasse [UltraLiteJ-API],140
- createConfigurationNonPersistent-Methode [BlackBerry]
- DatabaseManager-Klasse [UltraLiteJ-API],141
- createConfigurationObjectStore-Methode [BlackBerry]
- DatabaseManager-Klasse [UltraLiteJ-API],141
- createDatabase-Methode
 - DatabaseManager-Klasse [UltraLiteJ-API],142
- createDecimalNumber-Methode
 - Connection-Schnittstelle [UltraLiteJ-API],110
- createFileTransfer-Methode
 - DatabaseManager-Klasse [UltraLiteJ-API],142
- createFileTransferAndroid-Methode [BlackBerry]
 - DatabaseManager-Klasse [UltraLiteJ-API],143
- createObjectStoreTransfer-Methode [BlackBerry]
 - DatabaseManager-Klasse [UltraLiteJ-API],144
- createSISHTTPListener-Methode [BlackBerry]
 - DatabaseManager-Klasse [UltraLiteJ-API],144
- createSyncParms-Methode
 - Connection-Schnittstelle [UltraLiteJ-API],112
- createUUIDValue-Methode
 - Connection-Schnittstelle [UltraLiteJ-API],113
- CustDB
 - Android-Beispiel,30
 - BlackBerry, Beispiel,24

D

- DatabaseInfo-Schnittstelle [UltraLiteJ-API]
 - Beschreibung,133
 - getCommitCount-Methode [BlackBerry],134
 - getDbFormat-Methode [BlackBerry],135
 - getDbSize-Methode [BlackBerry],135
 - getLogSize-Methode [BlackBerry],135
 - getNumberRowsToUpload-Methode,135
 - getPageReads-Methode,136
 - getPageSize-Methode,137
 - getPageWrites-Methode,137
 - getRelease-Methode,137
- DatabaseManager, Klasse [UltraLiteJ-API]
 - createConfigurationFile, Methode,140
- DatabaseManager-Klasse [UltraLiteJ-API]
 - Beschreibung,138
 - connect-Methode,139
 - createConfigurationFileAndroid-Methode,140
 - createConfigurationNonPersistent-Methode [BlackBerry],141
 - createConfigurationObjectStore-Methode [BlackBerry],141
 - createDatabase-Methode,142
 - createFileTransfer-Methode,142
 - createFileTransferAndroid-Methode [Android],143
 - createObjectStoreTransfer-Methode [BlackBerry],144
 - createSISHTTPListener-Methode [BlackBerry],144
 - release-Methode,145
 - setErrorLanguage-Methode,145
- DatabaseManager-Objekte
 - UltraLiteJ,5
- DATE-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],154
- Datenänderung
 - UltraLiteJ mit SQL,9
- Datenbankschemata
 - UltraLiteJ-API, Zugriff,19
- Datenbankspeicher
 - Android- und BlackBerry-Speicher,5
- Datensynchronisation
 - UltraLite Java Edition-Datenbanken,23
- DecimalNumber-Schnittstelle [UltraLiteJ-API]
 - add-Methode,146
 - Beschreibung,146
 - divide-Methode,147
 - getString-Methode,147
 - isNull-Methode,148
 - multiply-Methode,148
 - set-Methode,148
 - setNull-Methode,148
 - subtract-Methode,149
- Deployment
 - UltraLiteJ-Anwendungen,27,29
- DESCENDING-Variable
 - IndexSchema-Schnittstelle [UltraLiteJ-API],179
- DISCONNECTING-Variable
 - SyncObserver.States-Schnittstelle [UltraLiteJ-API],248
- divide-Methode
 - DecimalNumber-Schnittstelle [UltraLiteJ-API],147
 - Unsigned64-Klasse [UltraLiteJ-API],290

DML

- UltraLiteJ,15

Domain-Schnittstelle [UltraLiteJ-API]

- Beschreibung,149
- BIG-Variable,153
- BINARY-Variable,153
- BIT-Variable,153
- DATE-Variable,154
- DOMAIN_MAX-Variable,154
- DOUBLE-Variable,154
- INTEGER-Variable,154
- LONGBINARY-Variable,155
- LONGBINARYFILE-Variable,155
- LONGVARCHAR-Variable,155
- NUMERIC-Variable,155
- REAL-Variable,156
- SHORT-Variable,156
- ST_GEOMETRY-Variable,156
- TIME-Variable,156
- TIMESTAMP-Variable,156
- TIMESTAMP_ZONE-Variable,157
- TINY-Variable,157
- UNSIGNED_BIG-Variable,157
- UNSIGNED_INTEGER-Variable,157
- UNSIGNED_SHORT-Variable,158
- UUID-Variable,158
- VARCHAR-Variable,158

DOMAIN_MAX-Variable

- Domain-Schnittstelle [UltraLiteJ-API],154

DONE-Variable

- SyncObserver.States-Schnittstelle [UltraLiteJ-API],248

DOUBLE-Variable

- Domain-Schnittstelle [UltraLiteJ-API],154

downloadFile-Methode

- FileTransfer-Schnittstelle [UltraLiteJ-API],161

dropDatabase-Methode

- Connection-Schnittstelle [UltraLiteJ-API],113

E

E2EE_RSA-Variable [BlackBerry]

- StreamHTTPParams-Schnittstelle [UltraLiteJ-API],238

emergencyShutdown-Methode [BlackBerry]

- Connection-Schnittstelle [UltraLiteJ-API],114

enableAesDBEncryption-Methode

- ConfigPersistent-Schnittstelle [UltraLiteJ-API],95

enableObfuscation-Methode

- ConfigPersistent-Schnittstelle [UltraLiteJ-API],95

Entwicklung

- UltraLiteJ,3,27,29

Entwicklungsplattformen

- UltraLiteJ,1

Ergebnismengen

- UltraLiteJ-Navigation,18

Ergebnismengenschemata

- UltraLiteJ,18

ERROR-Variable

- SyncObserver.States-Schnittstelle [UltraLiteJ-API],248

execute-Methode

- PreparedStatement-Schnittstelle [UltraLiteJ-API],182

executeQuery-Methode

- PreparedStatement-Schnittstelle [UltraLiteJ-API],183

EXPIRED-Variable

- SyncResult.AuthStatusCode-Schnittstelle [UltraLiteJ-API],279

F

Fehler

- UltraLiteJ-API-Behandlung,20

Fehlerbehandlung

- UltraLiteJ,20

- UltraLiteJ, Fehler beheben,20

Festschreiben

- UltraLiteJ-Transaktionen,15

FileTransfer-Schnittstelle [UltraLiteJ-API]

- Beschreibung,159

- downloadFile-Methode,161

- getAuthenticationParams-Methode,163

- getAuthStatus-Methode,163

- getAuthValue-Methode,163

- getFileAuthCode-Methode,163

- getLivenessTimeout-Methode,164

- getLocalFileName-Methode,164

- getLocalPath-Methode,164

- getPassword-Methode,165

- getRemoteKey-Methode,165

- getServerFileName-Methode,165

- getStreamErrorCode-Methode,166

- getStreamErrorMessage-Methode,166

- getStreamParams-Methode,166

- getUserName-Methode,167
- getVersion-Methode,167
- isResumePartialTransfer-Methode,167
- isTransferredFile-Methode,168
- setAuthenticationParms-Methode,168
- setLivenessTimeout-Methode,168
- setLocalFileName-Methode,169
- setLocalPath-Methode,170
- setPassword-Methode,170
- setRemoteKey-Methode,171
- setResumePartialTransfer-Methode,171
- setServerFileName-Methode,172
- setUserName-Methode,173
- setVersion-Methode,173
- uploadFile-Methode,174
- FileTransferProgressData-Schnittstelle [UltraLiteJ-API]
 - Beschreibung,175
 - getBytesTransferred-Methode,175
 - getFileSize-Methode,176
 - getResumedAtSize-Methode,176
- fileTransferProgressed-Methode
 - FileTransferProgressListener-Schnittstelle [UltraLiteJ-API],177
- FileTransferProgressListener-Schnittstelle [UltraLiteJ-API]
 - Beschreibung,176
 - fileTransferProgressed-Methode,177
- FINISHING_UPLOAD-Variable
 - SyncObserver.States-Schnittstelle [UltraLiteJ-API],248
- First-Methode
 - UltraLiteJ-Beispiel,18
- first-Methode [Android]
 - ResultSet-Schnittstelle [UltraLiteJ-API],200
- G**
- getAcknowledgeDownload-Methode
 - SyncParms-Klasse [UltraLiteJ-API],253
- getAdditionalParms-Methode [Android]
 - SyncParms-Klasse [UltraLiteJ-API],253
- getAliasName-Methode
 - ResultSetMetadata-Schnittstelle [UltraLiteJ-API],217
- getAuthenticationParms-Methode
 - FileTransfer-Schnittstelle [UltraLiteJ-API],163
 - SyncParms-Klasse [UltraLiteJ-API],254
- getAuthMessage-Methode
 - SyncResult-Klasse [UltraLiteJ-API],270
- getAuthStatus-Methode
 - FileTransfer-Schnittstelle [UltraLiteJ-API],163
 - SyncResult-Klasse [UltraLiteJ-API],271
- getAuthValue-Methode
 - FileTransfer-Schnittstelle [UltraLiteJ-API],163
 - SyncResult-Klasse [UltraLiteJ-API],271
- getBlobInputStream-Methode
 - ResultSet-Schnittstelle [UltraLiteJ-API],201
- getBlobOutputStream-Methode
 - PreparedStatement-Schnittstelle [UltraLiteJ-API],183
- getBoolean-Methode
 - ResultSet-Schnittstelle [UltraLiteJ-API],202
- getBytes-Methode
 - ResultSet-Schnittstelle [UltraLiteJ-API],203
- getBytesTransferred-Methode
 - FileTransferProgressData-Schnittstelle [UltraLiteJ-API],175
- getCacheSize-Methode
 - ConfigPersistent-Schnittstelle [UltraLiteJ-API],96
- getCausingException-Methode
 - ULjException-Klasse [UltraLiteJ-API],287
- getCertificateCompany-Methode
 - StreamHTTPSParms-Schnittstelle [UltraLiteJ-API],242
- getCertificateName-Methode
 - StreamHTTPSParms-Schnittstelle [UltraLiteJ-API],242
- getCertificateUnit-Methode
 - StreamHTTPSParms-Schnittstelle [UltraLiteJ-API],242
- getClobReader-Methode
 - ResultSet-Schnittstelle [UltraLiteJ-API],203
- getClobWriter-Methode
 - PreparedStatement-Schnittstelle [UltraLiteJ-API],184
- getColumnCount-Methode
 - ResultSetMetadata-Schnittstelle [UltraLiteJ-API],217
- getCommitCount-Methode [BlackBerry]
 - DatabaseInfo-Schnittstelle [UltraLiteJ-API],134
- getConnectionString-Methode [Android]
 - ConfigPersistent-Schnittstelle [UltraLiteJ-API],96
- getCorrelationName-Methode
 - ResultSetMetadata-Schnittstelle [UltraLiteJ-API],218

getCreationString-Methode [Android]	getErrorCode-Methode
ConfigPersistent-Schnittstelle [UltraLiteJ-API],96	ULjException-Klasse [UltraLiteJ-API],287
getCurrentTableName-Methode	getExtraParameters-Methode [Android]
SyncResult-Klasse [UltraLiteJ-API],271	StreamHTTPParams-Schnittstelle [UltraLiteJ-API],229
getCustomHTTPHeader-Methode [BlackBerry]	getFileAuthCode-Methode
StreamHTTPParams-Schnittstelle [UltraLiteJ-API],228	FileTransfer-Schnittstelle [UltraLiteJ-API],163
getDatabaseId-Methode [BlackBerry]	getFileSize-Methode
Connection-Schnittstelle [UltraLiteJ-API],114	FileTransferProgressData-Schnittstelle [UltraLiteJ-API],176
getDatabaseInfo-Methode	getFloat-Methode
Connection-Schnittstelle [UltraLiteJ-API],114	ResultSet-Schnittstelle [UltraLiteJ-API],207
getDatabaseName-Methode	getHost-Methode
Configuration-Schnittstelle [UltraLiteJ-API],103	StreamHTTPParams-Schnittstelle [UltraLiteJ-API],229
getDatabaseProperty-Methode	getHTTPPassword-Methode [BlackBerry]
Connection-Schnittstelle [UltraLiteJ-API],114	StreamHTTPParams-Schnittstelle [UltraLiteJ-API],230
getDate-Methode	getHTTPUserId-Methode [BlackBerry]
ResultSet-Schnittstelle [UltraLiteJ-API],204	StreamHTTPParams-Schnittstelle [UltraLiteJ-API],230
getDbFormat-Methode [BlackBerry]	getIgnoredRows-Methode
DatabaseInfo-Schnittstelle [UltraLiteJ-API],135	SyncResult-Klasse [UltraLiteJ-API],271
getDbSize-Methode [BlackBerry]	getInt-Methode
DatabaseInfo-Schnittstelle [UltraLiteJ-API],135	ResultSet-Schnittstelle [UltraLiteJ-API],208
getDecimalNumber-Methode	getKeepPartialDownload-Methode [Android]
ResultSet-Schnittstelle [UltraLiteJ-API],205	SyncParams-Klasse [UltraLiteJ-API],254
getDomainName-Methode	getLastDownloadTime-Methode
ResultSetMetadata-Schnittstelle [UltraLiteJ-API],218	Connection-Schnittstelle [UltraLiteJ-API],115
getDomainPrecision-Methode	getLastIdentity-Methode
ResultSetMetadata-Schnittstelle [UltraLiteJ-API],218	Connection-Schnittstelle [UltraLiteJ-API],115
getDomainScale-Methode	getLastWarning-Methode
ResultSetMetadata-Schnittstelle [UltraLiteJ-API],219	Connection-Schnittstelle [UltraLiteJ-API],116
getDomainSize-Methode	getLazyLoadIndexes-Methode [BlackBerry]
ResultSetMetadata-Schnittstelle [UltraLiteJ-API],219	ConfigPersistent-Schnittstelle [UltraLiteJ-API],97
getDomainType-Methode	getLivenessTimeout-Methode
ResultSetMetadata-Schnittstelle [UltraLiteJ-API],219	FileTransfer-Schnittstelle [UltraLiteJ-API],164
getDouble-Methode	SyncParams-Klasse [UltraLiteJ-API],254
ResultSet-Schnittstelle [UltraLiteJ-API],206	getLocalFileName-Methode
getE2eePublicKey-Methode [BlackBerry]	FileTransfer-Schnittstelle [UltraLiteJ-API],164
StreamHTTPParams-Schnittstelle [UltraLiteJ-API],229	getLocalPath-Methode
getE2eeType-Methode [BlackBerry]	FileTransfer-Schnittstelle [UltraLiteJ-API],164
StreamHTTPParams-Schnittstelle [UltraLiteJ-API],229	getLogSize-Methode [BlackBerry]
getEncryptionKey-Methode	DatabaseInfo-Schnittstelle [UltraLiteJ-API],135
ConfigPersistent-Schnittstelle [UltraLiteJ-API],97	getLong-Methode
	ResultSet-Schnittstelle [UltraLiteJ-API],209
	getMessage-Methode
	SQLInfo-Schnittstelle [Android] [UltraLiteJ],224

- getNewPassword-Methode
 - SyncParams-Klasse [UltraLiteJ-API],255
- getNumberRowsToUpload-Methode
 - DatabaseInfo-Schnittstelle [UltraLiteJ-API],135
- getOption-Methode [BlackBerry]
 - Connection-Schnittstelle [UltraLiteJ-API],116
- getOrdinal-Methode
 - PreparedStatement-Schnittstelle [UltraLiteJ-API],185
 - ResultSet-Schnittstelle [UltraLiteJ-API],210
- getOutputBufferSize-Methode
 - StreamHTTPParams-Schnittstelle [UltraLiteJ-API],230
- getPageReads-Methode
 - DatabaseInfo-Schnittstelle [UltraLiteJ-API],136
- getPageSize-Methode
 - Configuration-Schnittstelle [UltraLiteJ-API],103
 - DatabaseInfo-Schnittstelle [UltraLiteJ-API],137
- getPageWrites-Methode
 - DatabaseInfo-Schnittstelle [UltraLiteJ-API],137
- getParameter-Methode
 - SQLInfo-Schnittstelle [Android] [UltraLiteJ-API],224
 - ULjEvent-Schnittstelle [Android] [UltraLiteJ-API],285
- getParameter-Methode [Android]
 - ULjException-Klasse [UltraLiteJ-API],287
- getParameterCount-Methode
 - SQLInfo-Schnittstelle [Android] [UltraLiteJ-API],224
- getParameterCount-Methode [Android]
 - PreparedStatement-Schnittstelle [UltraLiteJ-API],185
 - ULjException-Klasse [UltraLiteJ-API],288
- getParameterType-Methode [Android]
 - PreparedStatement-Schnittstelle [UltraLiteJ-API],185
- getParams-Methode
 - ValidateDatabaseProgressData-Schnittstelle [Android] [UltraLiteJ-API],294
- getPartialDownloadRetained-Methode [Android]
 - SyncResult-Klasse [UltraLiteJ-API],272
- getPassword-Methode
 - FileTransfer-Schnittstelle [UltraLiteJ-API],165
 - SyncParams-Klasse [UltraLiteJ-API],255
- getPlan-Methode
 - PreparedStatement-Schnittstelle [UltraLiteJ-API],186
- getPlanTree-Methode
 - PreparedStatement-Schnittstelle [UltraLiteJ-API],187
- getPort-Methode
 - StreamHTTPParams-Schnittstelle [UltraLiteJ-API],231
- getPublications-Methode
 - SyncParams-Klasse [UltraLiteJ-API],255
- getQualifiedName-Methode
 - ResultSetMetadata-Schnittstelle [UltraLiteJ-API],220
- getReceivedByteCount-Methode
 - SyncResult-Klasse [UltraLiteJ-API],272
- getReceivedDeletes-Methode
 - SyncResult-Klasse [UltraLiteJ-API],272
- getReceivedIgnoredDeletes-Methode
 - SyncResult-Klasse [UltraLiteJ-API],272
- getReceivedIgnoredUpdates-Methode
 - SyncResult-Klasse [UltraLiteJ-API],273
- getReceivedInserts-Methode
 - SyncResult-Klasse [UltraLiteJ-API],273
- getReceivedRowCount-Methode
 - SyncResult-Klasse [UltraLiteJ-API],274
- getReceivedTruncateDeletes-Methode
 - SyncResult-Klasse [UltraLiteJ-API],274
- getReceivedUpdates-Methode
 - SyncResult-Klasse [UltraLiteJ-API],275
- getRelease-Methode
 - DatabaseInfo-Schnittstelle [UltraLiteJ-API],137
- getRemoteKey-Methode
 - FileTransfer-Schnittstelle [UltraLiteJ-API],165
- getResultSet-Methode
 - PreparedStatement-Schnittstelle [UltraLiteJ-API],188
- getResultSetMetadata-Methode
 - ResultSet-Schnittstelle [UltraLiteJ-API],210
- getResumedAtSize-Methode
 - FileTransferProgressData-Schnittstelle [UltraLiteJ-API],176
- getResumePartialDownload-Methode [Android]
 - SyncParams-Klasse [UltraLiteJ-API],256
- getRowCount-Methode [Android]
 - ResultSet-Schnittstelle [UltraLiteJ-API],211
- getRowScoreFlushSize-Methode [BlackBerry]
 - ConfigPersistent-Schnittstelle [UltraLiteJ-API],97
- getRowScoreMaximum-Methode [BlackBerry]
 - ConfigPersistent-Schnittstelle [UltraLiteJ-API],97
- getSentByteCount-Methode

SyncResult-Klasse [UltraLiteJ-API],275
 getSentDeletes-Methode
 SyncResult-Klasse [UltraLiteJ-API],275
 getSentInserts-Methode
 SyncResult-Klasse [UltraLiteJ-API],276
 getSentUpdates-Methode
 SyncResult-Klasse [UltraLiteJ-API],276
 getServerFileName-Methode
 FileTransfer-Schnittstelle [UltraLiteJ-API],165
 getSize-Methode
 ResultSet-Schnittstelle [UltraLiteJ-API],211
 getSQLCode-Methode
 SQLInfo-Schnittstelle [Android] [UltraLiteJ-API],225
 getSQLCount-Methode
 SQLInfo-Schnittstelle [Android] [UltraLiteJ-API],225
 getSqlOffset-Methode
 ULjException-Klasse [UltraLiteJ-API],288
 getState, Methode
 Connection, Schnittstelle [UltraLiteJ-API],117
 getStatusId-Methode
 ValidateDatabaseProgressData-Schnittstelle [Android] [UltraLiteJ-API],295
 getStreamErrorCode-Methode
 FileTransfer-Schnittstelle [UltraLiteJ-API],166
 SyncResult-Klasse [UltraLiteJ-API],277
 getStreamErrorMessage-Methode
 FileTransfer-Schnittstelle [UltraLiteJ-API],166
 SyncResult-Klasse [UltraLiteJ-API],277
 getStreamParams-Methode
 FileTransfer-Schnittstelle [UltraLiteJ-API],166
 SyncParams-Klasse [UltraLiteJ-API],256
 getString-Methode
 DecimalNumber-Schnittstelle [UltraLiteJ-API],147
 ResultSet-Schnittstelle [UltraLiteJ-API],212
 UUIDValue-Schnittstelle [UltraLiteJ-API],293
 getSyncedTableCount-Methode
 SyncResult-Klasse [UltraLiteJ-API],277
 getSyncObserver-Methode
 Connection-Schnittstelle [UltraLiteJ-API],117
 SyncParams-Klasse [UltraLiteJ-API],256
 getSyncResult-Methode
 Connection-Schnittstelle [UltraLiteJ-API],117
 SyncParams-Klasse [UltraLiteJ-API],257
 getTableName-Methode
 ResultSetMetadata-Schnittstelle [UltraLiteJ-API],220
 getTableName-Methode
 ResultSetMetadata-Schnittstelle [UltraLiteJ-API],221
 getTableOrder-Methode
 SyncParams-Klasse [UltraLiteJ-API],257
 getTotalDownloadRowCount-Methode
 SyncResult-Klasse [UltraLiteJ-API],277
 getTotalTableCount-Methode
 SyncResult-Klasse [UltraLiteJ-API],278
 getTrustedCertificates-Methode
 StreamHTTPSParms-Schnittstelle [UltraLiteJ-API],242
 getType-Methode
 ULjEvent-Schnittstelle [Android] [UltraLiteJ-API],286
 getUpdateCount-Methode
 PreparedStatement-Schnittstelle [UltraLiteJ-API],188
 getURLSuffix-Methode
 StreamHTTPParams-Schnittstelle [UltraLiteJ-API],231
 getUsername-Methode
 FileTransfer-Schnittstelle [UltraLiteJ-API],167
 SyncParams-Klasse [UltraLiteJ-API],258
 getUsername-Methode [Android]
 ConfigPersistent-Schnittstelle [UltraLiteJ-API],98
 getUUIDValue-Methode
 ResultSet-Schnittstelle [UltraLiteJ-API],213
 getVersion-Methode
 FileTransfer-Schnittstelle [UltraLiteJ-API],167
 SyncParams-Klasse [UltraLiteJ-API],258
 getWrittenName-Methode
 ResultSetMetadata-Schnittstelle [UltraLiteJ-API],221
 Gleichzeitige Synchronisation
 BlackBerry,23

H

hasResultSet-Methode
 PreparedStatement-Schnittstelle [UltraLiteJ-API],188
 hasShadowPaging-Methode [BlackBerry]
 ConfigPersistent-Schnittstelle [UltraLiteJ-API],98
 HTTP_STREAM-Variable
 SyncParams-Klasse [UltraLiteJ-API],268
 HTTPS_STREAM-Variable
 SyncParams-Klasse [UltraLiteJ-API],268

I

- IN_USE-Variable
 - SyncResult.AuthStatusCode-Schnittstelle [UltraLiteJ-API],279
- Index.Schema-Objekt
 - UltraLiteJ-Entwicklung,20
- Index.Schema-Schnittstelle [UltraLiteJ-API]
 - ASCENDING-Variable,179
 - Beschreibung,178
 - DESCENDING-Variable,179
 - PERSISTENT-Variable,179
 - PRIMARY_INDEX-Variable,179
 - UNIQUE_INDEX-Variable,180
 - UNIQUE_KEY-Variable,180
- Indizes
 - UltraLiteJ-API-Schemainformationen,20
- Inner-Joins
 - Beispielcode,34
- INTEGER-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],154
- INVALID-Variable
 - SyncResult.AuthStatusCode-Schnittstelle [UltraLiteJ-API],279
- isDownloadOnly-Methode
 - SyncParms-Klasse [UltraLiteJ-API],258
- isNull-Methode
 - DecimalNumber-Schnittstelle [UltraLiteJ-API],148
 - ResultSet-Schnittstelle [UltraLiteJ-API],214
 - UUIDValue-Schnittstelle [UltraLiteJ-API],293
- isPingOnly-Methode
 - SyncParms-Klasse [UltraLiteJ-API],258
- isRestartable-Methode
 - StreamHTTPParms-Schnittstelle [UltraLiteJ-API],231
- isResumePartialTransfer-Methode
 - FileTransfer-Schnittstelle [UltraLiteJ-API],167
- isSynchronizationDeleteDisabled-Methode [BlackBerry]
 - Connection-Schnittstelle [UltraLiteJ-API],118
- isTransferredFile-Methode
 - FileTransfer-Schnittstelle [UltraLiteJ-API],168
- isUploadOK-Methode
 - SyncResult-Klasse [UltraLiteJ-API],278
- isUploadOnly-Methode
 - SyncParms-Klasse [UltraLiteJ-API],259

L

- Last-Methode
 - UltraLiteJ-Beispiel,18
- last-Methode [Android]
 - ResultSet-Schnittstelle [UltraLiteJ-API],215
- LONGBINARY-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],155
- LONGBINARYFILE-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],155
- LONGVARCHAR-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],155

M

- multiply-Methode
 - DecimalNumber-Schnittstelle [UltraLiteJ-API],148
 - Unsigned64-Klasse [UltraLiteJ-API],290

N

- Navigation in SQL-Ergebnismengen
 - UltraLiteJ,18
- next-Methode
 - ResultSet-Schnittstelle [UltraLiteJ-API],215
 - UltraLiteJ-Beispiel,18
- NOT_CONNECTED-Variable
 - Connection-Schnittstelle [UltraLiteJ-API],125
- NUMERIC-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],155

O

- onError-Methode
 - SISRequestHandler-Schnittstelle [BlackBerry] [UltraLiteJ-API],223
- onRequest-Methode
 - SISRequestHandler-Schnittstelle [BlackBerry] [UltraLiteJ-API],223
- OPTION_BLOB_FILE_BASE_DIR-Variable [BlackBerry]
 - Connection-Schnittstelle [UltraLiteJ-API],125
- OPTION_DATABASE_ID-Variable [BlackBerry]
 - Connection-Schnittstelle [UltraLiteJ-API],125
- OPTION_DATE_FORMAT, Variable
 - Connection, Schnittstelle [UltraLiteJ-API],125
- OPTION_DATE_ORDER, Variable
 - Connection, Schnittstelle [UltraLiteJ-API],126
- OPTION_MAX_HASH_SIZE-Variable
 - Connection-Schnittstelle [UltraLiteJ-API],126
- OPTION_ML_REMOTE_ID-Variable [BlackBerry]

- Connection-Schnittstelle [UltraLiteJ-API],127
- OPTION_ML_SERVER_VERSION-Variable [BlackBerry]
 - Connection-Schnittstelle [UltraLiteJ-API],127
- OPTION_NEAREST_CENTURY, Variable
 - Connection, Schnittstelle [UltraLiteJ-API],127
- OPTION_PRECISION, Variable
 - Connection, Schnittstelle [UltraLiteJ-API],128
- OPTION_SCALE, Variable
 - Connection, Schnittstelle [UltraLiteJ-API],128
- OPTION_TIME_FORMAT, Variable
 - Connection, Schnittstelle [UltraLiteJ-API],129
- OPTION_TIMESTAMP_FORMAT, Variable
 - Connection, Schnittstelle [UltraLiteJ-API],129
- OPTION_TIMESTAMP_INCREMENT, Variable
 - Connection, Schnittstelle [UltraLiteJ-API],130
- OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT-Variable
 - Connection-Schnittstelle [UltraLiteJ-API],130

P

- PERSISTENT-Variable
 - IndexSchema-Schnittstelle [UltraLiteJ-API],179
- Plattformen
 - unterstützt in UltraLiteJ,1
- Praktische Einführungen
 - eine Android-Anwendung erstellen,41
 - eine BlackBerry-Anwendung erstellen,49
- preparedStatement, Schnittstelle
 - UltraLiteJ,15
- PreparedStatement-Schnittstelle [UltraLiteJ-API]
 - Beschreibung,180
 - close-Methode,182
 - execute-Methode,182
 - executeQuery-Methode,183
 - getBlobOutputStream-Methode,183
 - getClobWriter-Methode,184
 - getOrdinal-Methode,185
 - getParameterCount-Methode [Android],185
 - getParameterType-Methode [Android],185
 - getPlan-Methode,186
 - getPlanTree-Methode,187
 - getResultSet-Methode,188
 - getUpdateCount-Methode,188
 - hasResultSet-Methode,188
 - set-Methode,189
 - setNull-Methode,197

- prepareStatement-Methode
 - Connection-Schnittstelle [UltraLiteJ-API],118
- previous-Methode
 - ResultSet-Schnittstelle [UltraLiteJ-API],215
 - UltraLiteJ-Beispiel,18
- PRIMARY_INDEX-Variable
 - IndexSchema-Schnittstelle [UltraLiteJ-API],179
- Programmcode
 - BlackBerry-Anwendung, praktische Einführung,74
- PROPERTY_DATABASE_NAME, Variable
 - Connection, Schnittstelle [UltraLiteJ-API],130
- PROPERTY_PAGE_SIZE, Variable
 - Connection, Schnittstelle [UltraLiteJ-API],131

R

- REAL-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],156
- RECEIVING_DATA-Variable
 - SyncObserver.States-Schnittstelle [UltraLiteJ-API],248
- RECEIVING_TABLE-Variable
 - SyncObserver.States-Schnittstelle [UltraLiteJ-API],248
- RECEIVING_UPLOAD_ACK-Variable
 - SyncObserver.States-Schnittstelle [UltraLiteJ-API],249
- registerForEvent-Methode [Android]
 - Connection-Schnittstelle [UltraLiteJ-API],119
- Relative-Methode
 - UltraLiteJ-Beispiel,18
- relative-Methode [Android]
 - ResultSet-Schnittstelle [UltraLiteJ-API],215
- release-Methode
 - Connection-Schnittstelle [UltraLiteJ-API],119
 - DatabaseManager-Klasse [UltraLiteJ-API],145
- remainder-Methode
 - Unsigned64-Klasse [UltraLiteJ-API],291
- resetLastDownloadTime-Methode
 - Connection-Schnittstelle [UltraLiteJ-API],119
- ResultSet-Objekt
 - UltraLiteJ-Datenabfragebeispiel,18
- ResultSet-Schnittstelle [UltraLiteJ-API]
 - afterLast-Methode [Android],200
 - beforeFirst-Methode [Android],200
 - Beschreibung,197
 - close-Methode,200
 - first-Methode [Android],200

- getBlobInputStream-Methode,201
- getBoolean-Methode,202
- getBytes-Methode,203
- getClobReader-Methode,203
- getDate-Methode,204
- getDecimalNumber-Methode,205
- getDouble-Methode,206
- getFloat-Methode,207
- getInt-Methode,208
- getLong-Methode,209
- getOrdinal-Methode,210
- getResultSetMetadata-Methode,210
- getRowCount-Methode [Android],211
- getSize-Methode,211
- getString-Methode,212
- getUUIDValue-Methode,213
- isNull-Methode,214
- last-Methode [Android],215
- next-Methode,215
- previous-Methode,215
- relative-Methode [Android],215
- ResultSetMetadata-Schnittstelle [UltraLiteJ-API]
 - Beschreibung,216
 - getAliasName-Methode,217
 - getColumnCount-Methode,217
 - getCorrelationName-Methode,218
 - getDomainName-Methode,218
 - getDomainPrecision-Methode,218
 - getDomainScale-Methode,219
 - getDomainSize-Methode,219
 - getDomainType-Methode,219
 - getQualifiedName-Methode,220
 - getTableColumnName-Methode,220
 - getTableName-Methode,221
 - getWrittenName-Methode,221
- rollback-Methode
 - Connection-Schnittstelle [UltraLiteJ-API],120
 - UltraLiteJ-Transaktionen,15
- rollbackPartialDownload-Methode [Android]
 - Connection-Schnittstelle [UltraLiteJ-API],120
- Rollbacks
 - UltraLiteJ-Transaktionen,15
- ROLLING_BACK_DOWNLOAD-Variable
 - SyncObserver.States-Schnittstelle [UltraLiteJ-API],249

S

- Schemata
 - UltraLiteJ-API, Zugriff,19
- SELECT-Anweisung
 - UltraLiteJ-Datenabfragebeispiel,18
- SENDING_DATA-Variable
 - SyncObserver.States-Schnittstelle [UltraLiteJ-API],249
- SENDING_DOWNLOAD_ACK-Variable
 - SyncObserver.States-Schnittstelle [UltraLiteJ-API],249
- SENDING_HEADER-Variable
 - SyncObserver.States-Schnittstelle [UltraLiteJ-API],249
- SENDING_TABLE-Variable
 - SyncObserver.States-Schnittstelle [UltraLiteJ-API],250
- set-Methode
 - DecimalNumber-Schnittstelle [UltraLiteJ-API],148
 - PreparedStatement-Schnittstelle [UltraLiteJ-API],189
 - UUIDValue-Schnittstelle [UltraLiteJ-API],293
- setAcknowledgeDownload-Methode
 - SyncParms-Klasse [UltraLiteJ-API],259
- setAdditionalParms-Methode [Android]
 - SyncParms-Klasse [UltraLiteJ-API],259
- setAuthenticationParms-Methode
 - FileTransfer-Schnittstelle [UltraLiteJ-API],168
 - SyncParms-Klasse [UltraLiteJ-API],260
- setCacheSize-Methode
 - ConfigPersistent-Schnittstelle [UltraLiteJ-API],98
- setCertificateCompany-Methode
 - StreamHTTPSParms-Schnittstelle [UltraLiteJ-API],243
- setCertificateName-Methode
 - StreamHTTPSParms-Schnittstelle [UltraLiteJ-API],243
- setCertificateUnit-Methode
 - StreamHTTPSParms-Schnittstelle [UltraLiteJ-API],243
- setConnectionString-Methode [Android]
 - ConfigPersistent-Schnittstelle [UltraLiteJ-API],99
- setCreationString-Methode [Android]
 - ConfigPersistent-Schnittstelle [UltraLiteJ-API],99
- setDatabaseId-Methode
 - Connection-Schnittstelle [UltraLiteJ-API],120
- setDatabaseName-Methode

Configuration-Schnittstelle [UltraLiteJ-API],103	Configuration-Schnittstelle [UltraLiteJ-API],104
setDownloadOnly-Methode	FileTransfer-Schnittstelle [UltraLiteJ-API],170
SyncParms-Klasse [UltraLiteJ-API],261	SyncParms-Klasse [UltraLiteJ-API],263
setE2eePublicKey-Methode [BlackBerry]	setPingOnly-Methode
StreamHTTPParms-Schnittstelle [UltraLiteJ-API],232	SyncParms-Klasse [UltraLiteJ-API],264
setE2eeType-Methode [BlackBerry]	setPort-Methode
StreamHTTPParms-Schnittstelle [UltraLiteJ-API],233	StreamHTTPParms-Schnittstelle [UltraLiteJ-API],235
setEncryptionKey-Methode	setPublications-Methode
ConfigPersistent-Schnittstelle [UltraLiteJ-API],100	SyncParms-Klasse [UltraLiteJ-API],264
setErrorLanguage-Methode	setRemoteKey-Methode
DatabaseManager-Klasse [UltraLiteJ-API],145	FileTransfer-Schnittstelle [UltraLiteJ-API],171
setExtraParameters-Methode [Android]	setRestartable-Methode
StreamHTTPParms-Schnittstelle [UltraLiteJ-API],233	StreamHTTPParms-Schnittstelle [UltraLiteJ-API],235
setHost-Methode	setResumePartialDownload-Methode [Android]
StreamHTTPParms-Schnittstelle [UltraLiteJ-API],233	SyncParms-Klasse [UltraLiteJ-API],265
setHTTPUserIdAndPassword-Methode [BlackBerry]	setResumePartialTransfer-Methode
StreamHTTPParms-Schnittstelle [UltraLiteJ-API],234	FileTransfer-Schnittstelle [UltraLiteJ-API],171
setKeepPartialDownload-Methode [Android]	setRowScoreFlushSize-Methode [BlackBerry]
SyncParms-Klasse [UltraLiteJ-API],261	ConfigPersistent-Schnittstelle [UltraLiteJ-API],101
setLazyLoadIndexes-Methode [BlackBerry]	setRowScoreMaximum-Methode [BlackBerry]
ConfigPersistent-Schnittstelle [UltraLiteJ-API],100	ConfigPersistent-Schnittstelle [UltraLiteJ-API],101
setLivenessTimeout-Methode	setServerFileName-Methode
FileTransfer-Schnittstelle [UltraLiteJ-API],168	FileTransfer-Schnittstelle [UltraLiteJ-API],172
SyncParms-Klasse [UltraLiteJ-API],262	setSyncObserver-Methode
setLocalFileName-Methode	Connection-Schnittstelle [UltraLiteJ-API],122
FileTransfer-Schnittstelle [UltraLiteJ-API],169	SyncParms-Klasse [UltraLiteJ-API],265
setLocalPath-Methode	setTableOrder-Methode
FileTransfer-Schnittstelle [UltraLiteJ-API],170	SyncParms-Klasse [UltraLiteJ-API],266
setNewPassword-Methode	setTrustedCertificates-Methode
SyncParms-Klasse [UltraLiteJ-API],263	StreamHTTPSParms-Schnittstelle [UltraLiteJ-API],244
setNull-Methode	setUploadOnly-Methode
DecimalNumber-Schnittstelle [UltraLiteJ-API],148	SyncParms-Klasse [UltraLiteJ-API],266
PreparedStatement-Schnittstelle [UltraLiteJ-API],197	setURLSuffix-Methode
UUIDValue-Schnittstelle [UltraLiteJ-API],294	StreamHTTPParms-Schnittstelle [UltraLiteJ-API],236
setOption-Methode	setUserName-Methode
Connection-Schnittstelle [UltraLiteJ-API],121	FileTransfer-Schnittstelle [UltraLiteJ-API],173
setOutputBufferSize-Methode	SyncParms-Klasse [UltraLiteJ-API],267
StreamHTTPParms-Schnittstelle [UltraLiteJ-API],234	setUserName-Methode [Android]
setPageSize-Methode	ConfigPersistent-Schnittstelle [UltraLiteJ-API],102
Configuration-Schnittstelle [UltraLiteJ-API],104	setVersion-Methode
setPassword-Methode	FileTransfer-Schnittstelle [UltraLiteJ-API],173
	SyncParms-Klasse [UltraLiteJ-API],267
	setZlibCompression-Methode

- StreamHTTPParms-Schnittstelle [UltraLiteJ-API],236
- setZlibDownloadWindowSize-Methode
 - StreamHTTPParms-Schnittstelle [UltraLiteJ-API],237
- setZlibUploadWindowSize-Methode
 - StreamHTTPParms-Schnittstelle [UltraLiteJ-API],237
- SHORT-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],156
- SISListener-Schnittstelle [BlackBerry] [UltraLiteJ-API]
 - Beschreibung,221
 - startListening-Methode,222
 - stopListening-Methode,222
- SISRequestHandler-Schnittstelle [BlackBerry] [UltraLiteJ-API]
 - Beschreibung,222
 - onError-Methode,223
 - onRequest-Methode,223
- Spalten
 - UltraLiteJ-API-Schemainformationen,20
- SQLCODE
 - UltraLiteJ-Fehlerbehandlung,20
- SQLInfo-Schnittstelle [Android] [UltraLiteJ-API]
 - Beschreibung,223
 - getParameter-Methode,224
 - getParameterCount-Methode,224
 - getSQLCode-Methode,225
 - getSQLCount-Methode,225
- SQLInfo-Schnittstelle [Android] [UltraLiteJ]
 - getMessage-Methode,224
- ST_GEOMETRY-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],156
- STARTING-Variable
 - SyncObserver.States-Schnittstelle [UltraLiteJ-API],250
- startListening-Methode
 - SISListener-Schnittstelle [BlackBerry] [UltraLiteJ-API],222
- stopListening-Methode
 - SISListener-Schnittstelle [BlackBerry] [UltraLiteJ-API],222
- StreamHTTPParms-Schnittstelle [UltraLiteJ-API]
 - addCustomHTTPHeader-Methode [BlackBerry],228
 - Beschreibung,225
 - E2EE_RSA-Variable [BlackBerry],238
 - getCustomHTTPHeaders-Methode [BlackBerry],228
 - getE2eePublicKey-Methode [BlackBerry],229
 - getE2eeType-Methode [BlackBerry],229
 - getExtraParameters-Methode [Android],229
 - getHost-Methode,229
 - getHTTPPassword-Methode [BlackBerry],230
 - getHTTPUserId-Methode [BlackBerry],230
 - getOutputBufferSize-Methode,230
 - getPort-Methode,231
 - getURLSuffix-Methode,231
 - isRestartable-Methode,231
 - setE2eePublicKey-Methode [BlackBerry],232
 - setE2eeType-Methode [BlackBerry],233
 - setExtraParameters-Methode [Android],233
 - setHost-Methode,233
 - setHTTPUserIdAndPassword-Methode [BlackBerry],234
 - setOutputBufferSize-Methode,234
 - setPort-Methode,235
 - setRestartable-Methode,235
 - setURLSuffix-Methode,236
 - setZlibCompression-Methode,236
 - setZlibDownloadWindowSize-Methode,237
 - setZlibUploadWindowSize-Methode,237
 - zlibCompressionEnabled-Methode,238
- StreamHTTPSPParms-Schnittstelle [UltraLiteJ-API]
 - Beschreibung,238
 - getCertificateCompany-Methode,242
 - getCertificateName-Methode,242
 - getCertificateUnit-Methode,242
 - getTrustedCertificates-Methode,242
 - setCertificateCompany-Methode,243
 - setCertificateName-Methode,243
 - setCertificateUnit-Methode,243
 - setTrustedCertificates-Methode,244
- subtract-Methode
 - DecimalNumber-Schnittstelle [UltraLiteJ-API],149
 - Unsigned64-Klasse [UltraLiteJ-API],292
- SYNC_ALL-Variable
 - Connection-Schnittstelle [UltraLiteJ-API],131
- SYNC_ALL_DB_PUB_NAME-Variable
 - Connection-Schnittstelle [UltraLiteJ-API],131
- SYNC_ALL_PUBS-Variable
 - Connection-Schnittstelle [UltraLiteJ-API],132
- SYNC_COMPLETE_EVENT-Variable
 - ULjEvent-Schnittstelle [Android] [UltraLiteJ-API],286

Synchronisation
 BlackBerry-Anwendung hinzufügen,65
 UltraLiteJ,21
 zu BlackBerry-Anwendung hinzufügen,41
synchronize-Methode
 Connection-Schnittstelle [UltraLiteJ-API],122
SyncObserver-Schnittstelle [UltraLiteJ-API]
 Beschreibung,244
 syncProgress-Methode,245
SyncObserver.States-Schnittstelle [UltraLiteJ-API]
 Beschreibung,246
 COMMITTING_DOWNLOAD-Variable,247
 CONNECTING-Variable,247
 DISCONNECTING-Variable,248
 DONE-Variable,248
 ERROR-Variable,248
 FINISHING_UPLOAD-Variable,248
 RECEIVING_DATA-Variable,248
 RECEIVING_TABLE-Variable,248
 RECEIVING_UPLOAD_ACK-Variable,249
 ROLLING_BACK_DOWNLOAD-Variable,249
 SENDING_DATA-Variable,249
 SENDING_DOWNLOAD_ACK-Variable,249
 SENDING_HEADER-Variable,249
 SENDING_TABLE-Variable,250
 STARTING-Variable,250
SyncParms-Klasse [UltraLiteJ-API]
 Beschreibung,250
 getAcknowledgeDownload-Methode,253
 getAdditionalParms-Methode [Android],253
 getAuthenticationParms-Methode,254
 getKeepPartialDownload-Methode [Android],254
 getLivenessTimeout-Methode,254
 getNewPassword-Methode,255
 getPassword-Methode,255
 getPublications-Methode,255
 getResumePartialDownload-Methode
 [Android],256
 getStreamParms-Methode,256
 getSyncObserver-Methode,256
 getSyncResult-Methode,257
 getTableOrder-Methode,257
 getUserName-Methode,258
 getVersion-Methode,258
 HTTP_STREAM-Variable,268
 HTTPS_STREAM-Variable,268
 isDownloadOnly-Methode,258
 isPingOnly-Methode,258
 isUploadOnly-Methode,259
 setAcknowledgeDownload-Methode,259
 setAdditionalParms-Methode [Android],259
 setAuthenticationParms-Methode,260
 setDownloadOnly-Methode,261
 setKeepPartialDownload-Methode [Android],261
 setLivenessTimeout-Methode,262
 setNewPassword-Methode,263
 setPassword-Methode,263
 setPingOnly-Methode,264
 setPublications-Methode,264
 setResumePartialDownload-Methode
 [Android],265
 setSyncObserver-Methode,265
 setTableOrder-Methode,266
 setUploadOnly-Methode,266
 setUserName-Methode,267
 setVersion-Methode,267
syncProgress-Methode
 SyncObserver-Schnittstelle [UltraLiteJ-API],245
SyncResult-Klasse [UltraLiteJ-API]
 Beschreibung,268
 getAuthMessage-Methode,270
 getAuthStatus-Methode,271
 getAuthValue-Methode,271
 getCurrentTableName-Methode,271
 getIgnoredRows-Methode,271
 getPartialDownloadRetained-Methode
 [Android],272
 getReceivedByteCount-Methode,272
 getReceivedDeletes-Methode,272
 getReceivedIgnoredDeletes-Methode,272
 getReceivedIgnoredUpdates-Methode,273
 getReceivedInserts-Methode,273
 getReceivedRowCount-Methode,274
 getReceivedTruncateDeletes-Methode,274
 getReceivedUpdates-Methode,275
 getSentByteCount-Methode,275
 getSentDeletes-Methode,275
 getSentInserts-Methode,276
 getSentUpdates-Methode,276
 getStreamErrorCode-Methode,277
 getStreamErrorMessage-Methode,277
 getSyncedTableCount-Methode,277
 getTotalDownloadRowCount-Methode,277
 getTotalTableCount-Methode,278
 isUploadOK-Methode,278

SyncResult.AuthStatusCode-Schnittstelle [UltraLiteJ-API]

Beschreibung,278

EXPIRED-Variable,279

IN_USE-Variable,279

INVALID-Variable,279

UNKNOWN-Variable,280

VALID-Variable,280

VALID_BUT_EXPIRES_SOON-Variable,280

SYS_ARTICLES-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],282

SYS_COLUMNS-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],282

SYS_FKEY_COLUMNS-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],282

SYS_FOREIGN_KEYS-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],282

SYS_INDEX_COLUMNS-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],282

SYS_INDEXES-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],283

SYS_INTERNAL, Variable

TableSchema, Schnittstelle [UltraLiteJ-API],283

SYS_PRIMARY_INDEX-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],283

SYS_PUBLICATIONS-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],283

SYS_TABLES-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],283

SYS_ULDATA-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],283

SYS_ULDATA_INTERNAL-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],284

SYS_ULDATA_OPTION-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],284

SYS_ULDATA_PROPERTY-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],284

Systemtabellen

UltraLiteJ,37

T

Tabellen

UltraLiteJ-API-Schemainformationen,20

TABLE_IS_DOWNLOAD_ONLY-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],284

TABLE_IS_NOSYNC-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],284

TABLE_IS_SYSTEM-Variable

TableSchema-Schnittstelle [UltraLiteJ-API],285

TABLE_MODIFIED_EVENT-Variable

ULjEvent-Schnittstelle [Android] [UltraLiteJ-API],286

TableSchema, Schnittstelle [UltraLiteJ-API]

SYS_INTERNAL, Variable,283

TableSchema-Objekt

UltraLiteJ-Entwicklung,20

TableSchema-Schnittstelle [UltraLiteJ-API]

Beschreibung,280

SYS_ARTICLES-Variable,282

SYS_COLUMNS-Variable,282

SYS_FKEY_COLUMNS-Variable,282

SYS_FOREIGN_KEYS-Variable,282

SYS_INDEX_COLUMNS-Variable,282

SYS_INDEXES-Variable,283

SYS_PRIMARY_INDEX-Variable,283

SYS_PUBLICATIONS-Variable,283

SYS_TABLES-Variable,283

SYS_ULDATA-Variable,283

SYS_ULDATA_INTERNAL-Variable,284

SYS_ULDATA_OPTION-Variable,284

SYS_ULDATA_PROPERTY-Variable,284

TABLE_IS_DOWNLOAD_ONLY-Variable,284

TABLE_IS_NOSYNC-Variable,284

TABLE_IS_SYSTEM-Variable,285

TIME-Variable

Domain-Schnittstelle [UltraLiteJ-API],156

TIMESTAMP-Variable

Domain-Schnittstelle [UltraLiteJ-API],156

TIMESTAMP_ZONE-Variable

Domain-Schnittstelle [UltraLiteJ-API],157

TINY-Variable

Domain-Schnittstelle [UltraLiteJ-API],157

Transaktionen

UltraLiteJ-Verwaltung,15

Transaktionsverarbeitung

UltraLiteJ-Verwaltung,15

U

UL_VALID_BAD_ROWID-Variable

ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],296

UL_VALID_CHECKING_INDEX-Variable

ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],296

- UL_VALID_CHECKING_PAGE-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],297
- UL_VALID_CHECKING_TABLE-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],297
- UL_VALID_CONNECT_ERROR-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],297
- UL_VALID_CORRUPT_PAGE-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],297
- UL_VALID_CORRUPT_PAGE_TABLE-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],298
- UL_VALID_DATABASE_ERROR-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],298
- UL_VALID_END-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],298
- UL_VALID_FAILED_CHECKSUM-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],299
- UL_VALID_INTERRUPTED-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],299
- UL_VALID_NO_ERROR-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],299
- UL_VALID_ROWCOUNT_MISMATCH-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],299
- UL_VALID_START-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],300
- UL_VALID_STARTUP_ERROR-Variable
 - ValidateDatabaseProgressData.StatusId-Schnittstelle [Android] [UltraLiteJ-API],300
- ULDatabaseSchema-Objekt
 - UltraLiteJ-Entwicklung,20
- ULjEvent-Schnittstelle [Android] [UltraLiteJ-API]
 - Beschreibung,285
 - COMMIT_EVENT-Variable,286
 - getParameter-Methode,285
 - getType-Methode,286
 - SYNC_COMPLETE_EVENT-Variable,286
 - TABLE_MODIFIED_EVENT-Variable,286
- ULjException-Klasse [UltraLiteJ-API]
 - Beschreibung,286
 - getCausingException-Methode,287
 - getErrorCode-Methode,287
 - getParameter-Methode [Android],287
 - getParameterCount-Methode [Android],288
 - getSqlOffset-Methode,288
- UltraLite Java Edition-Datenbanken
 - UltraLiteJ, Zugriff auf Informationen,19
 - verbinden in UltraLiteJ,5
- UltraLite, Datenbanken
 - in UltraLiteJ verbinden,5
- UltraLite-Datenbanken
 - UltraLiteJ-API, Zugriff auf Informationen,19
- UltraLiteJ
 - Android CustDB-Beispiel,30
 - Android-Anwendung, praktische Einführung,41
 - Architektur,81
 - Beispielcode,30
 - Beständigkeit,5
 - BlackBerry-Anwendung, praktische Einführung,49
 - BlackBerry-CustDB, Beispiel,24
 - Datenabfrage,18
 - Datenänderung,15
 - Datenänderung mit SQL,9
 - Datenbank erstellen,49
 - Deployment,27,29
 - Info,3
 - JAR-Ressourcendateien,4
 - Java Edition-Datenbanken,4
 - mit MobiLink synchronisieren,21
 - Netzwerkprotokolloptionen,23
 - Schnellstart,3
 - Systemtabellenschemas,37
 - Transaktionsverarbeitung,15
 - unterstützte Plattformen,1
 - Zugriff auf Schemainformationen,19
- UltraLiteJ-
 - Datenbankspeicher,5
- UltraLiteJ-API
 - ColumnSchema-Schnittstelle,81
 - ConfigFile-Schnittstelle,87
 - ConfigFileAndroid-Schnittstelle [Android],89
 - ConfigNonPersistent-Schnittstelle [BlackBerry],90
 - ConfigObjectStore-Schnittstelle [BlackBerry],91
 - ConfigPersistent-Schnittstelle,93
 - Configuration-Schnittstelle,102
 - Connection-Schnittstelle,105

- DatabaseInfo-Schnittstelle,133
- DatabaseManager-Klasse,138
- DecimalNumber-Schnittstelle,146
- Domain-Schnittstelle,149
- FileTransfer-Schnittstelle,159
- FileTransferProgressData-Schnittstelle,175
- FileTransferProgressListener-Schnittstelle,176
- IndexSchema-Schnittstelle,178
- PreparedStatement-Schnittstelle,180
- ResultSet-Schnittstelle,197
- ResultSetMetadata-Schnittstelle,216
- SISListener-Schnittstelle [BlackBerry],221
- SISRequestHandler-Schnittstelle [BlackBerry],222
- SQLInfo-Schnittstelle [Android],223
- StreamHTTPParms-Schnittstelle,225
- StreamHTTPSParms-Schnittstelle,238
- SyncObserver-Schnittstelle,244
- SyncObserver.States-Schnittstelle,246
- SyncParms-Klasse,250
- SyncResult-Klasse,268
- SyncResult.AuthStatusCode-Schnittstelle,278
- TableSchema-Schnittstelle,280
- ULjEvent-Schnittstelle [Android],285
- ULjException-Klasse,286
- Unsigned64-Klasse,288
- UUIDValue-Schnittstelle,292
- ValidateDatabaseProgressData-Schnittstelle [Android],294
- ValidateDatabaseProgressData.StatusId-Schnittstelle [Android],295
- ValidateDatabaseProgressListener-Schnittstelle [Android],300
- UltraLiteJ-API-Referenz
 - Package-Namen,81
- ULVF_DATABASE-Variable [Android]
 - Connection-Schnittstelle [UltraLiteJ-API],132
- ULVF_EXPRESS-Variable [Android]
 - Connection-Schnittstelle [UltraLiteJ-API],132
- ULVF_FULL_VALIDATE-Variable [Android]
 - Connection-Schnittstelle [UltraLiteJ-API],133
- ULVF_INDEX-Variable [Android]
 - Connection-Schnittstelle [UltraLiteJ-API],133
- ULVF_TABLE-Variable [Android]
 - Connection-Schnittstelle [UltraLiteJ-API],133
- UNIQUE_INDEX-Variable
 - IndexSchema-Schnittstelle [UltraLiteJ-API],180
- UNIQUE_KEY-Variable
 - IndexSchema-Schnittstelle [UltraLiteJ-API],180

- UNKNOWN-Variable
 - SyncResult.AuthStatusCode-Schnittstelle [UltraLiteJ-API],280
- unregisterForEvent-Methode [Android]
 - Connection-Schnittstelle [UltraLiteJ-API],123
- Unsigned64-Klasse [UltraLiteJ-API]
 - add-Methode,289
 - Beschreibung,288
 - compare-Methode,289
 - divide-Methode,290
 - multiply-Methode,290
 - remainder-Methode,291
 - subtract-Methode,292
- UNSIGNED_BIG-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],157
- UNSIGNED_INTEGER-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],157
- UNSIGNED_SHORT-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],158
- Unterstützte Plattformen
 - UltraLiteJ,1
- uploadFile-Methode
 - FileTransfer-Schnittstelle [UltraLiteJ-API],174
- UUID-Variable
 - Domain-Schnittstelle [UltraLiteJ-API],158
- UUIDValue-Schnittstelle [UltraLiteJ-API]
 - Beschreibung,292
 - getString-Methode,293
 - isNull-Methode,293
 - set-Methode,293
 - setNull-Methode,294

V

- VALID-Variable
 - SyncResult.AuthStatusCode-Schnittstelle [UltraLiteJ-API],280
- VALID_BUT_EXPIRES_SOON-Variable
 - SyncResult.AuthStatusCode-Schnittstelle [UltraLiteJ-API],280
- validateDatabase-Methode [Android]
 - Connection-Schnittstelle [UltraLiteJ-API],123
- ValidateDatabaseProgressData-Schnittstelle [Android] [UltraLiteJ-API]
 - Beschreibung,294
 - getStatusId-Methode,295
- ValidateDatabaseProgressData-Schnittstelle [Android] [UltraLiteJ-API]

getParms-Methode,294
 ValidateDatabaseProgressData.StatusId-Schnittstelle
 [Android] [UltraLiteJ-API]
 Beschreibung,295
 UL_VALID_BAD_ROWID-Variable,296
 UL_VALID_CHECKING_INDEX-Variable,296
 UL_VALID_CHECKING_PAGE-Variable,297
 UL_VALID_CHECKING_TABLE-Variable,297
 UL_VALID_CONNECT_ERROR-Variable,297
 UL_VALID_CORRUPT_PAGE-Variable,297
 UL_VALID_CORRUPT_PAGE_TABLE-
 Variable,298
 UL_VALID_DATABASE_ERROR-Variable,298
 UL_VALID_END-Variable,298
 UL_VALID_FAILED_CHECKSUM-Variable,299
 UL_VALID_INTERRUPTED-Variable,299
 UL_VALID_NO_ERROR-Variable,299
 UL_VALID_ROWCOUNT_MISMATCH-
 Variable,299
 UL_VALID_START-Variable,300
 UL_VALID_STARTUP_ERROR-Variable,300
 ValidateDatabaseProgressListener-Schnittstelle
 [Android] [UltraLiteJ-API]
 Beschreibung,300
 validateProgressed-Methode,300
 validateProgressed-Methode
 ValidateDatabaseProgressListener-Schnittstelle
 [Android] [UltraLiteJ-API],300
 VARCHAR-Variable
 Domain-Schnittstelle [UltraLiteJ-API],158
 Verbinden
 UltraLite- und UltraLite Java Edition-
 Datenbanken,5
 Verwalten
 UltraLiteJ-Transaktionen,15
 Vorbereitete Anweisungen
 UltraLiteJ,15

StreamHTTPParms-Schnittstelle [UltraLiteJ-
 API],238
 Zugriff auf Schemainformationen
 UltraLiteJ-Info,19

W

waitForEvent-Methode [Android]
 Connection-Schnittstelle [UltraLiteJ-API],124

Z

Zielplattformen
 UltraLiteJ,1
 zlibCompressionEnabled-Methode

