Contents

About this book .................................................................................................................. vii

System requirements and supported platforms ......................................................... 1

UltraLiteJ application development ............................................................................. 3

Quick start guide to UltraLiteJ application development ........................................... 3
Android and BlackBerry setup considerations ................................................................. 4
UltraLite and UltraLite Java edition database creation and connection approaches ........................................................................................................... 5
Quick start guide to schema operations and data management ............................... 8
Schema information access ......................................................................................... 19
Error handling .............................................................................................................. 19
MobiLink data synchronization ................................................................................... 21
How to close an UltraLite Java edition database connection ................................... 26
How to build and deploy UltraLiteJ applications ...................................................... 27
Code examples ............................................................................................................ 30

Tutorial: Building an Android application .................................................................. 41

Lesson 1: Setting up a new Android project ............................................................. 41
Lesson 2: Starting the MobiLink server ..................................................................... 43
Lesson 3: Running your Android application ............................................................ 44
Lesson 4: Testing your Android application and synchronizing .............................. 45
Cleaning up ................................................................................................................ 46

Tutorial: Building a BlackBerry application ............................................................. 49

Part 1: Creating a new BlackBerry application .......................................................... 49
Part 2: Using MobiLink to synchronize the BlackBerry application ....................... 64
Cleaning up ................................................................................................................ 71
Code listing for tutorial ................................................................................................. 72
<table>
<thead>
<tr>
<th>Class/Interface</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnSchema interface</td>
<td>79</td>
</tr>
<tr>
<td>ConfigFile interface</td>
<td>79</td>
</tr>
<tr>
<td>ConfigFileAndroid interface [Android]</td>
<td>85</td>
</tr>
<tr>
<td>ConfigNonPersistent interface [BlackBerry]</td>
<td>86</td>
</tr>
<tr>
<td>ConfigObjectStore interface [BlackBerry]</td>
<td>88</td>
</tr>
<tr>
<td>ConfigPersistent interface</td>
<td>89</td>
</tr>
<tr>
<td>Configuration interface</td>
<td>91</td>
</tr>
<tr>
<td>Connection interface</td>
<td>100</td>
</tr>
<tr>
<td>DatabaseInfo interface</td>
<td>102</td>
</tr>
<tr>
<td>DatabaseManager class</td>
<td>130</td>
</tr>
<tr>
<td>DecimalNumber interface</td>
<td>134</td>
</tr>
<tr>
<td>Domain interface</td>
<td>142</td>
</tr>
<tr>
<td>FileTransfer interface</td>
<td>145</td>
</tr>
<tr>
<td>FileTransferProgressData interface</td>
<td>155</td>
</tr>
<tr>
<td>FileTransferProgressListener interface</td>
<td>170</td>
</tr>
<tr>
<td>IndexSchema interface</td>
<td>172</td>
</tr>
<tr>
<td>PreparedStatement interface</td>
<td>173</td>
</tr>
<tr>
<td>ResultSet interface</td>
<td>175</td>
</tr>
<tr>
<td>ResultSetMetadata interface</td>
<td>191</td>
</tr>
<tr>
<td>SISListener interface [BlackBerry]</td>
<td>209</td>
</tr>
<tr>
<td>SISRquestHandler interface [BlackBerry]</td>
<td>215</td>
</tr>
<tr>
<td>SQLInfo interface [Android]</td>
<td>216</td>
</tr>
<tr>
<td>StreamHTTPParms interface</td>
<td>217</td>
</tr>
<tr>
<td>StreamHTTPSParms interface</td>
<td>218</td>
</tr>
<tr>
<td>SyncObserver interface</td>
<td>231</td>
</tr>
<tr>
<td>SyncObserver.States interface</td>
<td>236</td>
</tr>
<tr>
<td>SyncObserver.States interface</td>
<td>238</td>
</tr>
<tr>
<td>SyncParms class</td>
<td>242</td>
</tr>
<tr>
<td>SyncResult class</td>
<td>260</td>
</tr>
<tr>
<td>SyncResult.AuthStatusCode interface</td>
<td>269</td>
</tr>
<tr>
<td>TableSchema interface</td>
<td>271</td>
</tr>
<tr>
<td>ULjEvent interface [Android]</td>
<td>276</td>
</tr>
<tr>
<td>ULjException class</td>
<td>277</td>
</tr>
<tr>
<td>Unsigned64 class</td>
<td>279</td>
</tr>
<tr>
<td>UUIDValue interface</td>
<td>283</td>
</tr>
</tbody>
</table>
About this book

This book describes the UltraLiteJ programming interface. With UltraLiteJ, you can develop and deploy database applications to Android and BlackBerry smartphones.
System requirements and supported platforms

Development platforms
To develop UltraLiteJ applications, you must have the following:

- A Java IDE, such as Eclipse
- Java SE 1.6 or later

Target platforms
UltraLiteJ supports the following target platforms:

- Android smartphones
- BlackBerry smartphones running OS 4.2 or later
- Java SE 1.6 or later running on a computer or device

For information about UltraLite supported platforms, see http://www.sybase.com/detail?id=1002288.
UltraLiteJ application development

The UltraLiteJ API provides database functionality and synchronization to your Java applications. It is designed to work specifically with Android and BlackBerry smartphones, but is compatible with Java SE platforms. The API contains all the methods required to connect to an UltraLite Java edition database or an UltraLite database for Android, perform schema operations, and maintain data using SQL statements. Advanced operations, such as data encryption and synchronization, are also supported.

Note
When developing for Android smartphones, the UltraLiteJ API shares a common C++ code base with UltraLite for other platforms and its behavior is similar to that of other platforms. There are a few features available for accessing tables and rows without using SQL statements for which no API is provided on Android.

See also
- “UltraLite overview” [UltraLite - Database Management and Reference]
- “Benefits of UltraLite APIs for Windows Mobile” [UltraLite - Database Management and Reference]

Quick start guide to UltraLiteJ application development

When creating an UltraLiteJ application, you typically complete the following data management tasks in your application code:

1. Import the UltraLiteJ API package into your Java file(s).

The UltraLiteJ package name and location depends on the device you are developing applications for.

2. Create a new Configuration object to create or connect to a database.

Configuration objects define where the client database is located or where it should be created. They also specify the username and password required to connect to the database. Variations of a Configuration object are available for different devices and for non-persistent database stores.

3. Create a new Connection object.

Connection objects connect to a client database using the specifications defined in the Configuration object.

4. Create or modify the database schema using SQL statements, and use the PreparedStatement interface to query the database.

You can use SQL statements to create or update tables, indexes, foreign keys, and publications for your database.
PreparedStatement objects query the database associated with the Connection object. They accept supported SQL statements, which are passed as strings. You can use PreparedStatement objects to update the contents of the database.

5. Generate ResultSet objects.

ResultSet objects are created when the Connection object executes a PreparedStatement containing a SQL SELECT statement. You can use ResultSet objects to obtain rows of query results to view the table contents of the database.

See also
- “Android and BlackBerry setup considerations” on page 4
- “Configuration interface [UltraLiteJ]” on page 100
- “Connection interface [UltraLiteJ]” on page 102
- “UltraLite SQL statements” [UltraLite - Database Management and Reference]
- “PreparedStatement interface [UltraLiteJ]” on page 175
- “ResultSet interface [UltraLiteJ]” on page 191

Android and BlackBerry setup considerations

There are UltraLiteJ API considerations to make before developing applications for Android or BlackBerry smartphones.

JAR resource files

When setting up an application for the UltraLiteJ API, make sure that your project is correctly configured to use the appropriate UltraLiteJ16.jar or UltraLiteJNI16.jar file.

The UltraLiteJ API for Android is stored in the UltraLite\UltraLiteJ\Android\UltraLiteJNI16.jar file of your SQL Anywhere installation. You must configure your Android development project to include the UltraLiteJNI16.jar file in the classpath. For more information, see “Tutorial: Building an Android application” on page 41.

For Android development, use the following statement to import the UltraLiteJ package into your Java file:

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

The UltraLiteJ API for BlackBerry and Java SE can be found in the UltraLite\UltraLiteJ\ directory of your SQL Anywhere installation. There is a subdirectory and UltraLiteJ16.jar file for each target platform. You must configure your BlackBerry development project to include the UltraLiteJ16.jar file in the classpath. For more information, see “Tutorial: Building a BlackBerry application” on page 49.

For BlackBerry development, use the following statement to import the UltraLiteJ package into your Java file:

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

All coding samples and tutorials contained in this document assume that the above statement is specified and that you are familiar with developing Java applications in Eclipse.
Java SE applications

Java SE applications are not supported by the UltraLiteJNI.jar file. You must use the UltraLite\UltraLiteJ2SE\UltraLiteJ16.jar file.

UltraLite and UltraLite Java edition databases

On Android smartphones, UltraLiteJ provides an interface to the same UltraLite database management system that is provided for Windows Mobile, iPhone, and Windows. On BlackBerry smartphones, UltraLiteJ provides an interface to the UltraLite Java edition database management system. UltraLite Java edition has similar features to UltraLite, but they are not identical. UltraLite Java edition databases are not interchangeable with UltraLite databases.

Android smartphones only support UltraLite databases. They can be created using Sybase Central or UltraLite command line utilities. For more information about creating an UltraLite database, see “UltraLite database creation approaches” [UltraLite - Database Management and Reference].

BlackBerry smartphones only support UltraLite Java edition databases. For more information about creating or using an UltraLite Java edition database, see “UltraLite and UltraLite Java edition database creation and connection approaches” on page 5.

See also

- “Benefits of UltraLite APIs for Windows Mobile” [UltraLite - Database Management and Reference]
- “UltraLite, UltraLite Java edition, and SQL Anywhere feature comparisons” [UltraLite - Database Management and Reference]
- “UltraLite and UltraLite Java edition database limitations” [UltraLite - Database Management and Reference]

UltraLite and UltraLite Java edition database creation and connection approaches

Java applications must connect to a database before operations can be performed on the data. This section explains how to create or connect to an UltraLite or UltraLite Java edition database with a specified password using the UltraLiteJ API.
To create an UltraLite database without using the UltraLiteJ API, you can use Sybase Central or UltraLite command line utilities. See “UltraLite database creation approaches” [UltraLite - Database Management and Reference].

To create an UltraLite Java edition database without using the UltraLiteJ API, you can perform one of the following tasks:

- Create the database using the uljload utility. See “UltraLite Java Edition Database Load utility (uljload)” [UltraLite - Database Management and Reference].

- Use the ulunload and uljload utilities to convert an UltraLite database. See “UltraLite Database Unload utility (ulunload)” [UltraLite - Database Management and Reference].

- Deploy a Java SE application to a BlackBerry smartphone by copying the UltraLite Java database to an SD card or transferring it from MobiLink using the file transfer mechanism. See “MobiLink file transfers” [UltraLite - Database Management and Reference].

For more information about the differences between an UltraLite and an UltraLite Java edition database, see “UltraLite and UltraLite Java edition databases” on page 5.

A Configuration object is used to configure a database store. Several implementations of a Configuration object are provided. A unique implementation exists for every type of database store supported by the UltraLiteJ API. Each implementation provides a set of methods used to configure the database store.

The following table lists the available Configuration object implementations for the supported database stores:

<table>
<thead>
<tr>
<th>Store type</th>
<th>UltraLiteJ API support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android file system</td>
<td>See “ConfigFileAndroid interface [Android] [UltraLiteJ]”.</td>
</tr>
<tr>
<td>RIM object (BlackBerry)</td>
<td>See “ConfigObjectStore interface [BlackBerry] [UltraLiteJ]”.</td>
</tr>
<tr>
<td>Java SE file system</td>
<td>See “ConfigFile interface [UltraLiteJ]”.</td>
</tr>
<tr>
<td>Non-persistent (in memory)</td>
<td>See “ConfigNonPersistent interface [BlackBerry] [UltraLiteJ]”.</td>
</tr>
</tbody>
</table>

After creating and configuring a Configuration object, you use a Connection object to create or connect to the database. Connection objects can also be used to perform the following operations:

- **Transactions**  A transaction is the set of operations between commits or rollbacks. For persistent database stores, a commit makes permanent any changes since the last commit or rollback. A rollback returns the database to the state it was in when the last commit was invoked.

  Each transaction and row-level operation in UltraLite is atomic. An insert involving multiple columns either inserts data to all the columns or to none of the columns.

  Transactions must be committed to the database using the commit method of the Connection object.
● **Prepared SQL statements**  Methods are provided by the PreparedStatement interface to handle SQL statements. A PreparedStatement can be created using the prepareStatement method of the Connection object.

● **Synchronizations**  A set of objects governing MobiLink synchronization is accessed from the Connection object.

**See also**
- “Java SE example: Creating a database” on page 31
- “Connection interface [UltraLiteJ]” on page 102
- “DatabaseManager class [UltraLiteJ]” on page 134

### Creating or connecting to a database

Use the UltraLiteJ API with your Java application to create or connect to a database.

**Prerequisites**

An existing Java application for an Android or a BlackBerry smartphone that implements the UltraLiteJ API.

**Task**

1. Create a new Configuration object that references the database name and is appropriate for your platform.

   In the following examples, **config** is the Configuration object name and **DBname** is the database name.

   For Android smartphones:

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

   For BlackBerry smartphones:

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

   For Java SE platforms:

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

   For any platform, you can create a non-persistent database Configuration object:

   ```java
   ConfigNonPersistent config =
   DatabaseManager.createConfigurationNonPersistent("DBname.ulj");
   ```
2. Set database properties using methods of the Configuration object.

For example, you can set a new database password using the setPassword method:

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

For Android smartphones, you can use the setCreationString and setConnectionString methods to set additional creation and connection parameters, respectively.

For more information about creation and connection parameters, see:
- “UltraLite creation parameters” [UltraLite - Database Management and Reference]
- “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97
- “UltraLite connection parameters” [UltraLite - Database Management and Reference]
- “ConfigPersistent.setConnectionString method [Android] [UltraLiteJ]” on page 96

3. Create a Connection object to create or connect to the database:

For example, the following code creates a new database:

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

The DatabaseManager.createDatabase method creates the database and returns a connection to it.

In the above example, the following code is used to connect to an existing database:

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

The connect method finalizes the database connection process. If the database does not exist, an error is thrown.

**Results**

You can execute SQL statements from your Java application to create the tables and indexes in your database but you cannot change certain database creation parameters, such as the database name, password, or page size.

**See also**
- “DatabaseManager class [UltraLiteJ]” on page 134
- “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]”
- “UltraLite connection parameters” [UltraLite - Database Management and Reference]
- “ConfigPersistent.setConnectionString method [Android] [UltraLiteJ]”

**Quick start guide to schema operations and data management**

Create, update, or retrieve tables, indexes, foreign keys, publications, and rows in your database using SQL statements and queries.
When performing schema operations to manage data, you typically perform the following tasks in your application code:

1. Perform schema operations.
   Manage and modify the schema by using SQL statements such as CREATE TABLE or CREATE INDEX on the database connection.

2. Manage row operations.
   Manage data in tables using SQL statements such as INSERT, UPDATE, or DELETE on the database connection.

3. Retrieve row data in a result set.
   Retrieve a result set using the SELECT statement, and then traverse the row data using result set navigation methods, such as previous and next.

Example: Managing database operations on an Android smartphone

The following example illustrates a sample class in an Android application that uses the UltraLiteJ API to perform the following operations:

- Create a table in a database
- Insert new rows into the table
- Update a row in the table
- Delete a row from the table
- Commit changes to the database
- Select all rows from the table by creating a result set
- Traverse the result set to view the rows in the database

In addition to performing these operations, the class contains a method named PrintText that is used to output successful operations to the log (see the LogCat tab in Eclipse), and a HandleError method that is used to report errors that may occur while performing UltraLiteJ API operations.

```java
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);
        } catch (ULjException e) {
            HandleError(e);
        }
    }
}
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();
    } 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++) {
        ExecuteSQLStatement(stmt[i]);
    }
    InitStatements();
    AddDepartment(101, "Electronics");
    AddDepartment(105, "Sales");
    AddDepartment(109, "Accounting");
    UpdateDepartment(101, 102);
    DeleteDepartment(102);
    Commit();
}

public void HandleError(Exception e) {
    PrintText("Error: " + e.getMessage());
}

public void ExecuteSQLStatement(String stmt) {
    try {
        _conn.createStatement(stmt);
        PrintText("Successfully executed: " + stmt);
    } catch (SQLException e) {
        HandleError(e);
    }
}
_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();
}

See also

- “DatabaseManager class [UltraLiteJ]” on page 134
- “DatabaseManager.createConfigurationFileAndroid method [UltraLiteJ]” on page 137
- “Connection.prepareStatement method [UltraLiteJ]” on page 115
- “PreparedStatement interface [UltraLiteJ]” on page 175
- “PreparedStatement.set method [UltraLiteJ]” on page 183
- “PreparedStatement.execute method [UltraLiteJ]” on page 177
- “PreparedStatement.executeQuery method [UltraLiteJ]” on page 177
- “Connection.commit method [UltraLiteJ]” on page 107
- “Connection.rollback method [UltraLiteJ]” on page 117
- “ResultSet interface [UltraLiteJ]” on page 191
- “ResultSet.afterLast method [Android] [UltraLiteJ]” on page 194
- “ResultSet.beforeFirst method [Android] [UltraLiteJ]” on page 194
- “ResultSet.first method [Android] [UltraLiteJ]” on page 194
- “ResultSet.last method [Android] [UltraLiteJ]” on page 208
- “ResultSet.next method [UltraLiteJ]” on page 208
- “ResultSet.previous method [UltraLiteJ]” on page 208
- “ResultSet.relative method [Android] [UltraLiteJ]” on page 209
- “UltraLite SQL statements” [UltraLite - Database Management and Reference]

**Schema operations**

Perform schema operations by following these general tasks:

1. Construct a SQL statement in a String variable.
2. Create a PreparedStatement object by passing the String variable to the Connection.prepareStatement method.
3. Call the PreparedStatement.execute method to perform the operation on the database.

4. Close the PreparedStatement object to free resources.

Example

The code referenced in this example is part of a complete sample that illustrates how to perform basic schema and data management operations using the UltraLiteJ API. See “Example: Managing database operations on an Android smartphone”.

Individual CREATE TABLE and CREATE INDEX statements are constructed and passed to a custom method named ExecuteSQLStatement to perform all necessary schema operations:

```java
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]);
}
```

The ExecuteSQLStatement method consists of the following code:

```java
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);
    }
}
```

See also

- “Connection.prepareStatement method [UltraLiteJ]” on page 115
- “PreparedStatement interface [UltraLiteJ]” on page 175
- “PreparedStatement.set method [UltraLiteJ]” on page 183
- “PreparedStatement.execute method [UltraLiteJ]” on page 177

Row operation management

Manage row operations by performing the following general tasks:
1. Construct a SQL statement in a String variable.

2. Create a PreparedStatement object by passing the String variable to the Connection.prepareStatement method.

3. Set any host variables, indicated by the ? character, by using the PreparedStatement.set method.

   Each host variable can be referenced in accordance to its ordinal position in the statement. For example, the first ? is referenced as 1, and the second as 2. The set method, illustrated in the example below, allows you to reference the ordinal position of the variable and specify a new value.

4. Call the PreparedStatement.executeUpdate method to perform the operation on the database.

5. Commit the changes to the database by calling the Connection.commit method to make the changes permanent; otherwise, call the Connection.rollback method.

   Transactions must be explicitly committed or rolled back by using the methods supported by the Connection interface.

6. Close the PreparedStatement object to free resources.

Example

The code referenced in this example is part of a complete sample that illustrates how to perform basic schema and data management operations using the UltraLiteJ API. See “Example: Managing database operations on an Android smartphone”.

Global PreparedStatement objects are defined and instantiated by an InitStatements method call. Data insert, update, and delete operations are illustrated in the custom AddDepartment, UpdateDepartment, and DeleteDepartment methods, respectively. The Commit method illustrates how to make the row operations permanent. The FiniStatements method closes the global PreparedStatement objects and frees resources.

```java
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) {
    HandleException(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) {
    HandleException(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) {
    HandleException(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) {
    HandleException(e);
}

public void Commit() {
    try {
        _conn.commit();
    } catch (ULjException e1) {
        HandleException(e1);
    }
}
String concatenation is recommended over host variable usage when constructing SQL statements that only need to be executed once.

For example, the following code, which uses String concatenation to construct SQL statements, can be used to replace the DeleteDepartment method:

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

See also

- “Connection.prepareStatement method [UltraLiteJ]” on page 115
- “PreparedStatement interface [UltraLiteJ]” on page 175
- “PreparedStatement.set method [UltraLiteJ]” on page 183
- “PreparedStatement.execute method [UltraLiteJ]” on page 177

**Row data retrieval**

Retrieve row data from a table by performing the following general tasks:

1. Construct a SELECT SQL statement in a String variable.
2. Create a PreparedStatement object by passing the String variable to the Connection.prepareStatement method.
3. Call the PreparedStatement.executeQuery method to assign the query results to a ResultSet object.
4. Traverse through the ResultSet object using the navigational methods to retrieve the row data.

The following navigational methods can be used to traverse a result set:

- `afterLast` Position immediately after the last row.
- `beforeFirst` Position immediately before the first row.
- `first` Move to the first row.
- `last` Move to the last row.
- `next` Move to the next row.
• **previous**  Move to the previous row.

• **relative(offset)**1  Move a specified number of rows relative to the current row, as specified by the signed offset value. Positive offset values move forward in the result set, relative to the current pointer position in the result set. Negative offset values move backward in the result set. An offset value of zero does not move the current location, but allows you to repopulate the row buffer.

1 This method is not supported for UltraLite Java edition databases.

5. Close the ResultSet and PreparedStatement objects to free resources.

**Example**

The code referenced in this example is part of a complete sample that illustrates how to perform basic schema and data management operations using the UltraLiteJ API. See “Example: Managing database operations on an Android smartphone”.

A result set is retrieved in the custom SelectDepartmentRows method:

```java
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;
}
```

Row data is retrieved by traversing through a result set using the **next** navigational method:

```java
_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);
    }
}
```
See also

- “PreparedStatement.executeQuery method [UltraLiteJ]” on page 177
- “ResultSet interface [UltraLiteJ]” on page 191
- “ResultSet.afterLast method [Android] [UltraLiteJ]” on page 194
- “ResultSet.beforeFirst method [Android] [UltraLiteJ]” on page 194
- “ResultSet.first method [Android] [UltraLiteJ]” on page 194
- “ResultSet.last method [Android] [UltraLiteJ]” on page 208
- “ResultSet.next method [UltraLiteJ]” on page 208
- “ResultSet.previous method [UltraLiteJ]” on page 208
- “ResultSet.relative method [Android] [UltraLiteJ]” on page 209

Schema information access

You can programmatically retrieve database schema descriptions. These descriptions are known as schema information and are accessible using system tables and the UltraLiteJ API schema interfaces.

Accessing schema information using system tables

Database schema information is stored in UltraLite or UltraLite Java edition system tables. You can access this information by executing a regular SQL query to select the desired information from the appropriate system table, and then accessing the result set.

Accessing schema information using schema interfaces

Some schema information can be accessed using schema interfaces instead of system tables. The UltraLiteJ API contains the following schema interfaces:

- **TableSchema**  
  Returns information about the column and index configurations.

- **IndexSchema**  
  Returns information about the columns in the index. IndexSchema objects can be retrieved from TableSchema objects.

- **ColumnSchema**  
  Returns information about the columns in the table. ColumnSchema objects can be retrieved from TableSchema objects.

See also

- “Row data retrieval” on page 17
- “UltraLite system tables” [*UltraLite - Database Management and Reference*]
- “UltraLite Java edition system tables” [*UltraLite - Database Management and Reference*]
- “TableSchema interface [UltraLiteJ]” on page 271
- “IndexSchema interface [UltraLiteJ]” on page 173
- “ColumnSchema interface [UltraLiteJ]” on page 79

Error handling

You can use the ULjException and SQLCode classes to handle errors. Most UltraLite methods throw ULjException errors. You can use the ULjException.getErrorCode method to retrieve the SQLCode value.
assigned to the error. You can use the ULjException.toString method to obtain a descriptive text of the error. SQLCode errors are negative numbers indicating the error type, and can be referenced using constants such as ULjException.SQLE_INDEX_NOT_FOUND.

Example

The following example illustrates a Java class that uses the ULjException class to handle an error that may occur when connecting to an UltraLite Java edition database:

```java
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 static Connection _conn;
    private static DataAccess _da;
}

See also

- “ULjException class [UltraLiteJ]” on page 277
- “ULjException.getErrorCode method [UltraLiteJ]” on page 278
- “SQL Anywhere error messages sorted by SQLCODE” [Error Messages]
MobiLink data synchronization

The UltraLite Java edition management system contains a built-in change tracking system that allows database changes to be synchronized.

Data synchronization can be performed using HTTP or HTTPS network protocols. HTTPS synchronization provides secure encryption to the MobiLink server.

To synchronize data, your application must perform the following steps:

1. Instantiate a syncParms object, which contains information about the consolidated database (name of the server, port number), the name of the database to be synchronized, and the definition of the tables to be synchronized.

2. Call the synchronize method from the connection object with the syncParms object to perform the synchronization.

The data to be synchronized can be defined at the table level. You cannot configure synchronization for portions of a table.

See also

- “SyncParms class [UltraLiteJ]” on page 242
- “SyncResult class [UltraLiteJ]” on page 260
- “Tutorial: Building an Android application” on page 41

Example

The following example demonstrates data synchronization with an UltraLiteJ application and can be found in %SQLANYSAMP16%\UltraLiteJ\J2SE\Sync.java:

```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" +
```

```java
    Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 21
```
''', 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();
}
}

To start the MobiLink server with CustDB as the consolidated database, run start_ml.bat from the %SQLANYSAMP16%\J2SE\UltraLiteJ directory.

For Android smartphones, a tutorial based on using the CustDB as a consolidated database is available.

**Data synchronization on a BlackBerry smartphone**

In a BlackBerry environment, data is always encrypted between the device and the BlackBerry Enterprise Server (BES). HTTPS is useful when encryption is required between the BES and the MobiLink server, or when the device is not managed by a BES.

**Concurrent synchronization**

Normally only one thread is allowed in the UltraLite runtime at a time. An exception to this rule occurs during synchronization. While one connection is performing a synchronization operation, other connections can access the UltraLite runtime; however, no other thread can call the synchronize method for the database being synchronized while a synchronize operation is taking place.

As UltraLite operates at isolation level zero, a connection can access downloaded rows during the synchronization (that is, before they are committed) that may later vanish if the synchronization fails. If, during a synchronization, a connection modifies a row that the synchronization then attempts to change, the synchronization fails. During a synchronization, if a connection attempts to modify a row that the synchronization has changed, the attempt to modify fails.
Network protocol options for UltraLiteJ synchronization streams

When synchronizing with a MobiLink server, you must set the network protocol in your application. Each database synchronizes over a network protocol. Two network protocols are available for UltraLiteJ—HTTP and HTTPS.

For the network protocol you set, you can choose from a set of corresponding protocol options to ensure that the UltraLiteJ application can locate and communicate with the MobiLink server. Network protocol options provide information such as addressing information (host and port) and protocol-specific information.

Setting up an HTTP network protocol

An HTTP network protocol is set with the StreamHTTPParms interface in the UltraLiteJ API. Use the interface methods to specify the network protocol options defined on the MobiLink server.

Setting up an HTTPS network protocol

An HTTPS network protocol is set with the StreamHTTPSParms interface in the UltraLiteJ API. Use the interface methods to specify the network protocol options defined on the MobiLink server.

See also

- “MobiLink client network protocol options” [MobiLink - Client Administration]
- “StreamHTTPParms interface [UltraLiteJ]” on page 218
- “StreamHTTPSParms interface [UltraLiteJ]” on page 231

CustDB application synchronization on a BlackBerry smartphone

CustDB (customer database) is a sample database and UltraLite application installed with SQL Anywhere. The CustDB database is a simple sales order database.

Finding and deploying the application

UltraLiteJ includes a sample BlackBerry application that is based on the CustDB database. The application is named CustDB; the source code and related files are found in the %SQLANYSAMP16%UltraLiteJBlackBerryCustDB directory. The CustDB directory includes the project files that can be opened with the BlackBerry JDE. Additional information about the CustDB application can be found in readme.txt.

After the CustDB application has been build, deploy CustDB16.cod and the required files to a simulator or device.

Files related to CustDB application

The following files in the CustDB project contain the database access code:
- **CustDB.java**  This file contains all the basic database access methods. These methods include creating and connecting to databases, and inserting, deleting, and updating orders. This file contains many of the database calls to communicate to the back-end server.

- **SchemaCreator.java**  This file contains the code to create tables on the device using UltraLiteJ.

**Using the CustDB application**

Run `%SQLANYSAMP16%\UltraLite\BlackBerry\CustDB\mobilink.bat` to start a local MobiLink server.

When initially started, the CustDB program collects information to use to interact with the server where the CustDB database is hosted. You specify the Employee ID to use for queries (50 is recommended), the host name or IP address of the server where the data is hosted, and a port number for the connection to the server.

Once these values are specified and the settings are saved (by choosing **Menu » Save**), the application synchronizes with the specified server. The application only downloads orders from the server that match the Employee ID corresponding to the specified employee number (50). Only orders that are still open are selected (orders can be in one of three states: Open, Approved, or Denied).

Each order is displayed on the screen with the customer name, product ordered, quantity, price, and discount. The screen also shows the current status of the order and any notes pertaining to that order.
On this screen, you can add notes to the order, or change the status of the order (to either Approved or Denied). You can navigate through the orders using the Next and Previous buttons.

The CustDB program also allows you to add new orders into the database. To add a new order, click Menu » New Order.

Enter quantity and discount values.

Before exiting the application, click Synchronize from the main menu to synchronize your changes and new orders with the consolidated database.
How to close an UltraLite Java edition database connection

An UltraLite Java edition database closes when all the concurrent connections have been released. You can release all the connections using the DatabaseManager.release method.

**Note**

Use the Connection.release method to release the current connection.

If a BlackBerry application crashes before it can release its active database connections, a ULjException object with the SQL_FILE_IN_USE error code may be thrown when the application attempts to reconnect to the database. In this scenario, you must reboot the smartphone so that the application can reconnect.

To avoid a reboot, a catch-all exception handler should be used to call the emergencyShutdown method so that the connections are released.

For example, the following catch-all exception handler can be used in a BlackBerry application to call the emergencyShutdown method when an unrecoverable error occurs:

```java
try {
    // top level application code
    // release all connections in a normal termination
} catch( Exception e ) {
    conn.emergencyShutdown();
    throw e;
}
```
This example assumes that the conn object represents an active connection to an UltraLite Java edition database.

See also

- “DatabaseManager.release method [UltraLiteJ]” on page 141
- “Connection.release method [UltraLiteJ]” on page 116

How to build and deploy UltraLiteJ applications

UltraLiteJ applications can be deployed to Android and BlackBerry smartphones. For an UltraLiteJ application to run successfully, you must deploy the UltraLiteJ API with your distribution.

See also

- “UltraLite application build and deployment specifications” [UltraLite - Database Management and Reference]

Deploying an UltraLiteJ application for Android

Specify appropriate creation parameters, connection parameters, synchronization parameters, protocol options, method calls, and deployment files to ensure that your UltraLiteJ application runs successfully on Android smartphones.

Prerequisites

There are no prerequisites for this task.

Task

1. Add the following files to your Android project:

   - %SQLANY16%\UltraLite\UltraLiteJ\Android\UltraLiteJNI16.jar
   - %SQLANY16%\UltraLite\UltraLiteJ\Android\ARM\libultralitej16.so

2. Specify the following parameters:

   - When using obfuscation, set the creation parameter obfuscate=1 while creating the database.
   - When using AES encryption, set the connection parameter DBKEY=encryption-key while creating or connecting to the database.

   To set these parameters, use the setCreationString and setConnectionString methods.

3. Set the appropriate parameter settings when using synchronization in your UltraLite application:
### Synchronization type

<table>
<thead>
<tr>
<th>Parameter settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the Stream synchronization parameter to &quot;tcpip&quot;.</td>
</tr>
<tr>
<td>Set the Stream synchronization parameter to &quot;http&quot;.</td>
</tr>
<tr>
<td>Set the Stream synchronization parameter to &quot;tls&quot;.</td>
</tr>
<tr>
<td>Set the Stream synchronization parameter to &quot;https&quot;.</td>
</tr>
</tbody>
</table>

4. When using RSA end-to-end encryption (RSA E2EE), set the protocol option `e2ee_public_key=key-file`.

5. When using ZLIB compression, set the protocol option `compression=zlib`.


7. When using RSA TLS, RSA HTTPS, or RSA E2EE, deploy `%SQLANY16%\UltraLite\UltraLiteJ\Android\ARM\libmlcrsa16.so`.

### Results

The UltraLiteJ application runs successfully on the Android device that it is deployed to.

### Next

Deploy an UltraLite database to the Android mobile device that the application was deployed to, or create a new database with the deployed application.

### See also

- “UltraLite application build and deployment specifications” [UltraLite - Database Management and Reference]
- “UltraLite and UltraLite Java edition database deployment techniques” [UltraLite - Database Management and Reference]
- “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97
- “ConfigPersistent.setConnectionString method [Android] [UltraLiteJ]” on page 96

### Deploying an UltraLiteJ application for BlackBerry

Use method calls to specify appropriate creation parameters, connection parameters, synchronization parameters, and protocol options, and deploy the appropriate files to ensure that your UltraLiteJ application runs successfully on BlackBerry smartphones.

### Prerequisites

There are no prerequisites for this task.
Task

1. Call the following methods during database creation and connection:
   - When using obfuscation, `ConfigPersistent.enableObfuscation`.
   - When using AES encryption, `ConfigPersistent.EnableAesDBEncryption` and `ConfigPersistent.setEncryptionKey`.

2. When using synchronization, specify the synchronization stream by passing the `SyncParms.HTTP_STREAM` or `SyncParms.HTTPS_STREAM` constant to the `Connection.createSyncParms` method, and then use the `StreamHTTPParms` or `StreamHTTPSParms` interface.

3. Set stream compression and encryption options through the interface you selected in the previous step:
   - For ZLIB compression, call the `setZlibCompression` method.
   - For RSA E2EE encryption, call the `setE2eePublicKey` method.

4. Deploy the following files, located in the `%SQLANY16%\UltraLite\UltraLiteJ\BlackBerry4.2` directory:
   - `UltraLiteJ16.cod`
   - `UltraLiteJ16.jad`

   `UltraLiteJ16.jad` is required for over-the-air (OTA) deployment only. Alternatively, you can create your own `.jad` file that deploys UltraLiteJ with your application.

5. When using RSA E2EE encryption, transfer the DER encoded file using the `FileTransfer` interface, or store the file on an SD card.

Results

The UltraLiteJ application runs successfully on the BlackBerry smartphone that it is deployed to.

Next

Deploy an UltraLite Java edition database to the BlackBerry smartphone that the application was deployed to, or create a new database with the deployed application.

See also

- “UltraLite application build and deployment specifications” [UltraLite - Database Management and Reference]
- “UltraLite and UltraLite Java edition database deployment techniques” [UltraLite - Database Management and Reference]
- “ConfigPersistent interface [UltraLiteJ]” on page 91
- “StreamHTTPParms interface [UltraLiteJ]” on page 218
- “StreamHTTPSParms interface [UltraLiteJ]” on page 231
- “FileTransfer interface [UltraLiteJ]” on page 155
Code examples

This section illustrates Java code samples that utilize the UltraLiteJ API. The examples use a demo class that displays messages and handles `ULjException` objects for debugging purposes.

All coding examples can be found in the `%SQLANYSAMP16\UltraLiteJ\J2SE` directory. It is recommended that you create backup copies of the original source code before modifying the file contents.

Although these code examples are written for the Windows desktop environment, the concepts presented apply to all UltraLiteJ platforms (unless otherwise noted).

To run these examples, set your `JAVA_HOME` environment variable to an installed version of the JDK 1.6. You can then run `rundemo.cmd` with the name of the example.

The demo class found in `%SQLANYSAMP16\UltraLiteJ\J2SE\Demo.java` is used by all the examples contained in this section of the documentation.

For example, the following command runs the `CreateDb` sample:

```
    rundemo CreateDb
```

The command produces the following output:

```
    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
```

The output is saved in a file named `demos.out`.

BlackBerry examples

The following demos are available for BlackBerry developers in the `%SQLANYSAMP16\UltraLiteJ\BlackBerry` directory:

- The `BinaryStoreAsFile` demo, which illustrates the use of the ability to associate external files as part of a database.

- The `CustDB` demo, which shows a mobile customer order application.

- The `BlackBerryEncryption` demo, which demonstrates advance concepts for ultra-secure UltraLiteJ database.
● The HelloBlackBerry demo.

A tutorial based on the HelloBlackBerry demo is available.

**Android example**

A sample Eclipse project that uses the CustDB sample database is available in the %SQLANYSAMP16%\UltraLite\Android\CustDB directory. The source code can be found in %SQLANYSAMP16%\UltraLiteJ\Android\CustDB\src\com\sybase\custdb.

A tutorial based on the example is available.

**See also**

- “Tutorial: Building a BlackBerry application” on page 49
- “Tutorial: Building an Android application” on page 41

**Java SE example: Creating a database**

Run the Java SE sample to understand how to create a file system database store in a Java SE Java environment.

**Prerequisites**

There are no prerequisites for this task.

**Context and remarks**

The Configuration object is used to create the database. Once created, a Connection object is returned. To create tables, the schemaUpdateBegin method is invoked to start changes to the underlying schema and the schemaUpdateComplete method completes changing the schema.

**Task**

1. Change to the following directory: %SQLANYSAMP16%\UltraLiteJ\J2SE.

2. Run the CreateDb example:

   rundemo CreateDb

**Results**

The application runs successfully. An UltraLite Java edition file system database store is created.

**Java SE example: Inserting rows**

Run the Java SE sample to understand how to insert rows into an UltraLite Java edition database.

**Prerequisites**

There are no prerequisites for this task.
Task

1. Change to the following directory: `%SQLANYSAMP16%\UltraLiteJ\J2SE`.

2. Run the CreateDb example:
   
   `rundemo CreateDb`

3. Run the following command (the command is case sensitive):
   
   `rundemo LoadDb`

Results

The application runs successfully. Rows are inserted in the database.

Inserted data is persisted in the database only when the commit method is called from the Connection object.

When a row is inserted, but not yet committed, it is visible to other connections. This introduces the potential for a connection to retrieve row data that has not actually been committed.

See also

- “Java SE example: Creating a database” on page 31

Java SE example: Reading a table

Run the Java SE sample to understand how to read a table.

Prerequisites

There are no prerequisites for this task.

Context and remarks

In this example, a PreparedStatement object is obtained from a connection, and a ResultSet object is obtained from the PreparedStatement object. The next method on the ResultSet returns true each time a subsequent row can be obtained. Values for the columns in the current row can then be obtained from the ResultSet object.

Task

1. Change to the following directory: `%SQLANYSAMP16%\UltraLiteJ\J2SE`.

2. Run the CreateDb example:
   
   `rundemo CreateDb`

3. Run the LoadDb example:
   
   `rundemo LoadDb`
4. Run the following command (the command is case sensitive):

   rundemo ReadSeq

Results

The application runs successfully. When a ResultSet is created, it is positioned before the first row of the
result set. The ReadSeq application reads and displays result sets sequentially.

See also

- “Java SE example: Creating a database” on page 31
- “Java SE example: Inserting rows” on page 31

Java SE example: Inner join operations

Run the Java SE sample to understand how to perform an inner join operation.

Prerequisites

There are no prerequisites for this task.

Context and remarks

Every employee has corresponding department information. The join operation associates data from the
employee table with corresponding data from the department table. The association is made with the
department number in the employee table to locate the related information in the department table.

Task

1. Change to the following directory: %SQLANYSAMP16%\UltraLiteJ2SE.

2. Run the CreateDb example:

   rundemo CreateDb

3. Run the LoadDb example:

   rundemo LoadDb

4. Run the following command (the command is case sensitive):

   rundemo ReadInnerJoin

Results

The applications run successfully. The Employee table is read and each row is joined to the corresponding
row in the Department table.
See also

- “Java SE example: Creating a database” on page 31
- “Java SE example: Inserting rows” on page 31

Java SE example: Creating a sales database

Run the Java SE sample to understand how to create a database.

Prerequisites

There are no prerequisites for this task.

Task

1. Change to the following directory: `%SQLANYSAMP16%\UltraLiteJ\J2SE`.
2. Run the following command (the command is case sensitive):
   
   rundemo CreateSales

Results

The sample application runs successfully. A sales-oriented database is created.

Java SE example: Aggregation and grouping

Run the Java SE sample to understand how to handle the aggregation and grouping of results.

Prerequisites

There are no prerequisites for this task.

Task

1. Change to the following directory: `%SQLANYSAMP16%\UltraLiteJ\J2SE`.
2. Run the CreateSales example:
   
   rundemo CreateSales
3. Run the following command (the command is case sensitive):
   
   rundemo SalesReport

Results

The sample applications run successfully. A sales report is generated.
Java SE example: Retrieving rows in an alternative order

Run the Java SE sample to understand how to process rows in an alternate order.

Prerequisites

There are no prerequisites for this task.

Task

1. Change to the following directory: %SQLANYSAMP16%\UltraLite\J2SE.

2. Run the CreateSales example:

   rundemo CreateSales

3. Run the following command (the command is case sensitive):

   rundemo SortTransactions

Results

The sample applications run successfully. Table columns are ordered by product number when they are selected.

See also

- “Java SE example: Creating a sales database” on page 34

Java SE example: Modifying table definitions

Run the Java SE sample to understand how UltraLiteJ can change table definitions.

Prerequisites

There are no prerequisites for this task.

Context and remarks

An Invoice table is modified to expand a column length from 50 characters to 100 characters.

Task

1. Change to the following directory: %SQLANYSAMP16%\UltraLite\J2SE.

2. Run the CreateSales example:
3. Run the following command (the command is case sensitive):

   ```
rundemo Reorg
   ```

**Results**

The sample applications run successfully. The name column of the Invoice table is modified.

**See also**

- “Java SE example: Creating a sales database” on page 34

---

## Java SE example: Encrypting data

Run the Java SE sample to understand how to encrypt data in an UltraLite Java edition database and incur a performance penalty to decrypt the data.

**Prerequisites**

There are no prerequisites for this task.

**Context and remarks**

This example is for Java SE. For a complete BlackBerry encryption sample, see `%SQLANYSAMP16%\UltraLiteJ\BlackBerryEncryption`.

**Task**

1. Change to the following directory: `%SQLANYSAMP16%\UltraLiteJ\J2SE`.

2. Run the following command (the command is case sensitive):

   ```
rundemo Encrypted
   ```

**Results**

The sample application runs successfully. Encrypted data is inserted in the UltraLite Java edition database, and is decrypted when selecting the data.

---

## Java SE example: Displaying database schema information

Run the Java SE sample to gain an understanding of how UltraLite Java edition database system tables are traversed to examine the schema information.

**Prerequisites**

There are no prerequisites for this task.
Task

1. Change to the following directory: %SQLANYSAMPLE%\UltraLiteJ\J2SE.

2. Run the CreateSales example:
   
rundemo CreateSales

3. Run the following command (the command is case sensitive):
   
rundemo DumpSchema

Results

The data for each row of the tables also appears. The following output is displayed:

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 = 0x0 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 = 0x0 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-
<table>
<thead>
<tr>
<th>INDEX, PERSISTENT, PRIMARY-INDEX</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>key[0]: name = &quot;table_id&quot; flags = 0x1, FORWARD</td>
<td></td>
</tr>
<tr>
<td>key[1]: name = &quot;index_id&quot; flags = 0x1, FORWARD</td>
<td></td>
</tr>
</tbody>
</table>

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

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)

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

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

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)

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 = "index_id" flags = 0x1, FORWARD
- key[2]: name = "order" flags = 0x1, FORWARD

Table 1 name = "sysulj" id = 1 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)
See also
- “Java SE example: Creating a sales database” on page 34

Java SE example: Synchronizing a database

Run the Java SE sample that connects to a MobiLink server to understand how to synchronize a client database with a consolidated SQL Anywhere database.

Prerequisites
There are no prerequisites for this task.

Task

1. Change to the following directory: %SQLANYSAMP16%\UltraLiteJ\J2SE.

2. Run the CreateSales example:
   
   rundemo CreateSales

3. Run the following command to start the MobiLink server:
   
   start_ml

4. Run the following command (the command is case sensitive):
   
   rundemo Sync

Results

The sample application runs successfully. The client database synchronizes data with the consolidated database.

Next

Shut down the MobiLink server.

See also
- “Java SE example: Creating a sales database” on page 34
- “MobiLink server shutdown” [MobiLink - Server Administration]
Tutorial: Building an Android application

This tutorial guides you through the development of an application for Android smartphones using the UltraLiteJ API and the Eclipse environment. In this tutorial, you run the application on a Windows simulator.

The Android application used in this tutorial is located in the %SQLANYSAMP16%\UltraLiteJ\Android\CustDB directory. The application code, located in the src\com\sybase\custdb directory, references the UltraLiteJ API to perform the following tasks:

- UltraLite remote database creation.
- SQL operations on the database.
- Data synchronization with the SQL Anywhere CustDB sample database using MobiLink.

The res\menu and res\layout directories illustrate how to create Android menu items and interfaces. You can view these files through Eclipse when you create the new Android project.

The AndroidManifest.xml project file was modified so that the Android application can access the network, which is required for data synchronization. The following permission statement was added:

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

Required software

- Eclipse 3.5.2 or later
- Android SDK Starter Package
- Android Development Tools (ADT) Plug-in for Eclipse 1.1 or later
- SQL Anywhere 16 samples
- UltraLiteJ API

Competencies and experience

- Familiarity with Java
- Familiarity with Eclipse

See also

- “UltraLiteJ API reference” on page 79

Lesson 1: Setting up a new Android project

In this lesson, you create a new Android project through the Eclipse Integrated Development Environment.
Prerequisites

This lesson assumes that you have installed the required software. See “Tutorial: Building an Android application” on page 41.

Context and remarks

This tutorial assumes that you are familiar with Java and Eclipse.

Task

1. Copy the Android libraries to your Android CustDB sample directory.

   Open a command prompt, change to the %SQLANYSAMP16%\UltraLiteJ\Android\CustDB\ directory, and then run the following command:

   setup.bat

   The UltraLiteJNI16.jar and libultralitej16.so files are copied into the Android\CustDB\libs and Android\CustDB\libs\armeabi directories.

2. Run Eclipse.

   The default application path is C:\Eclipse\eclipse.exe.

3. In the Workspace field, specify a working directory that is not your CustDB sample directory, and then click OK.

4. Import the CustDB project into Eclipse.

   a. Click File » Import.
   
   b. Expand the General directory, and then click Existing Projects into Workspace. Click Next.
   
   c. In the Select Root Directory field, type %SQLANYSAMP16%\UltraLiteJ\Android\CustDB. Select Copy Projects Into Workspace, and then click Finish.

5. Make sure that the appropriate Android SDK path is specified in Eclipse.

   a. Click Window » Preferences.
   
   b. In the left pane, click Android.
   
   c. In the SDK Location field, type the location of the Android SDK and then click Apply.
      
      A list of available build targets appears.
   
   d. Click OK.

6. Make sure that the UltraLiteJ library path is specified in Eclipse.
Lesson 2: Starting the MobiLink server

In this lesson, you start the MobiLink server to perform synchronization.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Setting up a new Android project” on page 41.

Task

- Start MobiLink by running the following command from %SQLANYSAMPLE16%MobiLink\CustDB:

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

  The -c option connects MobiLink to the SQL Anywhere CustDB database. The -v+ option sets a high level of verbosity so that you can follow what is happening in the MobiLink server messages window.
The -x option specifies the port number being used for the communications. The -ot option specifies that a log file (ml.mls) is to be created in the directory where you started the MobiLink server.

Results

The MobiLink server has started.

Next

Proceed to “Lesson 3: Running your Android application” on page 44.

See also

● “MobiLink server options” [MobiLink - Server Administration]

Lesson 3: Running your Android application

In this lesson, you run your application through an Android simulator.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Setting up a new Android project” on page 41.

Task

1. Set up your Android virtual device in Eclipse.
   a. Click Window » AVD Manager.
   b. Click New.
      The Create New Android Virtual Device (AVD) window appears.
   c. In the Name field, type my_avd.
   d. In the Target field, click Android 2.2 - API Level 8.
   e. Click Create AVD.
   f. Close the AVD Manager window.

2. In the Package Explorer window, select CustDB.

3. From the Run menu, choose Run As » Android Application.
   The Android simulator loads.

4. Click Menu.
   Your Android application loads.
Lesson 4: Testing your Android application and synchronizing

In this lesson, you use your Android application to update the UltraLite remote database and synchronize the CustDB consolidated database.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Setting up a new Android project” on page 41.

Task

1. Ensure that the Employee ID field is 50, the Host field is 10.0.2.2, and the Port field is 80, and then click Save.

   The application automatically synchronizes and a set of customers, products, and orders is downloaded to the application from the CustDB consolidated database.

2. In the simulator, click Menu » New.

3. In the Customer field, choose Ace Properties.

4. In the Product field, choose 4x8 Drywall x100.

5. In the Quantity field, type 999.

6. In the Discount field, type 25.

7. Click OK to add the new order.

8. Synchronize the application with the CustDB consolidated database.

   In the simulator, click Menu and then click Sync.

9. Connect to the CustDB consolidated database with Interactive SQL.

   a. Click Start » Programs » SQL Anywhere 16 » Administration Tools » Interactive SQL, or run the following command:

      dbisql
b. Click **ODBC Data Source Name** and choose **SQL Anywhere 16 CustDB**.

c. Click **Connect**.

10. Verify that the synchronization was successful.

Execute the following SQL statement in Interactive SQL:

```sql
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'
```

Synchronization was successful when an order entry appears in Interactive SQL.

11. Close the simulator window.

**Results**

The changes you made in the simulator are synchronized with the CustDB consolidated database.

**Next**

Proceed to “Cleaning up” on page 46.

---

**Cleaning up**

Remove your recently-created tutorial materials from your computer.

**Prerequisites**

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Setting up a new Android project” on page 41.

**Task**

1. Close Eclipse.
   
   Click **File » Exit**.

2. Close MobiLink, Interactive SQL, and synchronization client windows by right-clicking each task bar item and clicking **Exit** or **Shut Down**.

3. Reset the CustDB database.
   
   Run the following command from the `%SQLANYSAMP16%\UltraLite\CustDB` directory:
   
   ```bash
   makedbs
   ```
Results

The materials are removed from your computer, and this tutorial can be repeated again from the first lesson.
Tutorial: Building a BlackBerry application

This tutorial guides you through the development of an application for BlackBerry smartphones using the UltraLiteJ API and the Eclipse environment. In this tutorial, you run the application on a Windows simulator and deploy it to a BlackBerry smartphone. Code samples are provided throughout the tutorial, and a complete code listing is available at the end of the tutorial.

Required software
- Eclipse 3.5 or later
- BlackBerry Java Plug-in for Eclipse 1.1 or later
- BlackBerry Email and MDS Services Simulator Package v4.1.4
- BlackBerry JDE 6.0 or later
- UltraLiteJ API

Competencies and experience
- Familiarity with Java
- Familiarity with Eclipse

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

Part 1: Creating a new BlackBerry application

This part explains how to create a BlackBerry application that maintains a list of names in an UltraLite Java edition database. The second part explains how to synchronize the application with a MobiLink server.

Lesson 1: Setting up a new BlackBerry project

In this lesson, you create a new BlackBerry project through the Eclipse Integrated Development Environment.

Prerequisites

This lesson assumes that you have installed the required software. See “Tutorial: Building a BlackBerry application” on page 49.

Context and remarks

This tutorial assumes that you are familiar with Java and Eclipse.

Task

1. Run Eclipse.
The default application path is C:\Eclipse\eclipse.exe.

2. In the **Workspace** field, specify a working directory and then click **OK**.
   
   This tutorial assumes that you are working in the C:\HelloBlackBerry directory.

3. Create your new project.
   
   Click **File » New » Project**.

4. Expand the **BlackBerry** folder and then select **BlackBerry Project**.

5. Click **Next**.

6. In the **Project Name** field, type **HelloBlackBerry**.

7. Click **Finish**.

8. Add the UltraLiteJ JAR file to the project.
   
   a. Display the **Package Explorer** window in Eclipse if it is not already shown.
      
      Click **Window » Show View » Package Explorer**.
   
   b. Access the package properties of your project.
      
      Click **HelloBlackBerry** in the **Package Explorer** window and then click **File » Properties**.
   
   c. In the left pane, click **Java Build Path** and then click the **Libraries** tab.
   
   d. Click **Add External Jars**, and then open \UltraLite\UltraLiteJ\BlackBerry4.2\UltraLiteJ16.jar from your SQL Anywhere installation directory.

9. Add the path to your UltraLiteJ Javadoc documentation to the project.
   
   a. In the **JARs And Class Folders On The Build Path** list, expand **UltraLiteJ16.jar** and click **JavaDoc Location**.
   
   b. Click **Edit**.
      
      The **Javadoc For UltraLiteJ16.Jar** window appears.
   
   c. Click **Browse** and then open \UltraLite\UltraLiteJ\BlackBerry4.2\html from your SQL Anywhere installation directory.
   
   d. Click **OK** to close the **Javadoc For UltraLiteJ16.Jar** window.

10. Click **OK** to close the window.

**Results**

The UltraLiteJ API is functional in the new BlackBerry application.

**Next**

Proceed to “Lesson 2: Writing and testing your BlackBerry application” on page 51.
Lesson 2: Writing and testing your BlackBerry application

In this lesson, you create a class with a main method in Eclipse that opens a HomeScreen class, which contains a title and a status message. You then compile and test the application.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Setting up a new BlackBerry project” on page 49.

Task

1. Add an Application class to your project.
   a. In the Package Explorer window, expand HelloBlackBerry and click src.
   b. Click File » New » Class.
      The New Java Class window appears.
   c. In the Name field, type Application.
   d. Under the Which Method Stubs Would You Like To Create option, select Public Static Void Main([String() Args]).
   e. Click Finish.
      The Application.java file appears under your project in the Package Explorer window.

2. Modify the Application class.

   Double-click Application.java in the Package Explorer window, and then add a constructor and a main method.

   Your Application.java code should look like the following code:

   ```java
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. Add a HomeScreen class to your project.
   a. In the Package Explorer window, expand HelloBlackBerry and click src.
   b. Click File » New » Class.
      The New Java Class window appears.
   c. In the Name field, type HomeScreen.
   d. Click Finish.
      The HomeScreen.java file appears under your project in the Package Explorer window.
4. Modify the **HomeScreen** class so that it displays a title and status messages.

Double-click `HomeScreen.java` in the **Package Explorer** window, and then update the code so that it displays a title and a status message.

Your `HomeScreen.java` code should look like the following code:

```java
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;
}
```

The `_statusLabel` is defined as a class variable so that it can be accessed from other parts of the application later.

5. Run the simulator.

In the **Package Explorer** window, click `Application.java`, and then click **Run ➔ Run As ➔ BlackBerry Simulator**.

**Note**
If multiple projects are open in your workspace, click **Run ➔ Run Configurations**, select **HelloBlackBerry**, and then click **Run**.

The **HelloBlackBerry** project compiles and then the simulator window appears.

Ensure that the project compiles without errors by selecting the **Problems** tab in Eclipse.

6. From the simulator menu, click **Simulate ➔ Set IT Policy**.

The **Set IT Policy** window appears.

7. In the **Policy** field, click **Allow Third Party Apps To Use Persistent Store ➔**.

8. Click **Set** and then click **Close**.

9. Launch your application.

In the simulator window, navigate to **Downloads** and then run the **HelloBlackBerry** application.
10. Stop the simulation.

   In the simulator window, click File » Exit.

Results

The BlackBerry application runs successfully in the simulator.

Next


Lesson 3: Creating an UltraLite Java edition database

In this lesson, you write code to create and connect to an UltraLite Java edition database.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Setting up a new BlackBerry project” on page 49.

Context and remarks

The code that creates a new database is defined in a singleton class named `DataAccess`, and is invoked from the `HomeScreen` constructor. Using a singleton class ensures that only one database connection is open at a time. While the UltraLiteJ API supports multiple connections, it is a common design pattern to use a single connection.

Task

1. Modify the `HomeScreen` class to instantiate a `DataAccess` object.

   The following is the complete and updated `HomeScreen` class code listing:

```java
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 may report warnings indicating that DataAccess cannot be resolved. The DataAccess object is held as a class-level variable so that it can be accessed from other parts of the code. You create the DataAccess class in the next step.

2. Add a DataAccess class to your project.
   a. In the Package Explorer window, expand HelloBlackBerry and click src.
   b. Click File » New » Class.
      The New Java Class window appears.
   c. In the Name field, type DataAccess.
   d. Click Finish.
      The DataAccess.java file appears under your project in the Package Explorer window.

3. Modify the DataAccess class so that it contains a getDataAccess method that ensures a single database connection.

   Double-click DataAccess.java in the Package Explorer window, and then replace the code with the following snippet:

   ```java
   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 static Connection _conn;
private static DataAccess _da;

This class imports the `com.ianywhere.ultralitej16` package from the `UltraLiteJ16.jar` file. The following steps are needed to create or connect to an UltraLite Java edition database:

a. Define a configuration. In this tutorial, the configuration object is defined by the `ConfigObjectStore` interface, which allows you to configure a persistent database that resides in the BlackBerry object store.

b. Attempt to connect to the database. In this tutorial, the database is created using the `createDatabase` method when the connection attempt fails. This method then returns an open connection.

4. Click File » Save.

5. Run the simulator.

In the Package Explorer window, click `Application.java`, and then click Run » Run As » BlackBerry Simulator.

**Note**
If multiple projects are open in your workspace, click Run » Run Configurations, select `HelloBlackBerry`, and then click Run.

The `HelloBlackBerry` project compiles and then the simulator window appears.

Ensure that the project compiles without errors by selecting the Problems tab in Eclipse.

6. From the simulator menu, click File » Load Java Program.

7. Browse to the `UltraLite\UltraLiteJ\BlackBerry4.2\` directory of your SQL Anywhere installation and open the `UltraLiteJ16.cod` file.

**Note**
You may need to copy `UltraLiteJ16.cod` and the DBG files to the working simulator directory (for example, `C:\Eclipse\plugins\net.rim.ejde.componentpack6.0.0_6.0.0.0.26\components\simulator\`) to run the application. When copied, you do not need to load the Java program from the simulator menu.

8. From the simulator menu, click Simulate » Set IT Policy.
The **Set IT Policy** window appears.

9. In the **Policy** field, click **Allow Third Party Apps to Use Persistent Store** and then click »».

10. Click **Set** and then click **Close**.

11. Launch your application.

   In the simulator window, navigate to **Downloads** and then run the **HelloBlackBerry** application.

   A screen appears that displays the **Hello BlackBerry** title bar and the **Status: Connected** text, which indicates that the application has successfully connected to the UltraLite Java edition database.

12. Stop the simulation.

   In the simulator window, click **File » Exit**.

**Results**

The application runs in the simulator, and the code creates an UltraLite Java Edition database.

**Next**

Proceed to “**Lesson 4: Creating a table in the database**” on page 56.

**See also**

- “**ConfigObjectStore interface [BlackBerry] [UltraLiteJ]**” on page 89
- “**DatabaseManager.createDatabase method [UltraLiteJ]**” on page 138

**Lesson 4: Creating a table in the database**

In this lesson, you update the application code to create a table named **Names** in the UltraLite Java edition database.

**Prerequisites**

This lesson assumes you have completed all preceding lessons. See “**Lesson 1: Setting up a new BlackBerry project**” on page 49.

**Task**

1. Add a new method to the **DataAccess** class that creates the **Names** table.

   Double-click **DataAccess.java** in the **Package Explorer** window, and then insert the following code after the **getDataSource** method:

   ```java
   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.executeUpdate();
  ps.close();
}
} catch (ULjException uex1) {
  Dialog.alert("ULjException: " + uex1.toString());
} catch (Exception ex1) {
  Dialog.alert("Exception: " + ex1.toString());
}

This method throws an exception if the Names table already exists in the database.

The table contains two columns that have the following properties:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Allow null?</th>
<th>Default</th>
<th>Primary key?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>UUID</td>
<td>No</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Name</td>
<td>Varchar(254)</td>
<td>No</td>
<td>None</td>
<td>No</td>
</tr>
</tbody>
</table>

2. Call the createDatabaseSchema method from the getDataAccess method.

Remove the code comments from the getDataAccess method so that the createDatabaseSchema calls look like the following code snippet:

  _da.createDatabaseSchema()

3. Compare your DataAccess code to the complete code listing of the DataAccess class to ensure that they are identical.

4. Click File » Save.

5. Run the simulator to verify that the application compiles and runs.

In the Package Explorer window, click Application.java, and then click Run » Run As » BlackBerry Simulator.

**Note**
If multiple projects are open in your workspace, click Run » Run Configurations, select HelloBlackBerry, and then click Run.

The HelloBlackBerry project compiles and then the simulator window appears.

Ensure that the project compiles without errors by selecting the Problems tab in Eclipse.

6. From the simulator menu, click File » Load Java Program.

7. Browse to the UltraLite\UltraLiteJ\BlackBerry4.2\ directory of your SQL Anywhere installation and open the UltraLiteJ16.cod file.
8. From the simulator menu, click **Simulate** » **Set IT Policy**.

The **Set IT Policy** window appears.

9. Click **Policy** » **Allow Third Party Apps to Use Persistent Store** and then click »».

10. Click **Set** and then click **Close**.

11. Launch your application.

    In the simulator window, navigate to **Downloads** and then run the **HelloBlackBerry** application.

    A screen appears that displays the **Hello BlackBerry** title bar and the **Status: Connected** text, which indicates that the application has successfully connected to the UltraLite Java edition database.

12. Stop the simulation.

    In the simulator window, click **File** » **Exit**.

**Results**

The application runs the new code and the **Names** table is created in the UltraLite Java edition database.

**Next**

Proceed to “Lesson 5: Adding data to the table” on page 58.

**See also**

- “**DataAccess.java**” on page 73

---

**Lesson 5: Adding data to the table**

In this lesson, you populate the UltraLite Java edition database by adding several controls to your application, implementing code to insert data into the **Names** table, running the application in a simulator, and then using the controls to input data.

**Prerequisites**

This lesson assumes you have completed all preceding lessons. See “**Lesson 1: Setting up a new BlackBerry project**” on page 49.
Task

1. Update the HomeScreen class to add the controls.

   Double-click HomeScreen.java in the Package Explorer window, and then insert the following code above the try-catch statement that calls the getDataAccess method:

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

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

   // Add a menu item
   addMenuItem(_addToListMenuItem);

2. Add class-level declarations for _nameEditField and _nameListField, and then define a MenuItem with a run method (empty for now). These declarations belong next to the declarations of _statusLabel and _da.

   Insert the following code below the private DataAccess _da; statement:

   private EditField _nameEditField;
   private ObjectListField _nameListField;

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

3. Add a new method to the DataAccess class that inserts a row into a table.

   Double-click DataAccess.java in the Package Explorer window, and then insert the following code after the createDatabaseSchema method:

   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(UljeException uex) {
        Dialog.alert("ULjException: " + uex.toString());
      }
      catch( Exception ex ){
        Dialog.alert("Exception: " + ex.toString());
      }
   }
4. Add a NameRow class to your project.
   a. In the Package Explorer window, expand HelloBlackBerry and click src.
   b. Click File » New » Class.
      The New Java Class window appears.
   c. In the Name field, type NameRow.
   d. Click Finish.
      The NameRow.java file appears under your project in the Package Explorer window.

5. Update the NameRow class so that it can store a row in the Names table as an object.

   Double-click NameRow.java in the Package Explorer window, and then replace the code with the following snippet:

   ```java
   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;
   }
   
   The toString method is used by the ObjectListField control.
   ```

6. Add a new method to the DataAccess class that reads a row into a Vector of objects.

   Double-click DataAccess.java in the Package Explorer window, and then insert the following code after the insertName method:

   ```java
   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);
           }
       }
   }
   ```
7. Add a new method to the **HomeScreen** class that refreshes the contents of the list displayed on the screen.

Double-click **HomeScreen.java** in the **Package Explorer** window, and then insert the following code after the **_addToListMenuItem** method:

```java
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. Update the **HomeScreen** class so that it calls the **refreshNameList** method, ensuring that the list is filled when the application starts.

Insert the following code before the end of the **HomeScreen** constructor:

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

9. Add a new method to the **HomeScreen** class that adds a row to the list on the screen.

Insert the following code after the **refreshNameList** method:

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

10. Update the **run** method in the **HomeScreen** class so that it calls the **onAddToList** method.

Replace the line of code that states `\ TODO` with the following code snippet:

```java
onAddToList();
```

11. Click **File** » **Save All**.

12. Run the simulator to verify that the application compiles and runs.
In the **Package Explorer** window, click *Application.java*, and then click **Run» Run As» BlackBerry Simulator**.

**Note**
If multiple projects are open in your workspace, click **Run» Run Configurations**, select **HelloBlackBerry**, and then click **Run**.

The **HelloBlackBerry** project compiles and then the simulator window appears.

Ensure that the project compiles without errors by selecting the **Problems** tab in Eclipse.

13. From the simulator menu, click **File » Load Java Program**.

14. Browse to the `UltraLite\UltraLite\BlackBerry4.2\` directory of your SQL Anywhere installation and open the `UltraLiteJ16.cod` file.

**Note**
You may need to copy `UltraLiteJ16.cod` and the DBG files to the working simulator directory (for example, `C:\Eclipse\plugins\net.rim.ejde.componentpack6.0.0_6.0.0.0.26\components\simulator\`) to run the application. When copied, you do not need to load the Java program from the simulator menu.

15. From the simulator menu, click **Simulate » Set IT Policy**.

The **Set IT Policy** window appears.

16. In the **Policy** field, click **Allow Third Party Apps To Use Persistent Store » »»**.

17. Click **Set** and then click **Close**.

18. Launch your application.

   In the simulator window, navigate to **Downloads** and then run the **HelloBlackBerry** application.

   A screen appears that displays the **Hello BlackBerry** title bar, the **Status: Connected** text, and a **Name** field.

19. At the name field, type **John Smith**.

20. Click ***EMPTY*» and then choose **Add**.

   **John Smith** appears in the list, which indicates that the name entry was added to the **Names** table in the database.

   Names are stored in the database as you add them. They are retrieved from the database and added to the list when you close and re-open the application.

21. Stop the simulation.

   In the simulator window, click **File » Exit**.
Results

The inputted data is inserted into the Names table in the UltraLite Java edition database.

Next

Proceed to “Lesson 6: Deploying your application to a BlackBerry smartphone” on page 63.

Lesson 6: Deploying your application to a BlackBerry smartphone

This lesson explains how to sign and deploy the application using the BlackBerry Desktop Manager software.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Setting up a new BlackBerry project” on page 49.

You must obtain a key from RIM so that you can use the BlackBerry Signature Tool to sign your application. For more information about obtaining keys, see the BlackBerry Developer Program web site at http://na.blackberry.com/eng/developers/.

Context and remarks

Applications running on a BlackBerry must be signed using the BlackBerry Signature Tool. This tool is available from Research in Motion (RIM) as part of the BlackBerry JDE Component Package. The UltraLiteJ16.cod file is already signed, but you must sign the HelloBlackBerry.cod file.

Task

1. Start the BlackBerry Signature Tool.

2. Run the following command from the \bin directory of your BlackBerry JDE installation (for example, C:\Program Files\Research in Motion\BlackBerry JDE 6.0.0\bin).
   
   ```java
   start javaw -jar SignatureTool.jar
   ```

3. Browse to C:\HelloBlackBerry and select the HelloBlackBerry.cod file, your compiled application.

4. Click Request To Sign The File.

5. Click Close to close the signature tool.

6. Connect your BlackBerry to your computer using the USB cable, and ensure that the BlackBerry Desktop Manager can see the device.

7. Click Application Loader and follow the instructions in the wizard.
8. Browse to the HelloBlackBerry.alx file and add it to your device.

9. Browse to the BlackBerry4.2\UltraLiteJ.alx file and add it to your device.

**Results**

The application is deployed and you can run the application on your BlackBerry smartphone.

**Next**

Proceed to “Part 2: Using MobiLink to synchronize the BlackBerry application” on page 64.

---

**Part 2: Using MobiLink to synchronize the BlackBerry application**

This part of the tutorial extends the BlackBerry application to support MobiLink synchronization. You complete the following tasks:

- Create a SQL Anywhere database that can synchronize with your UltraLite Java edition database.
- Start a MobiLink server to handle synchronizations.
- Update your BlackBerry application to support MobiLink synchronization.
- Synchronize your BlackBerry application with the consolidated database.

---

**Lesson 1: Setting up your MobiLink consolidated database**

In this lesson, you create a SQL Anywhere database.

**Prerequisites**

This lesson assumes that you have installed the required software and completed part one of this tutorial. See “Tutorial: Building a BlackBerry application” on page 49.

**Context and remarks**

Data synchronization requires a MobiLink consolidated database for the UltraLite database to synchronize with.

**Task**

1. Create a working directory to store the SQL Anywhere database.

   This tutorial assumes that you are working in the c:\HelloBlackBerry\database directory.

2. Run the following command to create an empty SQL Anywhere database with a DBA user ID of DBA and password of sql:
3. Create an ODBC data source to connect to the database.
   a. Click **Start** » **Programs** » **SQL Anywhere 16** » **Administration Tools** » **ODBC Data Source Administrator**.
   b. Click the **User DSN** tab, and then click **Add**.
   c. In the **Create New Data Source** window, click **SQL Anywhere 16** and click **Finish**.
   d. Click the **ODBC** tab.
   e. In the **Data Source Name** field, type **HelloBlackBerry**.
   f. Click the **Login** tab.
   g. In the **User ID** field, type **DBA**.
   h. In the **Password** field, type **sql**.
   i. From the **Action** list, click **Start And Connect To A Database On This Computer**.
   j. In the **Database File** field, type `c:\tutorial\database\HelloBlackBerry.db`.
   k. In the **Server Name** fields, type **HelloBlackBerry**.
   l. Disable the **Stop Database After Last Disconnect** option.
   m. Click **OK**.
   n. Click **OK** on the **ODBC Data Source Administrator** window.

4. Run the following command to start Interactive SQL and connect to the SQL Anywhere database:

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

5. Execute the following SQL statement in Interactive SQL to create the **Names** table on the consolidated database:

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

6. Close Interactive SQL.

   Click **File** » **Exit**.

**Results**

The SQL Anywhere database is created.

**Next**

Proceed to “Lesson 2: Setting up a MobiLink server and deploying a synchronization model” on page 66.
Lesson 2: Setting up a MobiLink server and deploying a synchronization model

In this lesson, you use Sybase Central to prepare your consolidated database for synchronization.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Setting up your MobiLink consolidated database” on page 64.

Task

1. Click **Start** » **Programs** » **SQL Anywhere 16» Administration Tools** » **Sybase Central**.

2. Click **Tools** » **MobiLink 16» New Project**.

The **Create Project Wizard** appears.

3. In the **Name** field, type **rim_project**.

4. In the **Location** field, type **C:\HelloBlackBerry\database**, and then click **Next**.

5. In the **Database Display Name** field, type **HelloBlackBerry**.

6. Click **Edit**.

7. Perform the following tasks on the **Connect To A Generic ODBC Database** page:
   a. In the **User ID** field, type **DBA**.
   b. In the **Password** field, type **sql**.
   c. In the **ODBC Data Source Name** field, click **Browse** and select **HelloBlackBerry**.
   d. Click **OK**, and then click **Save**.

8. Select the **Remember The Password** option, and then click **Next**.

9. Ensure that only the **Names** table is selected from the **Which Consolidated Database Tables And Columns Do You Want To Have In Your Remote Database** list, and then click **Next** on all remaining dialogs to end of the wizard.

A synchronization model has been created.

10. Right-click your new Synchronization model, and select **Properties**.

11. In the first text field, type **HelloBlackBerrySyncModel**, click **Apply**, and then click **Ok**.

12. In the left pane of Sybase Central under **MobiLink 16**, expand **rim_project**, **Synchronization Models** and then **HelloBlackBerrySyncModel**.

13. Click **File» Deploy**.
14. Under the **Specify The Deployment Details For One Or More Of The Following** option, ensure that only the **Consolidated Database** option is selected. Click **Next**.

15. Perform the following tasks on the **Consolidated Database Deployment Destination** page:
   a. Select **Save Changes To The Following SQL File** and accept the default location for the file. MobiLink generates a `.sql` file that makes changes to the consolidated database to set up for synchronization. You can examine the `.sql` file later and make your own changes. Then, you must run the `.sql` file manually.
   b. Immediately apply the changes to the consolidated database.
      Select **Connect To The Consolidated Database To Directly Apply The Changes**.
   c. Select the **HelloBlackBerry** consolidated database from the list.
   d. Click **Next**.

A prompt appears asking to create the `consolidated` directory. Click **Yes**.

16. On the **MobiLink User and Synchronization Profile** page, type `mluser` for the user name and `mlpassword` for the password, and then click **Finish**.

**Results**

A MobiLink project is created and the synchronization model for the consolidated database can now be deployed.

**Next**

Proceed to “Lesson 3: Adding MobiLink support to your BlackBerry application” on page 67.

**Lesson 3: Adding MobiLink support to your BlackBerry application**

In this lesson, you add synchronization capabilities to your application.

**Prerequisites**

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Setting up your MobiLink consolidated database” on page 64.

**Task**

1. Update the **HomeScreen** class to add a **Sync** menu item.

   Double-click `HomeScreen.java` in the **Package Explorer** window, and then insert the following code above the **try-catch** statement that calls the **getDataAccess** method:

   ```java
   // Add sync menu item
   addItem(_syncMenuItem);
   ```
2. Update the **HomeScreen** class to add a new method that defines the menu item in the class variable declarations.

Insert the following code below the `_addToListMenuItem` method:

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

3. Update the **HomeScreen** class to add the `onSync` method that is called in the previous step.

Insert the following code below the `onAddToList` method:

```java
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. Update the **DataAccess** class to define the `_syncParms` variable.

Double-click `DataAccess.java` in the **Package Explorer** window, and then insert the following code below the `private static DataAccess _da;` call:

```java
private static SyncParms _syncParms;
```

5. Update the **DataAccess** class to add a `sync` method.

Insert the following code below the `getNameVector` method:

```java
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) {
```

**Note**
You must replace `your-host-name` with your computer name. You cannot use this term in your application.
The synchronization parameters object, _syncParms, includes the user name and password that you specified when deploying the synchronization model. It also includes the name of the synchronization model you created. In MobiLink, this name can refer to the synchronization version or a set of synchronization logic that was deployed to your consolidated database.

The stream parameters object, StreamHTTPParms, indicates the host name and port number of the MobiLink server. When you start the MobiLink server in the next lesson, use your own computer name for simulator testing and select a port that is available.

**Note**
When using a device, use an externally visible computer or a computer that is accessible from the BlackBerry Enterprise Server your device is paired to, such as the Sybase-hosted Relay Server. For more information about the Relay Server, see “Introduction to the Relay Server” [Relay Server].

6. Click File » Save All.

**Results**

The BlackBerry application is capable of synchronizing with the consolidated database.

**Next**

Proceed to “Lesson 4: Starting the MobiLink server and synchronizing the application” on page 69.

**Lesson 4: Starting the MobiLink server and synchronizing the application**

In this lesson, you start the MobiLink server and synchronize the UltraLite Java edition database with the SQL Anywhere consolidated database.

**Prerequisites**

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Setting up your MobiLink consolidated database” on page 64.

**Context and remarks**

Before you can run the BlackBerry application and synchronize, the MobiLink server must be running. The MDS Simulator must also be running to provide a communication channel between the device simulator and MobiLink.
Task

1. Start MobiLink by running the following command from `c:\HelloBlackBerry\database\`:

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

   The `-c` option connects MobiLink to the SQL Anywhere database. The `-v+` option sets a high level of verbosity so that you can follow what is happening in the MobiLink server messages window. The `-x` option indicates the port number being used for the communications. The `-ot` option specifies that a log file (`ml.mls`) is to be created in the directory where you started the MobiLink server.

2. Run the MDS simulator so that the BlackBerry simulator can communicate over a network.

   Click `Start » Programs » Research In Motion » BlackBerry Email And MDS Services Simulator 4.1.4 » MDS`.

3. Add names to the MobiLink consolidated database so that your application updates the UltraLite Java edition database when synchronizing.

   a. Run the following command to start Interactive SQL and connect to the SQL Anywhere database:

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

   b. Execute the following SQL statement in Interactive SQL to add names to the `Names` table:

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

   c. Close Interactive SQL.

      Click `File » Exit`.

4. Run the simulator from Eclipse.

   In the `Package Explorer` window, click `Application.java`, and then click `Run » Run As » BlackBerry Simulator`.

   The `HelloBlackBerry` project compiles and then the simulator window appears.

   Ensure that the project compiles without errors by selecting the `Problems` tab in Eclipse.

5. From the simulator menu, click `File » Load Java Program`.

6. Browse to the `UltraLite\UltraLiteJ\BlackBerry4.2` directory of your SQL Anywhere installation and open the `UltraLiteJ16.cod` file.
7. From the simulator menu, click **Simulate » Set IT Policy**.

   The **Set IT Policy** window appears.

8. In the **Policy** field, click **Allow Third Party Apps To Use Persistent Store » »»**.

9. Click **Set** and then click **Close**.

10. Launch your application.

    In the simulator window, navigate to **Downloads** and then run the **HelloBlackBerry** application.

    A screen appears that displays the **Hello BlackBerry** title bar, the **Status: Connected** text, and a **Name** field.

11. Synchronize the application with the MobiLink server.

    Click ***EMPTY*** and then choose **Sync**.

    **Jane Smith** and **David Smith** appear in the list, which indicates that the application was able to synchronize with the MobiLink consolidated database. If you query the names in the **Names** table from Interactive SQL, you should see that any names you have entered in the simulator have reached the server.

12. Stop the simulation.

    In the simulator window, click **File » Exit**.

**Results**

The UltraLite Java edition and SQL Anywhere consolidated databases are synchronized.

**Next**

Proceed to “Cleaning up” on page 71.

---

**Cleaning up**

Remove your recently-created tutorial materials from your computer.

**Prerequisites**

This lesson assumes you have completed all preceding lessons. See “Part 1: Creating a new BlackBerry application” on page 49.
Task

1. Close Eclipse.
   Click **File » Exit**.

2. Close the command window that is running the MDS simulator.

3. Close Sybase Central and Interactive SQL by right-clicking each task bar item and choosing **Exit**.

4. Delete the **HelloBlackBerry** data source:
   a. Start the ODBC Data Source Administrator.
      Click **Start » Programs » SQL Anywhere 16 » Administration Tools » ODBC Data Source Administrator**.
   b. Select **HelloBlackBerry** from the list of **User Data Sources**, and click **Remove**.
   c. Close the ODBC Data Source Administrator.

5. Delete the **C:\HelloBlackBerry** directory.

Results

The materials are removed from your computer, and this tutorial can be repeated again from the first lesson.

Code listing for tutorial

This section provides the complete code for the preceding tutorial. There are four Java classes that are present in the tutorials. They are also available in `%SQLANYSAMP16%\UltraLite\BlackBerry \HelloBlackBerry\myapp`.

See also

- “Part 1: Creating a new BlackBerry application” on page 49
- “Part 2: Using MobiLink to synchronize the BlackBerry application” on page 64

**Application.java**

```java
//
***************************************************************************
// Copyright © 2014 SAP AG or an SAP affiliate company. 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();
    }
}

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...".);
                    }
                }
            }
        }
        return _da;
    }
}

DataAccess.java
// Copyright © 2014 SAP AG or an SAP affiliate company. 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;
_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 uex) {
        Dialog.alert("ULjException: " + uex.toString());
    } catch (Exception ex) {
        Dialog.alert("Exception: " + ex.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**

```java
// ***************************************************************************
// Copyright © 2014 SAP AG or an SAP affiliate company. 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
    }

```
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

```java
// ***************************************************************************
// Copyright © 2014 SAP AG or an SAP affiliate company. 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 reference

The following list describes some of the commonly used API objects:

- **DatabaseManager**  Provides methods for managing database connections, such as CreateDatabase and connect.

- **Connection**  Represents a connection to an UltraLite database. You can create one or more Connection objects.

- **SyncParms**  Synchronizes your UltraLite database with a MobiLink server.

- **PreparedStatement, ResultSet**  Create dynamic SQL statements, make queries, and execute INSERT, UPDATE, and DELETE statements, and attain programmatic control over database result sets.

Package [Android]
  com.ianywhere.ultralitejni16

Package [BlackBerry]
  com.ianywhere.ultralitej16

See also

- “Android and BlackBerry setup considerations” on page 4
- “DatabaseManager class [UltraLiteJ]” on page 134
- “Connection interface [UltraLiteJ]” on page 102
- “SyncParms class [UltraLiteJ]” on page 242
- “PreparedStatement interface [UltraLiteJ]” on page 175
- “ResultSet interface [UltraLiteJ]” on page 191

ColumnSchema interface

Specifies the schema of a column.

**Syntax**

```java
public interface ColumnSchema
```

**Members**

All members of the ColumnSchema interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN_DEFAULT_AUTOFILENAME variable</td>
<td>Indicates the AUTOFILENAME column default attribute.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>COLUMN_DEFAULT_AUTOINC variable</td>
<td>Indicates the AUTOINCREMENT column default attribute.</td>
</tr>
<tr>
<td>COLUMN_DEFAULT_CONSTANT variable</td>
<td>Indicates a constant column default attribute.</td>
</tr>
<tr>
<td>COLUMN_DEFAULT_CURRENT_DATE variable</td>
<td>Indicates the CURRENT DATE (year, month, and day) column default attribute.</td>
</tr>
<tr>
<td>COLUMN_DEFAULT_CURRENT_TIME variable</td>
<td>Indicates the CURRENT TIME column default attribute.</td>
</tr>
<tr>
<td>COLUMN_DEFAULT_CURRENT_TIMESTAMP variable</td>
<td>Indicates the CURRENT TIMESTAMP column default attribute.</td>
</tr>
<tr>
<td>COLUMN_DEFAULT_CURRENT_UTC_TIMESTAMP variable</td>
<td>Indicates the CURRENT UTC TIMESTAMP column default attribute.</td>
</tr>
<tr>
<td>COLUMN_DEFAULT_GLOBAL_AUTOINC variable</td>
<td>Indicates the GLOBAL AUTOINCREMENT column default attribute.</td>
</tr>
<tr>
<td>COLUMN_DEFAULT_NONE variable</td>
<td>Indicates that the column has no default attribute.</td>
</tr>
<tr>
<td>COLUMN_DEFAULT_UNIQUE_ID variable</td>
<td>Indicates a new unique identifier column default attribute.</td>
</tr>
</tbody>
</table>

**Remarks**

This interface only contains constants for different column default values stored in column_default column of the syscolumn system table.

**COLUMN_DEFAULT_AUTOFILENAME variable**

Indicates the AUTOFILENAME column default attribute.

**Syntax**

```java
final byte ColumnSchema.COLUMN_DEFAULT_AUTOFILENAME
```

**Remarks**

When a VARCHAR column has this default value, the column is the filename column in an external blob definition.

When a column has this type of default, the column_default_value column in the TableSchema.SYS_COLUMNS system table contains the prefix and extension strings found in the external blob definition, in the form 'prefix|extension'.
The default value of existing tables can be determined by querying the column_default column in the TableSchema.SYS_COLUMNS system table.

See also
- “TableSchema.SYS_COLUMNS variable [UltraLiteJ]” on page 273

COLUMN_DEFAULT_AUTOINC variable
Indicates the AUTOINCREMENT column default attribute.

Syntax
final byte ColumnSchema.COLUMN_DEFAULT_AUTOINC

Remarks
When using the AUTOINCREMENT attribute, the column must be one of the integer data types, or an exact numeric type. On an INSERT, if a value is not specified for the AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column that is larger than the current maximum value for the column, the value is used as a starting point for subsequent inserts.

In UltraLiteJ, the autoincremented value is not set to 0 when the table is created, and the AUTOINCREMENT attribute generates negative numbers when a signed data type is used for the column. Therefore, declare AUTOINCREMENT columns as unsigned integers to prevent negative values from being used.

The default value of existing tables can be determined by querying the column_default column in the TableSchema.SYS_COLUMNS system table.

See also
- “TableSchema.SYS_COLUMNS variable [UltraLiteJ]” on page 273

COLUMN_DEFAULT_CONSTANT variable
Indicates a constant column default attribute.

Syntax
final byte ColumnSchema.COLUMN_DEFAULT_CONSTANT

Remarks
The default value of existing tables can be determined by querying the column_default column in the TableSchema.SYS_COLUMNS system table.

See also
- “TableSchema.SYS_COLUMNS variable [UltraLiteJ]” on page 273
**COLUMN_DEFAULT_CURRENT_DATE variable**
Indicates the CURRENT DATE (year, month, and day) column default attribute.

Syntax
```java
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT_DATE
```

Remarks
See "CURRENT DATE special value" under "Special values in UltraLite" in the SQL Anywhere documentation set.

The default value of existing tables can be determined by querying the column_default column in the TableSchema.SYS_COLUMNS system table.

See also
● “TableSchema.SYS_COLUMNS variable [UltraLiteJ]” on page 273

**COLUMN_DEFAULT_CURRENT_TIME variable**
Indicates the CURRENT TIME column default attribute.

Syntax
```java
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT_TIME
```

Remarks
See "CURRENT TIME special value" under "Special values in UltraLite" in the SQL Anywhere documentation set.

The default value of existing tables can be determined by querying the column_default column in the TableSchema.SYS_COLUMNS system table.

See also
● “TableSchema.SYS_COLUMNS variable [UltraLiteJ]” on page 273

**COLUMN_DEFAULT_CURRENT_TIMESTAMP variable**
Indicates the CURRENT TIMESTAMP column default attribute.

Syntax
```java
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT_TIMESTAMP
```

Remarks
This constant combines the CURRENT DATE and CURRENT TIME values to form a TIMESTAMP value, which contains the year, month, day, hour, minute, second, and fraction of a second. The precision
of the fraction is set to 3 decimal places. The accuracy of this constant is limited by the accuracy of the system clock.

See "CURRENT TIMESTAMP special value" under "Special values in UltraLite" in the SQL Anywhere documentation set.

The default value of existing tables can be determined by querying the column_default column in the TableSchema.SYS_COLUMNS system table.

See also
- “TableSchema.SYS_COLUMNS variable [UltraLiteJ]” on page 273

COLUMN_DEFAULT_CURRENT.UTC_TIME variable
Indicates the CURRENT UTC TIME column default attribute.

Syntax
```java
final byte ColumnSchema.COLUMN_DEFAULT_CURRENT.UTC_TIME
```

Remarks
This constant combines the CURRENT DATE and CURRENT TIME values to form a UTC TIMESTAMP value, which contains the year, month, day, hour, minute, second, and fraction of a second as observed in GMT. The precision of the fraction is set to 3 decimal places. The accuracy of this constant is limited by the accuracy of the system clock.

See "CURRENT UTC TIMESTAMP special value" under "Special values in UltraLite" in the SQL Anywhere documentation set.

The default value of existing tables can be determined by querying the column_default column in the TableSchema.SYS_COLUMNS system table.

See also
- “TableSchema.SYS_COLUMNS variable [UltraLiteJ]” on page 273

COLUMN_DEFAULT_GLOBAL_AUTOINC variable
Indicates the GLOBAL AUTOINCREMENT column default attribute.

Syntax
```java
final byte ColumnSchema.COLUMN_DEFAULT_GLOBAL_AUTOINC
```

Remarks
This constant is similar to the AUTOINCREMENT attribute, but the domain is partitioned. Each partition contains the same number of values. You must assign each copy of the database a unique global database
identification number. UltraLiteJ supplies default values in a database from the partition uniquely identified by that database’s number.

The default value of existing tables can be determined by querying the column_default column in the TableSchema.SYS_COLUMNS system table.

See also
- “TableSchema.SYS_COLUMNS variable [UltraLiteJ]” on page 273
- “Connection.setDatabaseId method [UltraLiteJ]” on page 117

COLUMN_DEFAULT_NONE variable

Indicates that the column has no default attribute.

Syntax

```java
final byte ColumnSchema.COLUMN_DEFAULT_NONE
```

Remarks

The following defaults apply when no default attribute is assigned:

- Nullable columns default to null
- Not nullable numeric columns default to zero
- Not nullable varying length columns default to zero length values.

The default value of existing tables can be determined by querying the column_default column in the TableSchema.SYS_COLUMNS system table.

See also

- “TableSchema.SYS_COLUMNS variable [UltraLiteJ]” on page 273

COLUMN_DEFAULT_UNIQUE_ID variable

Indicates a new unique identifier column default attribute.

Syntax

```java
final byte ColumnSchema.COLUMN_DEFAULT_UNIQUE_ID
```

Remarks

UUIDs can be used to uniquely identify rows in a table. The generated values are unique on every computer or device, meaning they can be used as keys in synchronization and replication environments.

The default value of existing tables can be determined by querying the column_default column in the TableSchema.SYS_COLUMNS system table.
See also

- “TableSchema.SYS_COLUMNS variable [UltraLiteJ]” on page 273

## ConfigFile interface

Establishes a Configuration object for a persistent database saved in a file.

### Syntax

```
public interface ConfigFile
```

### Base classes

- “ConfigPersistent interface [UltraLiteJ]” on page 91

### Derived classes

- “ConfigFileAndroid interface [Android] [UltraLiteJ]” on page 86

### Members

All members of the ConfigFile interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableAesDBEncryption method</td>
<td>Enables AES encryption of the database.</td>
</tr>
<tr>
<td>enableObfuscation method</td>
<td>Enables obfuscation of the database.</td>
</tr>
<tr>
<td>getCacheSize method</td>
<td>Returns the cache size of the database, in bytes.</td>
</tr>
<tr>
<td>getConnectionString method [Android]</td>
<td>Gets the connection string registered with the setConnectionString method.</td>
</tr>
<tr>
<td>getCreationString method [Android]</td>
<td>Gets the creation string registered with the setCreationString method.</td>
</tr>
<tr>
<td>getDatabaseName method</td>
<td>Returns the database name.</td>
</tr>
<tr>
<td>getEncryptionKey method</td>
<td>Gets the database encryption key that was registered with the setEncryptionKey method.</td>
</tr>
<tr>
<td>getLazyLoadIndexes method [BlackBerry]</td>
<td>Determines if lazy loading indexes is turned on.</td>
</tr>
<tr>
<td>getPageSize method</td>
<td>Returns the page size of the database, in bytes.</td>
</tr>
<tr>
<td>getRowScoreFlushSize method [BlackBerry]</td>
<td>Returns the current row score flush size.</td>
</tr>
</tbody>
</table>
### Name | Description
--- | ---
getRowScoreMaximum method [BlackBerry] | Returns the current row score maximum size.
getUserName method [Android] | Gets the name of user set by the setUserName method.
hasShadowPaging method [BlackBerry] | Determines if shadow paging is turned on.
setCacheSize method | Sets the cache size of the database, in bytes.
setConnectionString method [Android] | Sets the connection string to be used to create or connect to a database.
setCreationString method [Android] | Sets the creation string to be used to create a database.
setDatabaseName method | Sets the database name.
setEncryptionKey method | Sets the key for encryption.
setLazyLoadIndexes method [BlackBerry] | Sets indexes to load as they are required, or to load all indexes at once on startup.
setPageSize method | Sets the page size of the database.
setPassword method | Sets the database password.
setRowScoreFlushSize method [BlackBerry] | Enables row limiting by specifying the score used to flush out old rows.
setRowScoreMaximum method [BlackBerry] | Sets the threshold for the maximum row score to retain in memory.
setUserName method [Android] | Sets the name of the user.

### Remarks
An object implementing the ConfigFile interface is created by using the DatabaseManager.createConfigurationFile method.

### See also
- “DatabaseManager class [UltraLiteJ]” on page 134
- “DatabaseManager.createConfigurationFile method [UltraLiteJ]” on page 136

### ConfigFileAndroid interface [Android]
Establishes a Configuration object for a persistent database saved in a file on an Android device.
Syntax

```
public interface ConfigFileAndroid
```

Base classes

- “ConfigFile interface [UltraLiteJ]” on page 85

Members

All members of the ConfigFileAndroid interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableAesDBEncryption method</td>
<td>Enables AES encryption of the database.</td>
</tr>
<tr>
<td>enableObfuscation method</td>
<td>Enables obfuscation of the database.</td>
</tr>
<tr>
<td>getCacheSize method</td>
<td>Returns the cache size of the database, in bytes.</td>
</tr>
<tr>
<td>getConnectionString method [Android]</td>
<td>Gets the connection string registered with the setConnectionString method.</td>
</tr>
<tr>
<td>getCreationString method [Android]</td>
<td>Gets the creation string registered with the setCreationString method.</td>
</tr>
<tr>
<td>getDatabaseName method</td>
<td>Returns the database name.</td>
</tr>
<tr>
<td>getEncryptionKey method</td>
<td>Gets the database encryption key that was registered with the setEncryptionKey method.</td>
</tr>
<tr>
<td>getLazyLoadIndexes method [BlackBerry]</td>
<td>Determines if lazy loading indexes is turned on.</td>
</tr>
<tr>
<td>getPageSize method</td>
<td>Returns the page size of the database, in bytes.</td>
</tr>
<tr>
<td>getRowScoreFlushSize method [BlackBerry]</td>
<td>Returns the current row score flush size.</td>
</tr>
<tr>
<td>getRowScoreMaximum method [BlackBerry]</td>
<td>Returns the current row score maximum size.</td>
</tr>
<tr>
<td>getUsername method [Android]</td>
<td>Gets the name of user set by the setUsername method.</td>
</tr>
<tr>
<td>hasShadowPaging method [BlackBerry]</td>
<td>Determines if shadow paging is turned on.</td>
</tr>
<tr>
<td>setCacheSize method</td>
<td>Sets the cache size of the database, in bytes.</td>
</tr>
<tr>
<td>setConnectionString method [Android]</td>
<td>Sets the connection string to be used to create or connect to a database.</td>
</tr>
</tbody>
</table>
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setCreationString method [Android]</code></td>
<td>Sets the creation string to be used to create a database.</td>
</tr>
<tr>
<td><code>setDatabaseName method</code></td>
<td>Sets the database name.</td>
</tr>
<tr>
<td><code>setEncryptionKey method</code></td>
<td>Sets the key for encryption.</td>
</tr>
<tr>
<td><code>setLazyLoadIndexes method [BlackBerry]</code></td>
<td>Sets indexes to load as they are required, or to load all indexes at once on startup.</td>
</tr>
<tr>
<td><code>setPageSize method</code></td>
<td>Sets the page size of the database.</td>
</tr>
<tr>
<td><code>setPassword method</code></td>
<td>Sets the database password.</td>
</tr>
<tr>
<td><code>setRowScoreFlushSize method [BlackBerry]</code></td>
<td>Enables row limiting by specifying the score used to flush out old rows.</td>
</tr>
<tr>
<td><code>setRowScoreMaximum method [BlackBerry]</code></td>
<td>Sets the threshold for the maximum row score to retain in memory.</td>
</tr>
<tr>
<td><code>setUserName method [Android]</code></td>
<td>Sets the name of the user.</td>
</tr>
</tbody>
</table>

### Remarks

An object implementing the ConfigFileAndroid interface is created by using the `DatabaseManager.createConfigurationFileAndroid` method.

### See also

- “DatabaseManager class [UltraLiteJ]” on page 134
- “DatabaseManager.createConfigurationFileAndroid method [UltraLiteJ]” on page 137

### ConfigNonPersistent interface [BlackBerry]

Establishes a Configuration object for a non-persistent (in-memory) database.

### Syntax

```
public interface ConfigNonPersistent
```

### Base classes

- “Configuration interface [UltraLiteJ]” on page 100

### Members

All members of the ConfigNonPersistent interface, including all inherited members.
### getDatabaseName method
Returns the database name.

### setPageSize method
Returns the page size of the database, in bytes.

### setDatabaseName method
Sets the database name.

### setPageSize method
Sets the page size of the database.

### setPassword method
Sets the database password.

**Remarks**

An object implementing the ConfigNonPersistent interface is created by using the DatabaseManager.createConfigurationNonPersistent method.

Creating a NonPersistent object configures a database store that only exists in memory. The database is created at startup, used while the application is running, then discarded when the application closes. When the application closes, all data contained in the non-persistent store is deleted.

**See also**
- “DatabaseManager class [UltraLiteJ]” on page 134

### ConfigObjectStore interface [BlackBerry]
Establishes a Configuration object for a persistent database saved in an object store.

**Syntax**

```java
public interface ConfigObjectStore
```

**Base classes**
- “ConfigPersistent interface [UltraLiteJ]” on page 91

**Members**

All members of the ConfigObjectStore interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableAesDBEncryption method</td>
<td>Enables AES encryption of the database.</td>
</tr>
<tr>
<td>enableObfuscation method</td>
<td>Enables obfuscation of the database.</td>
</tr>
<tr>
<td>getCacheSize method</td>
<td>Returns the cache size of the database, in bytes.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getConnectionString method [Android]</td>
<td>Gets the connection string registered with the setConnectionString method.</td>
</tr>
<tr>
<td>getCreationString method [Android]</td>
<td>Gets the creation string registered with the setCreationString method.</td>
</tr>
<tr>
<td>getDatabaseName method</td>
<td>Returns the database name.</td>
</tr>
<tr>
<td>getEncryptionKey method</td>
<td>Gets the database encryption key that was registered with the setEncryptionKey method.</td>
</tr>
<tr>
<td>getLazyLoadIndexes method [BlackBerry]</td>
<td>Determines if lazy loading indexes is turned on.</td>
</tr>
<tr>
<td>getPageSize method</td>
<td>Returns the page size of the database, in bytes.</td>
</tr>
<tr>
<td>getRowScoreFlushSize method [BlackBerry]</td>
<td>Returns the current row score flush size.</td>
</tr>
<tr>
<td>getRowScoreMaximum method [BlackBerry]</td>
<td>Returns the current row score maximum size.</td>
</tr>
<tr>
<td>getUsername method [Android]</td>
<td>Gets the name of user set by the setUsername method.</td>
</tr>
<tr>
<td>hasShadowPaging method [BlackBerry]</td>
<td>Determines if shadow paging is turned on.</td>
</tr>
<tr>
<td>setCacheSize method</td>
<td>Sets the cache size of the database, in bytes.</td>
</tr>
<tr>
<td>setConnectionString method [Android]</td>
<td>Sets the connection string to be used to create or connect to a database.</td>
</tr>
<tr>
<td>setCreationString method [Android]</td>
<td>Sets the creation string to be used to create a database.</td>
</tr>
<tr>
<td>setDatabaseName method</td>
<td>Sets the database name.</td>
</tr>
<tr>
<td>setEncryptionKey method</td>
<td>Sets the key for encryption.</td>
</tr>
<tr>
<td>setLazyLoadIndexes method [BlackBerry]</td>
<td>Sets indexes to load as they are required, or to load all indexes at once on startup.</td>
</tr>
<tr>
<td>setPageSize method</td>
<td>Sets the page size of the database.</td>
</tr>
<tr>
<td>setPassword method</td>
<td>Sets the database password.</td>
</tr>
<tr>
<td>setRowScoreFlushSize method [BlackBerry]</td>
<td>Enables row limiting by specifying the score used to flush out old rows.</td>
</tr>
</tbody>
</table>
setRowScoreMaximum method [BlackBerry]
Sets the threshold for the maximum row score to retain in memory.

setUserName method [Android]
Sets the name of the user.

Remarks
An object implementing the ConfigObjectStore interface is created by using the DatabaseManager.createConfigurationObjectStore method.

See also
● “DatabaseManager class [UltraLiteJ]” on page 134
● “DatabaseManager.createConfigurationObjectStore method [BlackBerry] [UltraLiteJ]” on page 138

ConfigPersistent interface
Establishes a Configuration object for a persistent database.

Syntax
public interface ConfigPersistent

Base classes
● “Configuration interface [UltraLiteJ]” on page 100

Derived classes
● “ConfigFile interface [UltraLiteJ]” on page 85
● “ConfigObjectStore interface [BlackBerry] [UltraLiteJ]” on page 89

Members
All members of the ConfigPersistent interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableAesDBEncryption method</td>
<td>Enables AES encryption of the database.</td>
</tr>
<tr>
<td>enableObfuscation method</td>
<td>Enables obfuscation of the database.</td>
</tr>
<tr>
<td>getCachSize method</td>
<td>Returns the cache size of the database, in bytes.</td>
</tr>
<tr>
<td>getConnectionString method [Android]</td>
<td>Gets the connection string registered with the setConnectionString method.</td>
</tr>
<tr>
<td>getCreationString method [Android]</td>
<td>Gets the creation string registered with the setCreationString method.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getDatabaseName method</td>
<td>Returns the database name.</td>
</tr>
<tr>
<td>getEncryptionKey method</td>
<td>Gets the database encryption key that was registered with the setEncryptionKey method.</td>
</tr>
<tr>
<td>getLazyLoadIndexes method [BlackBerry]</td>
<td>Determines if lazy loading indexes is turned on.</td>
</tr>
<tr>
<td>getPageSize method</td>
<td>Returns the page size of the database, in bytes.</td>
</tr>
<tr>
<td>getRowScoreFlushSize method [BlackBerry]</td>
<td>Returns the current row score flush size.</td>
</tr>
<tr>
<td>getRowScoreMaximum method [BlackBerry]</td>
<td>Returns the current row score maximum size.</td>
</tr>
<tr>
<td>getUserName method [Android]</td>
<td>Gets the name of user set by the setUserName method.</td>
</tr>
<tr>
<td>hasShadowPaging method [BlackBerry]</td>
<td>Determines if shadow paging is turned on.</td>
</tr>
<tr>
<td>setCacheSize method</td>
<td>Sets the cache size of the database, in bytes.</td>
</tr>
<tr>
<td>setConnectionString method [Android]</td>
<td>Sets the connection string to be used to create or connect to a database.</td>
</tr>
<tr>
<td>setCreationString method [Android]</td>
<td>Sets the creation string to be used to create a database.</td>
</tr>
<tr>
<td>setDatabaseName method</td>
<td>Sets the database name.</td>
</tr>
<tr>
<td>setEncryptionKey method</td>
<td>Sets the key for encryption.</td>
</tr>
<tr>
<td>setLazyLoadIndexes method [BlackBerry]</td>
<td>Sets indexes to load as they are required, or to load all indexes at once on startup.</td>
</tr>
<tr>
<td>setPageSize method</td>
<td>Sets the page size of the database.</td>
</tr>
<tr>
<td>setPassword method</td>
<td>Sets the database password.</td>
</tr>
<tr>
<td>setRowScoreFlushSize method [BlackBerry]</td>
<td>Enables row limiting by specifying the score used to flush out old rows.</td>
</tr>
<tr>
<td>setRowScoreMaximum method [BlackBerry]</td>
<td>Sets the threshold for the maximum row score to retain in memory.</td>
</tr>
<tr>
<td>setUserName method [Android]</td>
<td>Sets the name of the user.</td>
</tr>
</tbody>
</table>
Remarks
By default, a database uses a shadow paging persistent store which is updated when transactions are committed, or when indexes, or cached pages are paged out of memory.

Options such as lazy loading, row score flush size and row score maximum only apply to shadow paging persistent databases.

enableAesDBEncryption method
Enables AES encryption of the database.

Syntax
void ConfigPersistent.enableAesDBEncryption()

Remarks
Specify the DBKEY connection parameter when creating or connecting to the database, or use the setEncryptionKey method.

See also
● “ConfigPersistent.setEncryptionKey method [UltraLiteJ]” on page 97
● “UltraLite DBKEY connection parameter” [UltraLite - Database Management and Reference]

enableObfuscation method
Enables obfuscation of the database.

Syntax
void ConfigPersistent.enableObfuscation()

cacheSize method
Returns the cache size of the database, in bytes.

Syntax
int ConfigPersistent.getCacheSize()

Returns
The cache size.

See also
● “ConfigPersistent.setCacheSize method [UltraLiteJ]” on page 96
getConnectionString method [Android]

Gets the connection string registered with the setConnectionString method.

Syntax

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

Returns

The connection string registered with the setConnectionString method.

See also

- “ConfigPersistent.setConnectionString method [Android] [UltraLiteJ]” on page 96

getCreationString method [Android]

Gets the creation string registered with the setCreationString method.

Syntax

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

Returns

The creation string registered with the setCreationString method.

See also

- “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97

getEncryptionKey method

Gets the database encryption key that was registered with the setEncryptionKey method.

Syntax

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

Returns

The database encryption key that was registered with the setEncryptionKey method.

See also

- “ConfigPersistent.setEncryptionKey method [UltraLiteJ]” on page 97
- “Connection.changeEncryptionKey method [UltraLiteJ]” on page 107

getLazyLoadIndexes method [BlackBerry]

Determines if lazy loading indexes is turned on.
Syntax

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

**Returns**

True if lazy loading is turned on; otherwise, returns false.

**See also**
- “ConfigPersistent.setLazyLoadIndexes method [BlackBerry] [UltraLiteJ]” on page 97

---

**getRowScoreFlushSize method [BlackBerry]**

Returns the current row score flush size.

Syntax

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

**Returns**

The current row score flush size.

**See also**
- “ConfigPersistent.setRowScoreFlushSize method [BlackBerry] [UltraLiteJ]” on page 98

---

**getRowScoreMaximum method [BlackBerry]**

Returns the current row score maximum size.

Syntax

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

**Returns**

The current row score maximum size.

**See also**
- “ConfigPersistent.setRowScoreMaximum method [BlackBerry] [UltraLiteJ]” on page 99

---

**getUserName method [Android]**

Gets the name of user set by the setUserName method.

Syntax

```java
String ConfigPersistent.getUserName()
```
Returns
The name of user set by the setUserName method.

See also
● “ConfigPersistent.setUserName method [Android] [UltraLiteJ]” on page 99

**hasShadowPaging method [BlackBerry]**
Determines if shadow paging is turned on.

Syntax
```java
boolean ConfigPersistent.hasShadowPaging()
```

Returns
True if shadow paging is turned on; otherwise, returns false.

**setCacheSize method**
Sets the cache size of the database, in bytes.

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

Parameters
● `cache_size` The cache size. The default cache size is 20480 (20KB) on all platforms.

Returns
This ConfigPersistent object with the cache size specified.

Remarks
The cache size determines the number of database pages resident in the page cache. Increasing the size means less reading and writing of database pages, at the expense of increased time to locate pages in the cache.

See also
● “ConfigPersistent.getCacheSize method [UltraLiteJ]” on page 93

**setConnectionString method [Android]**
Sets the connection string to be used to create or connect to a database.
Syntax

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

Parameters

- **connection_string**  The connection string used in database connection or creation.

Remarks

Any other items that have been set in this configuration are also passed to create or connect to a database.

### setCreationString method [Android]

Sets the creation string to be used to create a database.

Syntax

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

Parameters

- **creation_string**  The creation string used in database creation.

Remarks

Any other items that have been set in this configuration are also passed to create a database.

### setEncryptionKey method

Sets the key for encryption.

Syntax

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

Parameters

- **encryption_key**  The string to use for the encryption key.

See also

- “ConfigPersistent.getEncryptionKey method [UltraLiteJ]” on page 94
- “ConfigPersistent.enableAesDBEncryption method [UltraLiteJ]” on page 93
- “Connection.changeEncryptionKey method [UltraLiteJ]” on page 107

### setLazyLoadIndexes method [BlackBerry]

Sets indexes to load as they are required, or to load all indexes at once on startup.
Syntax

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

Parameters

- `lazy_load` Set true so that indexes load as required; otherwise, set false to load all indexes at once on startup.

Returns

This ConfigPersistent object with the index loading schema specified.

Remarks

Enabling this option reduces the startup time of the database but future operations may perform slower.

Turning off lazy loading also turns off row limiting by setting the row score flush size to zero.

See also

- “ConfigPersistent.getLazyLoadIndexes method [BlackBerry] [UltraLiteJ]” on page 94
- “ConfigPersistent.setRowScoreFlushSize method [BlackBerry] [UltraLiteJ]” on page 98

**setRowScoreFlushSize method [BlackBerry]**

Enables row limiting by specifying the score used to flush out old rows.

Syntax

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

Parameters

- `flushSize` The row score value used to determine how many rows to flush. The default is 0, which indicates no row limiting.

Returns

This ConfigPersistent object with the flush size value specified.

Remarks

Row score is a measure of the references used to maintain recently used rows in memory. Each row in memory is assigned a score based on the number and types of columns they have, which approximates the maximum number of references they could use.

Most columns score as 1; varchar binary, long binary, and UUID score as 2; long varchar scores as 4.

When the maximum score threshold is reached, the flush size is used to determine how many old rows to remove.
Keep the flush size (measured as a row score) reasonable (less than 1000) to prevent large interruptions.

Databases accessed with row limiting enabled always lazy load indexes. Turning off lazy loading also turns off row limiting by setting the row score flush size to zero.

See also

- “ConfigPersistent.setRowScoreMaximum method [BlackBerry] [UltraLiteJ]” on page 99
- “ConfigPersistent.setLazyLoadIndexes method [BlackBerry] [UltraLiteJ]” on page 97

setRowScoreMaximum method [BlackBerry]

Sets the threshold for the maximum row score to retain in memory.

Syntax

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

Parameters

- **threshold**  
  The maximum threshold value. The maximum value is 200,000. The default value is 50,000.

Returns

This ConfigPersistent object with the maximum threshold value specified.

Remarks

Databases accessed with row limiting enabled always lazy load indexes.

See also

- “ConfigPersistent.setRowScoreFlushSize method [BlackBerry] [UltraLiteJ]” on page 98
- “ConfigPersistent.setLazyLoadIndexes method [BlackBerry] [UltraLiteJ]” on page 97

setUserName method [Android]

Sets the name of the user.

Syntax

```java
void setUserName(String user_name)
```

Parameters

- **user_name**  
  The name of the user.
Remarks
This name is used to connect or create to the native database with the UID= phrase in the connection string.

Configuration interface
Establishes a Configuration object for a database.

Syntax
public interface Configuration

Derived classes
- “ConfigNonPersistent interface [BlackBerry] [UltraLiteJ]” on page 88
- “ConfigPersistent interface [UltraLiteJ]” on page 91

Members
All members of the Configuration interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDatabaseName method</td>
<td>Returns the database name.</td>
</tr>
<tr>
<td>getPageSize method</td>
<td>Returns the page size of the database, in bytes.</td>
</tr>
<tr>
<td>setDatabaseName method</td>
<td>Sets the database name.</td>
</tr>
<tr>
<td>setPageSize method</td>
<td>Sets the page size of the database.</td>
</tr>
<tr>
<td>setPassword method</td>
<td>Sets the database password.</td>
</tr>
</tbody>
</table>

Remarks
Some attributes are used only during database creation while others apply to the initial connection to a database. Attributes are ignored if they are set after creating a database, or connecting to a database.

getDatabaseName method
Returns the database name.

Syntax
String Configuration.getDatabaseName()

Returns
The name of the database.
getPageSize method

Returns the page size of the database, in bytes.

Syntax

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

Returns

The page size.

setDatabaseName method

Sets the database name.

Syntax

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

Parameters

- `db_name` The name of database.

Returns

This Configuration object with the database name specified.

setPageSize method

Sets the page size of the database.

Syntax

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

Parameters

- `page_size` The page size, in bytes.

Returns

This Configuration object with the page size specified.

Remarks

The page size setting is used to determine the maximum size of a row stored in a persistent database. It establishes the size of an index page, and determines the number of children that each page can have.
When using an existing database, the size is already set to the page size of the database when it was created. You can not reset the page size of an existing database using this method.

For Android smartphones, the page size can be 1024, 2048, 4096, 8192, or 16384 bytes. The default is 4096 bytes.

For BlackBerry smartphones, the page size can range from 256 to 16384 bytes. The default is 1024 bytes. The page size is always adjusted to be a multiple of 32.

**setPassword method**

Sets the database password.

**Syntax**

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

**Parameters**

- **password**  A password for a new database, or the password to gain access to an existing database.

**Returns**

This Configuration object with the database password set.

**Remarks**

The password is used to gain access to the database, and must match the password specified when the database was created. The default is "dba".

**Connection interface**

Describes a database connection, which is required to initiate database operations.

**Syntax**

```java
public interface Connection
```

**Members**

All members of the Connection interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancelWaitForEvent method [Android]</td>
<td>Cancels any waitForEvent calls on this Connection object.</td>
</tr>
<tr>
<td>changeEncryptionKey method</td>
<td>Changes the database encryption key for an UltraLite database.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>commit method</td>
<td>Commits the database changes.</td>
</tr>
<tr>
<td>createDecimalNumber method</td>
<td>Creates a new DecimalNumber object.</td>
</tr>
<tr>
<td>createSyncParms method</td>
<td>Creates a set of synchronization parameters.</td>
</tr>
<tr>
<td>createUUIDValue method</td>
<td>Creates a UUID value.</td>
</tr>
<tr>
<td>dropDatabase method</td>
<td>Drops a database.</td>
</tr>
<tr>
<td>emergencyShutdown method [BlackBerry]</td>
<td>Performs an emergency shut down of a connected database.</td>
</tr>
<tr>
<td>getDatabaseId method [BlackBerry]</td>
<td>Returns the value of database ID.</td>
</tr>
<tr>
<td>getDatabaseInfo method</td>
<td>Returns a DataInfo object containing information about database properties.</td>
</tr>
<tr>
<td>getDatabaseProperty method</td>
<td>Returns a database property.</td>
</tr>
<tr>
<td>getLastDownloadTime method</td>
<td>Returns the time of the most recent download of the specified publication.</td>
</tr>
<tr>
<td>getLastIdentity method</td>
<td>Retrieves the most recent value inserted into a DEFAULT AUTOINCREMENT or DE-</td>
</tr>
<tr>
<td></td>
<td>FAULT GLOBAL AUTOINCREMENT column, or zero if the most recent INSERT trans-</td>
</tr>
<tr>
<td></td>
<td>action was made on a table that had no such column.</td>
</tr>
<tr>
<td>getLastWarning method</td>
<td>Returns information about the last SQL statement executed on this connection.</td>
</tr>
<tr>
<td>getOption method [BlackBerry]</td>
<td>Returns a database option.</td>
</tr>
<tr>
<td>getState method</td>
<td>Returns the state of the connection.</td>
</tr>
<tr>
<td>getSyncObserver method</td>
<td>Returns the SyncObserver object that is currently registered for this Conn-</td>
</tr>
<tr>
<td></td>
<td>ection object.</td>
</tr>
<tr>
<td>getSyncResult method</td>
<td>Returns the result of the last SYNCHRONIZE SQL statement.</td>
</tr>
<tr>
<td>isSynchronizationDeleteDisabled method [BlackBerr-</td>
<td></td>
</tr>
<tr>
<td>ry]</td>
<td>Determine if the synchronization of deletes is disabled.</td>
</tr>
<tr>
<td>prepareStatement method</td>
<td>Prepares a statement for execution.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>registerForEvent method [Android]</td>
<td>Registers a system event to receive notifications.</td>
</tr>
<tr>
<td>release method</td>
<td>Releases this connection.</td>
</tr>
<tr>
<td>resetLastDownloadTime method</td>
<td>Resets the time of the download for the specified publications.</td>
</tr>
<tr>
<td>rollback method</td>
<td>Commits a rollback to undo changes to the database.</td>
</tr>
<tr>
<td>rollbackPartialDownload method [Android]</td>
<td>Rolls back the changes from a failed synchronization.</td>
</tr>
<tr>
<td>setDatabaseId method</td>
<td>Sets the database ID for global autoincrement.</td>
</tr>
<tr>
<td>setOption method</td>
<td>Sets the database option.</td>
</tr>
<tr>
<td>setSyncObserver method</td>
<td>Sets a SyncObserver object to monitor the progress of synchronizations on this connection.</td>
</tr>
<tr>
<td>synchronize method</td>
<td>Synchronizes the database with a MobiLink server.</td>
</tr>
<tr>
<td>unregisterForEvent method [Android]</td>
<td>Unregisters from a system event to stop receiving notifications.</td>
</tr>
<tr>
<td>validateDatabase method [Android]</td>
<td>Validates the database on this connection.</td>
</tr>
<tr>
<td>CONNECTED variable</td>
<td>Denotes a connected state.</td>
</tr>
<tr>
<td>NOT_CONNECTED variable</td>
<td>Denotes a not connected state.</td>
</tr>
<tr>
<td>OPTION_DATE_FORMAT variable</td>
<td>Database option: date format.</td>
</tr>
<tr>
<td>OPTION_DATE_ORDER variable</td>
<td>Database option: date order.</td>
</tr>
<tr>
<td>OPTION_MAX_HASH_SIZE variable</td>
<td>Database option: maximum hash size.</td>
</tr>
<tr>
<td>OPTION_ML_REMOTE_ID variable [BlackBerry]</td>
<td>Database option: ML remote ID.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OPTION_ML_SERVER_VERSION variable</td>
<td>Database option: MobiLink server protocol version.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td></td>
</tr>
<tr>
<td>OPTION_NEAREST_CENTURY variable</td>
<td>Database option: nearest century.</td>
</tr>
<tr>
<td>OPTION_PRECISION variable</td>
<td>Database option: precision.</td>
</tr>
<tr>
<td>OPTION_SCALE variable</td>
<td>Database option: scale.</td>
</tr>
<tr>
<td>OPTION_TIME_FORMAT variable</td>
<td>Database option: time format.</td>
</tr>
<tr>
<td>OPTION_TIMESTAMP_FORMAT variable</td>
<td>Database option: timestamp format.</td>
</tr>
<tr>
<td>OPTION_TIMESTAMP_INCREMENT variable</td>
<td>Database option: timestamp increment.</td>
</tr>
<tr>
<td>OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT variable</td>
<td>Database option: timestamp with time zone format.</td>
</tr>
<tr>
<td>PROPERTY_DATABASE_NAME variable</td>
<td>Database Property: database name.</td>
</tr>
<tr>
<td>PROPERTY_PAGE_SIZE variable</td>
<td>Database Property: page size.</td>
</tr>
<tr>
<td>SYNC_ALL variable</td>
<td>The publication list used to request synchronization of all tables in the database, including tables not in any publication.</td>
</tr>
<tr>
<td>SYNC_ALL_DB_PUB_NAME variable</td>
<td>The reserved name of the SYNC_ALL_DB publication.</td>
</tr>
<tr>
<td>SYNC_ALL_PUBS variable</td>
<td>The publication list used to request synchronization of all publications in the database.</td>
</tr>
<tr>
<td>ULVF_DATABASE variable [Android]</td>
<td>Used to validate database.</td>
</tr>
<tr>
<td>ULVF_EXPRESS variable [Android]</td>
<td>Used to perform a faster, though less thorough, validation.</td>
</tr>
<tr>
<td>ULVF_FULL_VALIDATE variable [Android]</td>
<td>Performs all types of validation on the database.</td>
</tr>
<tr>
<td>ULVF_INDEX variable [Android]</td>
<td>Used to validate indexes.</td>
</tr>
<tr>
<td>ULVF_TABLE variable [Android]</td>
<td>Used to validate table(s).</td>
</tr>
</tbody>
</table>
Remarks

A connection is obtained using the connect or createDatabase methods of the DatabaseManager class. Use the release method when the connection is no longer needed. When all connections for a database are released, the database is closed.

A Connection object provides the following capabilities:

- Create new schema (tables, indexes and publications)
- Create new value and domain objects
- Permanently commit changes to the database
- Prepare SQL statements for execution
- Roll back uncommitted changes to the database

The following example demonstrates how to create a schema for a simple database with a Connection object, conn, created for it. The database contains a table named T1, which has a single integer primary key column named num, and a table named T2, which has an integer primary key column named num and an integer column named quantity. T2 has an addition index on quantity. A publication named PubA contains T1.

```java
// 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.executeUpdate();
ps.close();

ps = conn.prepareStatement( "CREATE TABLE T2 ( num INT NOT NULL PRIMARY KEY, quantity INT )" );
ps.executeUpdate();
ps.close();

ps = conn.prepareStatement( "CREATE INDEX index1 ON T2( quantity )" );
ps.executeUpdate();
ps.close();

ps = conn.prepareStatement( "CREATE Publication PubA ( Table T1 )" );
ps.executeUpdate();
ps.close();
```

See also

- “DatabaseManager class [UltraLiteJ]” on page 134
- “DatabaseManager.createDatabase method [UltraLiteJ]” on page 138
- “DatabaseManager.connect method [UltraLiteJ]” on page 136
- “Connection.release method [UltraLiteJ]” on page 116

**cancelWaitForEvent method [Android]**

Cancels any waitForEvent calls on this Connection object.
Syntax
```java
void Connection.cancelWaitForEvent() throws ULjException
```

See also
- “Connection.waitForEvent method [Android] [UltraLiteJ]” on page 121

### changeEncryptionKey method
Changes the database encryption key for an UltraLite database.

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

Parameters
- `newKey` The new encryption key for the database.

Remarks
Applications that call this method must first ensure that the user has either synchronized the database or created a reliable backup copy of the database. It is important to have a reliable backup of the database because the changeEncryptionKey method is an operation that must run to completion. When the database encryption key is changed, every row in the database is first decrypted with the old key and then encrypted with the new key and rewritten. This operation is not recoverable. If the encryption change operation does not complete, the database is left in an invalid state and you cannot access it again.

### commit method
Commits the database changes.

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

Remarks
Invoking this method causes all table data changes since the last commit or rollback to become permanent.

### createDecimalNumber method
Creates a new DecimalNumber object.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createDecimalNumber(int, int) method</td>
<td>Creates a DecimalNumber object.</td>
</tr>
</tbody>
</table>
createDecimalNumber(int, int) method

Creates a DecimalNumber object.

Syntax

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

Parameters

- **precision**  The number of digits in the number.
- **scale**     The number of decimal places in the number.

Returns

The DecimalNumber object with the specified type.

See also

- “DecimalNumber interface [UltraLiteJ]” on page 142

createDecimalNumber(int, int, String) method

Creates a DecimalNumber object.

Syntax

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

Parameters

- **precision**  The number of digits in the number.
- **scale**     The number of decimal places in the number.
- **value**     The value to be set.

Returns

The DecimalNumber object with the specified type.
See also

- “DecimalNumber interface [UltraLiteJ]” on page 142

**createSyncParms method**

Creates a set of synchronization parameters.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>createSyncParms(int, String, String)</code> method</td>
<td>Creates a set of synchronization parameters for HTTP synchronization.</td>
</tr>
<tr>
<td><code>createSyncParms(String, String)</code> method</td>
<td>Creates a set of synchronization parameters for HTTP synchronization.</td>
</tr>
</tbody>
</table>

**createSyncParms(int, String, String) method**

Creates a set of synchronization parameters for HTTP synchronization.

**Syntax**

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

**Parameters**

- **streamType** One of the constants defined in the SyncParms class used to identify the type of synchronization stream.
- **userName** The MobiLink user name.
- **version** The MobiLink script version.

**Returns**

A SyncParms object.

**See also**

- “Connection.createSyncParms method [UltraLiteJ]” on page 109
- “SyncParms.HTTP_STREAM variable [UltraLiteJ]” on page 260
- “SyncParms.HTTPS_STREAM variable [UltraLiteJ]” on page 260
createSyncParms(String, String) method

Creates a set of synchronization parameters for HTTP synchronization.

Syntax

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

Parameters

● userName  The unique MobiLink user name for this client database.

● version    The MobiLink script version.

Returns

The SyncParms object.

See also

● “Connection.createSyncParms method [UltraLiteJ]” on page 109
● “SyncParms.setUserName method [UltraLiteJ]” on page 258

createUUIDValue method

Creates a UUID value.

Syntax

UUIDValue Connection.createUUIDValue () throws ULjException

Returns

The UUIDValue instance for the domain.

dropDatabase method

Drops a database.

Syntax

void Connection.dropDatabase () throws ULjException

Remarks

The database referenced by the connection is erased and the connection is released. This connection is the only one that can be active for the database being dropped.
**emergencyShutdown method [BlackBerry]**

Performs an emergency shut down of a connected database.

**Syntax**

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

**Remarks**

This method should only be invoked in severe error situations. It should only be used if physical hardware or data is destroyed.

The method closes all open connections and shuts down the connected database.

---

**getDatabaseId method [BlackBerry]**

Returns the value of database ID.

**Syntax**

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

**Returns**

The database ID.

**Exceptions**

- **ULjException class** when the database ID is not set.

**See also**

- “Connection.getLastIdentity method [UltraLiteJ]” on page 112

---

**getDatabaseInfo method**

Returns a DatabaseInfo object containing information about database properties.

**Syntax**

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

**Returns**

The DatabaseInfo object.

---

**getDatabaseProperty method**

Returns a database property.
Syntax

String `Connection.getDatabaseProperty(String name)` throws ULjException

Parameters

- **name**  The name of the database property. For Android devices, you can set this parameter to any supported UltraLite database property name. For BlackBerry devices, you can set this parameter to any constant in the Connection interface that has a `PROPERTY_` prefix.

Returns

The value of the property that corresponds to the given name.

See also

- “Connection.PROPERTY_DATABASE_NAME variable [UltraLiteJ]” on page 127
- “Connection.PROPERTY_PAGE_SIZE variable [UltraLiteJ]” on page 128
- “UltraLite database properties” [UltraLite - Database Management and Reference]

getDownloadTime method

Returns the time of the most recent download of the specified publication.

Syntax

Date `Connection.getLastDownloadTime(String pub_name)` throws ULjException

Parameters

- **pub_name**  The name of the publication to check. This parameter must reference a single publication or be the special Connection.SYNC_ALL_DB_PUB_NAME publication for the time of the last download of the full database.

Returns

The timestamp of the last download.

See also

- “Connection.SYNC_ALL_DB_PUB_NAME variable [UltraLiteJ]” on page 128
- “Connection.resetLastDownloadTime method [UltraLiteJ]” on page 116

generate method

Retrieves the most recent value inserted into a DEFAULT AUTOINCREMENT or DEFAULT GLOBAL AUTOINCREMENT column, or zero if the most recent INSERT transaction was made on a table that had no such column.

Syntax

long `Connection.getLastIdentity()`
Returns
The most recent identity value.

Remarks
If a table contains more than one column of the (GLOBAL) AUTOINCREMENT type, then the column to which this value belongs to is undetermined.

getLastWarning method
Returns information about the last SQL statement executed on this connection.

Syntax
```
SQLInfo Connection.getLastWarning()
```

Returns
The SQLInfo object for the last SQL statement executed.

getOption method [BlackBerry]
Returns a database option.

Syntax
```
String Connection.getOption(String option_name) throws ULjException
```

Parameters
- **option_name**  The name of option to get.

- **option_name**  The name of the option to get. You can set this parameter to any constant in the Connection interface that has an OPTION_ prefix.

Returns
The value of the database option.

Remarks
Database options are stored within the database and may also be obtained when a database is connected at some time after the option has been set.

A set of required options are created when a database is created.

See also
- “Connection.setOption method [UltraLiteJ]” on page 118
**getState method**

Returns the state of the connection.

**Syntax**

```java
byte Connection.getState() throws ULjException
```

**Returns**

The byte representing the state of the connection.

**Remarks**

The following example demonstrates how to check the connection state and release the connection.

```java
if( _conn.getState() == Connection.CONNECTED ){
  _conn.release();
}
```

**See also**

- “Connection interface [UltraLiteJ]” on page 102

**getSyncObserver method**

Returns the SyncObserver object that is currently registered for this Connection object.

**Syntax**

```java
SyncObserver Connection.getSyncObserver()
```

**Returns**

The SyncObserver, or null if no observer exists.

**See also**

- “Connection.setSyncObserver method [UltraLiteJ]” on page 119

**getSyncResult method**

Returns the result of the last SYNCHRONIZE SQL statement.

**Syntax**

```java
SyncResult Connection.getSyncResult()
```

**Returns**

The SyncResult object representing the result of the last SYNCHRONIZE SQL statement.

**Remarks**

The following example illustrates how to get the result of the last SYNCHRONIZE SQL statement:
PreparedStatement ps = conn.prepareStatement("SYNCHRONIZE PROFILE myprofile");
ps.executeUpdate();
ps.close();
SyncResult result = conn.getSyncResult();
display(
   "*** Synchronized *** sent=" + result.getSentRowCount() + ", received=" + result.getReceivedRowCount() + ");

Note
This method does not return the result of the last call to the Connection.synchronize method. To obtain the SyncResult object for the last Connection.synchronize(SyncParms) method call, use the getSyncResult method on the SyncParms object passed in.

See also
- “SyncResult class [UltraLiteJ]” on page 260
- “SyncParms.getSyncResult method [UltraLiteJ]” on page 249

**isSynchronizationDeleteDisabled method [BlackBerry]**
Determine if the synchronization of deletes is disabled.

**Syntax**

```java
boolean Connection.isSynchronizationDeleteDisabled()
```

**Returns**
True if and only if the synchronization of deletes is disabled.

**prepareStatement method**
Prepares a statement for execution.

**Syntax**

```java
PreparedStatement Connection.prepareStatement(
   String sql
) throws ULjException
```

**Parameters**
- `sql` A SQL statement to prepare.

**Returns**
A PreparedStatement object.

**See also**
- “PreparedStatement interface [UltraLiteJ]” on page 175
**registerForEvent method [Android]**

Registers a system event to receive notifications.

**Syntax**

```java
void Connection.registerForEvent(
    short event_type,
    String object_name
) throws ULjException
```

**Parameters**

- **event_type** The type of event to register for.
- **object_name** The object to which the event applies, such as a table name.

**See also**

- “ULjEvent interface [Android] [UltraLiteJ]” on page 276

**release method**

Releases this connection.

**Syntax**

```java
void Connection.release() throws ULjException
```

**Remarks**

Once a connection has been released, it can no longer be used to access the database.

It is an error release a connection for which there exist uncommitted transactions.

**resetLastDownloadTime method**

Resets the time of the download for the specified publications.

**Syntax**

```java
void Connection.resetLastDownloadTime(
    String pub_name
) throws ULjException
```

**Parameters**

- **pub_name** The name of the publication to check.

**Remarks**

To reset the download time for when the entire database is synchronized, use the special Connection.SYNC_ALL_DB_PUB_NAME publication.
This method requires that no uncommitted transactions are on the current connection.

**rollback method**

Commits a rollback to undo changes to the database.

**Syntax**

```java
void Connection.rollback() throws ULjException
```

**Remarks**

Invoking this method undoes all table data changes on this Connection object since the last commit or rollback.

**rollbackPartialDownload method [Android]**

Rolls back the changes from a failed synchronization.

**Syntax**

```java
void Connection.rollbackPartialDownload() throws ULjException
```

**Remarks**

This method only affects resumable downloads. (synchronizing with SyncParms.setKeepPartialDownload set to true)

If a communication error occurs during the download phase of synchronization while the KeepPartialDownload parameter is true, the downloaded changes are retained so that synchronization can resume from the place where the download is interrupted.

This method discards the partial download when you no longer want to resume the download.

**See also**

- “SyncParms.setKeepPartialDownload method [Android] [UltraLiteJ]” on page 253

**setDatabaseId method**

Sets the database ID for global autoincrement.

**Syntax**

```java
void Connection.setDatabaseId(int id) throws ULjException
```

**Parameters**

- **id** The database ID.
Remarks

The database ID does not have a default value.

During an INSERT, GLOBAL AUTOINCREMENT columns have NULL values inserted unless a database ID has been set explicitly.

See also

- “Connection.getDatabaseId method [BlackBerry] [UltraLiteJ]” on page 111

**setOption method**

Sets the database option.

Syntax

```java
void Connection.setOption(
    String option_name,
    String option_value
) throws ULjException
```

Parameters

- **option_name**  The name of the option to set. For Android devices, you can set this parameter to any supported UltraLite database option name. For BlackBerry devices, you can set this parameter to any constant in the Connection interface that has an `OPTION_` prefix.

- **option_value**  The new value of the option.

Remarks

If the option is not currently stored on the database, it is created.

There cannot be any uncommitted transactions for this connection when the method is invoked.

See also

- “Connection.getOption method [BlackBerry] [UltraLiteJ]” on page 113
- “Connection.OPTION_BLOB_FILE_BASE_DIR variable [BlackBerry] [UltraLiteJ]” on page 121
- “Connection.OPTION_DATABASE_ID variable [BlackBerry] [UltraLiteJ]” on page 122
- “Connection.OPTION_DATE_FORMAT variable [UltraLiteJ]” on page 122
- “Connection.OPTION_DATE_ORDER variable [UltraLiteJ]” on page 122
- “Connection.OPTION_ML_REMOTE_ID variable [BlackBerry] [UltraLiteJ]” on page 123
- “Connection.OPTION_NEAREST_CENTURY variable [UltraLiteJ]” on page 124
- “Connection.OPTION_PRECISION variable [UltraLiteJ]” on page 124
- “Connection.OPTION_SCALE variable [UltraLiteJ]” on page 125
- “Connection.OPTION_TIME_FORMAT variable [UltraLiteJ]” on page 125
- “Connection.OPTION_TIMESTAMP_FORMAT variable [UltraLiteJ]” on page 126
- “Connection.OPTION_TIMESTAMP_INCREMENT variable [UltraLiteJ]” on page 126
- “Connection.OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT variable [UltraLiteJ]” on page 127
**setSyncObserver method**

Sets a SyncObserver object to monitor the progress of synchronizations on this connection.

**Syntax**

```java
void Connection.setSyncObserver(SyncObserver so)
```

**Parameters**

- `so` A SyncObserver object or null to remove the currently registered SyncObserver object.

**Remarks**

This SyncObserver object is used by subsequent SYNCHRONIZE SQL statements.

The default is null, suggesting no observer.

**See also**

- “SyncObserver interface [UltraLiteJ]” on page 236

---

**synchronize method**

Synchronizes the database with a MobiLink server.

**Syntax**

```java
void Connection.synchronize(SyncParms config) throws ULjException
```

**Parameters**

- `config` The SyncParms object containing the parameters used for synchronization.

**See also**

- “SyncParms class [UltraLiteJ]” on page 242

---

**unregisterForEvent method [Android]**

Unregisters from a system event to stop receiving notifications.

**Syntax**

```java
void Connection.unregisterForEvent(
    short event_type,
    String object_name
) throws ULjException
```

**Parameters**

- `event_type` The type of event to unregister.

- `object_name` The object to which the event applies, such as a table name.
validateDatabase method [Android]

Validates the database on this connection.

Syntax

```java
void Connection.validateDatabase(
    int flags,
    ValidateDatabaseProgressListener listener,
    String tableName
) throws ULjException
```

Parameters

- **flags** Flags controlling the type of validation.
- **listener** The listener to receive validation progress information.
- **tableName** A specific table to validate, or null for all tables.

Remarks

Tables, indexes, and database pages can be validated depending on the flags passed to this routine. To receive information during the validation, implement a callback function and pass the address to this routine. To limit the validation to a specific table, pass in the table name or ID as the last parameter.

The flags parameter is a combination of the following values:

- ULVF_TABLE
- ULVF_INDEX
- ULVF_DATABASE
- ULVF_EXPRESS
- ULVF_FULL_VALIDATE

The following example demonstrates table and index validation in express mode:

```java
flags = ULVF_TABLE | ULVF_INDEX | ULVF_EXPRESS;
```

See also

- “Connection.ULVF_TABLE variable [Android] [UltraLiteJ]” on page 130
- “Connection.ULVF_INDEX variable [Android] [UltraLiteJ]” on page 129
- “Connection.ULVF_DATABASE variable [Android] [UltraLiteJ]” on page 129
- “Connection.ULVF_EXPRESS variable [Android] [UltraLiteJ]” on page 129
- “Connection.ULVF_FULL_VALIDATE variable [Android] [UltraLiteJ]” on page 129
waitForEvent method [Android]

Waits for an event notification.

Syntax

```java
ULjEvent Connection.waitForEvent(int wait_ms) throws ULjException
```

Parameters

- `wait_ms`  The time, in milliseconds, to wait (block) before returning. To wait indefinitely, set to -1.

Returns

The event that occurred within the wait time, or null if no notification was received within the wait time.

Remarks

This call blocks until a notification is received or until the given wait period expires. To cancel a wait, use the cancelWaitForEvent method.

See also

- “ULjEvent interface [Android] [UltraLiteJ]” on page 276
- “Connection.cancelWaitForEvent method [Android] [UltraLiteJ]” on page 106

CONNECTED variable

Denotes a connected state.

Syntax

```java
final byte Connection.CONNECTED
```

NOT_CONNECTED variable

Denotes a not connected state.

Syntax

```java
final byte Connection.NOT_CONNECTED
```

OPTION_BLOB_FILE_BASE_DIR variable [BlackBerry]

Database option: blob file base dir.

Syntax

```java
final String Connection.OPTION_BLOB_FILE_BASE_DIR
```
Remarks
For BlackBerry devices, the default value of the corresponding option is "file:///SDCard/"; otherwise, it is "".

See also
● “Connection.setOption method [UltraLiteJ]” on page 118
● “UltraLite Java edition blob_file_base_dir option” [UltraLite - Database Management and Reference]

OPTION_DATABASE_ID variable [BlackBerry]
Database option: database id.

Syntax
final String Connection.OPTION_DATABASE_ID

Remarks
A default value is not specified. It must be explicitly assigned.

See also
● “Connection.setOption method [UltraLiteJ]” on page 118

OPTION_DATE_FORMAT variable
Database option: date format.

Syntax
final String Connection.OPTION_DATE_FORMAT

Remarks
For BlackBerry smartphones, use this constant with the Connection.setOption method.

For Android smartphones, set this option in the database creation string of the ConfigPersistent.setCreationString method only.

The default value of the corresponding option is "YYYY-MM-DD".

See also
● “Connection.setOption method [UltraLiteJ]” on page 118
● “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97
● “UltraLite Java edition date_format option” [UltraLite - Database Management and Reference]

OPTION_DATE_ORDER variable
Database option: date order.
Syntax

```java
final String Connection.OPTION_DATE_ORDER
```

Remarks

For BlackBerry smartphones, use this constant with the Connection.setOption method.

For Android smartphones, set this option in the database creation string with the ConfigPersistent.setCreationString method only.

The default value of the corresponding option is "YMD".

See also

- “Connection.setOption method [UltraLiteJ]” on page 118
- “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97
- “UltraLite Java edition date_order option” [UltraLite - Database Management and Reference]

**OPTION_MAX_HASH_SIZE variable**

Database option: maximum hash size.

Syntax

```java
final String Connection.OPTION_MAX_HASH_SIZE
```

Remarks

For BlackBerry smartphones, use this constant with the Connection.setOption method to set the MaxHashSize database property.

For Android smartphones, set this option in the database creation string with the ConfigPersistent.setCreationString method only. The option name is max_hash_size.

The default value of the corresponding option is "4".

If the SQL statement creating an index does not specify a hash size, the value specified by this option is used as the default.

See also

- “Connection.setOption method [UltraLiteJ]” on page 118
- “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97
- “Index hashing” [UltraLite - Database Management and Reference]

**OPTION_ML_REMOTE_ID variable [BlackBerry]**

Database option: ML remote ID.

Syntax

```java
final String Connection.OPTION_ML_REMOTE_ID
```
Remarks
A default value is not specified. A value is set after the first MobiLink synchronization.

See also
● “Connection.setOption method [UltraLiteJ]” on page 118

OPTION_ML_SERVER_VERSION variable [BlackBerry]
Database option: MobiLink server protocol version.

Syntax
final String Connection.OPTION_ML_SERVER_VERSION

See also
● “Connection.setOption method [UltraLiteJ]” on page 118

OPTION_NEAREST_CENTURY variable
Database option: nearest century.

Syntax
final String Connection.OPTION_NEAREST_CENTURY

Remarks
For BlackBerry smartphones, use this constant with the Connection.setOption method.

For Android smartphones, set this option in the database creation string with the ConfigPersistent.setCreationString method only.

The default value of the corresponding option is "50".

See also
● “Connection.setOption method [UltraLiteJ]” on page 118
● “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97
● “UltraLite Java edition nearest_century option” [UltraLite - Database Management and Reference]

OPTION_PRECISION variable
Database option: precision.

Syntax
final String Connection.OPTION_PRECISION
Remarks
For BlackBerry smartphones, use this constant with the Connection.setOption method.

For Android smartphones, set this option in the database creation string with the ConfigPersistent.setCreationString method only.

The default value of the corresponding option is "30".

See also
● “Connection.setOption method [UltraLiteJ]” on page 118
● “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97
● “UltraLite Java edition precision option” [UltraLite - Database Management and Reference]

**OPTION_SCALE** variable
Database option: scale.

Syntax
```java
final String Connection.OPTION_SCALE
```

Remarks
For BlackBerry smartphones, use this constant with the Connection.setOption method.

For Android smartphones, set this option in the database creation string with the ConfigPersistent.setCreationString method only.

The default value of the corresponding option is "6".

See also
● “Connection.setOption method [UltraLiteJ]” on page 118
● “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97
● “UltraLite Java edition scale option” [UltraLite - Database Management and Reference]

**OPTION_TIME_FORMAT** variable
Database option: time format.

Syntax
```java
final String Connection.OPTION_TIME_FORMAT
```

Remarks
For BlackBerry smartphones, use this constant with the Connection.setOption method.

For Android smartphones, set this option in the database creation string with the ConfigPersistent.setCreationString method only.
The default value of the corresponding option is "HH:NN:SS.SSS".

See also

- “Connection.setOption method [UltraLiteJ]” on page 118
- “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97
- “UltraLite Java edition time_format option” [UltraLite - Database Management and Reference]

**OPTION_TIMESTAMP_FORMAT variable**

Database option: timestamp format.

**Syntax**

```java
final String Connection.OPTION_TIMESTAMP_FORMAT
```

**Remarks**

For BlackBerry smartphones, use this constant with the Connection.setOption method.

For Android smartphones, set this option in the database creation string with the ConfigPersistent.setCreationString method only.

The default value of the corresponding option is "YYYY-MM-DD HH:NN:SS.SSS".

See also

- “Connection.setOption method [UltraLiteJ]” on page 118
- “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97
- “UltraLite Java edition timestamp_format option” [UltraLite - Database Management and Reference]

**OPTION_TIMESTAMP_INCREMENT variable**

Database option: timestamp increment.

**Syntax**

```java
final String Connection.OPTION_TIMESTAMP_INCREMENT
```

**Remarks**

For BlackBerry smartphones, use this constant with the Connection.setOption method.

For Android smartphones, set this option in the database creation string with the ConfigPersistent.setCreationString method only.

The default value of the corresponding option is "1".
OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT variable

Database option: timestamp with time zone format.

Syntax

final String Connection.OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT

Remarks

For BlackBerry smartphones, use this constant with the Connection.setOption method.

For Android smartphones, set this option in the database creation string with the ConfigPersistent.setCreationString method only.

The default value of the corresponding option is "YYYY-MM-DD HH:NN:SS.SSS+HH:NN".

See also

● “Connection.setOption method [UltraLiteJ]” on page 118
● “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” on page 97
● “UltraLite Java edition timestamp_increment option” [UltraLite - Database Management and Reference]

PROPERTY_DATABASE_NAME variable

Database Property: database name.

Syntax

final String Connection.PROPERTY_DATABASE_NAME

Remarks

Set this property is with the Configuration.setDatabaseName method.

See also

● “Configuration.setDatabaseName method [UltraLiteJ]” on page 101
● “Connection.getDatabaseProperty method [UltraLiteJ]” on page 111
PROPERTY_PAGE_SIZE variable

Database Property: page size.

Syntax
final String Connection.PROPERTY_PAGE_SIZE

Remarks
Set this property is with the Configuration.setPageSize method.

See also
- “Configuration.setPageSize method [UltraLiteJ]” on page 101
- “Connection.getDatabaseProperty method [UltraLiteJ]” on page 111

SYNC_ALL variable

The publication list used to request synchronization of all tables in the database, including tables not in any publication.

Syntax
final String Connection.SYNC_ALL

Remarks
Tables marked as NoSync are never synchronized.
This constant is equivalent to the null reference or an empty string.

SYNC_ALL_DB_PUB_NAME variable

The reserved name of the SYNC_ALL_DB publication.

Syntax
final String Connection.SYNC_ALL_DB_PUB_NAME

See also
- “Connection.getLastDownloadTime method [UltraLiteJ]” on page 112
- “Connection.resetLastDownloadTime method [UltraLiteJ]” on page 116

SYNC_ALL_PUBS variable

The publication list used to request synchronization of all publications in the database.

Syntax
final String Connection.SYNC_ALL_PUBS
Remarks
Tables that are marked as NoSync are never synchronized.

ULVF_DATABASE variable [Android]
Used to validate database.

Syntax
final int Connection.ULVF_DATABASE

Remarks
Verify all database pages using page checksums and additional checks.

See also
● “Connection.validateDatabase method [Android] [UltraLiteJ]” on page 120

ULVF_EXPRESS variable [Android]
Used to perform a faster, though less thorough, validation.

Syntax
final int Connection.ULVF_EXPRESS

Remarks
This flag modifies others specified.

See also
● “Connection.validateDatabase method [Android] [UltraLiteJ]” on page 120

ULVF_FULL_VALIDATE variable [Android]
Performs all types of validation on the database.

Syntax
final int Connection.ULVF_FULL_VALIDATE

See also
● “Connection.validateDatabase method [Android] [UltraLiteJ]” on page 120

ULVF_INDEX variable [Android]
Used to validate indexes.
Syntax

```java
final int Connection.ULVF_INDEX
```

Remarks
Check the integrity of the index.

See also
- “Connection.validateDatabase method [Android] [UltraLiteJ]” on page 120

**ULVF_TABLE variable [Android]**

Used to validate table(s).

Syntax

```java
final int Connection.ULVF_TABLE
```

Remarks
Check that table and index row counts match.

See also
- “Connection.validateDatabase method [Android] [UltraLiteJ]” on page 120

**DatabaseInfo interface**

Associated with a Connection object and provides methods to reveal database information.

Syntax

```java
public interface DatabaseInfo
```

Members
All members of the DatabaseInfo interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCommitCount method [BlackBerry]</td>
<td>Returns the total number of commit operations performed on the database.</td>
</tr>
<tr>
<td>getDbFormat method [BlackBerry]</td>
<td>Returns the database version number.</td>
</tr>
<tr>
<td>getDbSize method [BlackBerry]</td>
<td>Returns the database size.</td>
</tr>
<tr>
<td>getLogSize method [BlackBerry]</td>
<td>Returns the overall size of the transaction log, in bytes.</td>
</tr>
<tr>
<td>getNumberRowsToUpload method</td>
<td>Returns the number of rows awaiting upload.</td>
</tr>
</tbody>
</table>
### DatabaseInfo Interface

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getPageReads method</td>
<td>Returns the number of page reads.</td>
</tr>
<tr>
<td>getPageSize method</td>
<td>Returns the page size of the database, in bytes.</td>
</tr>
<tr>
<td>getPageWrites method</td>
<td>Returns the number of page writes.</td>
</tr>
<tr>
<td>getRelease method</td>
<td>Returns the software release number.</td>
</tr>
</tbody>
</table>

**Remarks**

This interface is invoked with the getDatabaseInfo method of a Connection object.

**See also**

- “Connection interface [UltraLiteJ]” on page 102
- “Connection.getDatabaseInfo method [UltraLiteJ]” on page 111

### getCommitCount method [BlackBerry]

Returns the total number of commit operations performed on the database.

**Syntax**

```java
int DatabaseInfo.getCommitCount()
```

**Returns**

The total number of commit operations.

### getDbFormat method [BlackBerry]

Returns the database version number.

**Syntax**

```java
int DatabaseInfo.getDbFormat()
```

**Returns**

The version number.

### getDbSize method [BlackBerry]

Returns the database size.

**Syntax**

```java
int DatabaseInfo.getDbSize()
```
Returns

-1 when the database is not persistent; otherwise, the current size of the persistent store is returned.

**getLogSize method [BlackBerry]**

Returns the overall size of the transaction log, in bytes.

Syntax

```java
int DatabaseInfo.getLogSize()
```

Returns

The size of transaction log.

**getNumberRowsToUpload method**

Returns the number of rows awaiting upload.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getNumberRowsToUpload() method</td>
<td>Returns the number of rows awaiting upload.</td>
</tr>
<tr>
<td>getNumberRowsToUpload(String, int) method [Android]</td>
<td>Returns the number of rows awaiting upload up to a given threshold.</td>
</tr>
</tbody>
</table>

**getNumberRowsToUpload() method**

Returns the number of rows awaiting upload.

Syntax

```java
int DatabaseInfo.getNumberRowsToUpload()
```

Returns

The number of rows.

**getNumberRowsToUpload(String, int) method [Android]**

Returns the number of rows awaiting upload up to a given threshold.

Syntax

```java
int DatabaseInfo.getNumberRowsToUpload(String pubList, int threshold)
```
Parameters

- **pubList**  A string containing a comma-separated list of publications to check. An empty string (the UL_SYNC_ALL macro) implies all tables except tables marked as **no sync**. A string containing just an asterisk (the UL_SYNC_ALL_PUBS macro) implies that all tables referred to in any publication. Some tables may not be part of any publication and are not included if this value is *.

- **threshold**  The maximum number of rows to count, limiting the amount of time taken by the call. A threshold of 0 corresponds to no limit (all rows that need to be synchronized are counted). A threshold of 1 can be used to quickly determine if any rows need to be synchronized.

Returns

The number of rows that need to be synchronized, either in a specified set of publications or in the whole database.

**getPageReads method**

Returns the number of page reads.

**Syntax**

```c
int DatabaseInfo.getPageReads()
```

**Returns**

The number of page reads.

**Remarks**

For Android, the number is the total accumulated page reads, and is stored in an instance variable of this class. For Blackberry and J2SE, the number is the total accumulated page reads, and is stored in the database.

**getPageSize method**

Returns the page size of the database, in bytes.

**Syntax**

```c
int DatabaseInfo.getPageSize()
```

**Returns**

The page size.

**getPageWrites method**

Returns the number of page writes.
Syntax

```java
int DatabaseInfo.getPageWrites()
```

**Returns**

The number of page writes. For Android, the number is the total accumulated page writes, and is stored in an instance variable of this class. For Blackberry and J2SE, the number is the total accumulated page writes, and is stored in the database.

**getRelease method**

Returns the software release number.

**Syntax**

```java
String DatabaseInfo.getRelease()
```

**Returns**

The release number.

**Remarks**

For example, a software release value of "12.0.1.1234" represents the 12.0.1 release and the 1234 build number.

**DatabaseManager class**

Provides static methods to obtain basic configurations, create a new database, and connect to an existing database.

**Syntax**

```java
public class DatabaseManager
```

**Members**

All members of the DatabaseManager class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect method</td>
<td>Connects to an existing database based on a configuration set.</td>
</tr>
<tr>
<td>createConfigurationNonPersistent method [BlackBerry]</td>
<td>Creates a Configuration object for a non-persistent database store, and returns a ConfigNonPersist object.</td>
</tr>
<tr>
<td>createConfigurationObjectStore method [BlackBerry]</td>
<td>Creates a Configuration object for a RIM object store, and returns a ConfigObjectStore object.</td>
</tr>
</tbody>
</table>
# DatabaseManager class

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createDatabase method</td>
<td>Creates a new database based on a set of configurations, and connects to the database.</td>
</tr>
<tr>
<td>createFileTransfer method</td>
<td>Creates a FileTransfer object for transferring files to or from MobiLink.</td>
</tr>
<tr>
<td>createFileTransferAndroid method [Android]</td>
<td>Creates a FileTransfer object for transferring files to or from MobiLink.</td>
</tr>
<tr>
<td>createObjectStoreTransfer method [BlackBerry]</td>
<td>Creates a FileTransfer object for transferring UltraLite Java edition databases to or from MobiLink, and storing them into a RIM object store.</td>
</tr>
<tr>
<td>createSISHTTPLlistener method [BlackBerry]</td>
<td>Creates an SISListener object for server-initiated synchronizations.</td>
</tr>
<tr>
<td>release method</td>
<td>Closes this DatabaseManager object to release all connections and to shutdown all databases.</td>
</tr>
<tr>
<td>setErrorLanguage method</td>
<td>Sets the language to use for error messages.</td>
</tr>
</tbody>
</table>

## Remarks

The following example demonstrates how to open an existing database or create a new one if it does not exist on the J2SE platform:

```java
Connection conn;
ConfigFile config = DatabaseManager.createConfigurationFile("test.ulj");
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```

The following example demonstrates how to open an existing database or create a new one if it does not exist on a BlackBerry device:

```java
Connection conn;
ConfigObjectStore config = DatabaseManager.createConfigurationObjectStore("test.ulj");
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```

The following example demonstrates how to open an existing database or create a new one if it does not exist on an Android device:
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.
        }
    }
}

See also
● “Connection interface [UltraLiteJ]” on page 102
● “Configuration interface [UltraLiteJ]” on page 100

connect method
Connects to an existing database based on a configuration set.

Syntax
Connection DatabaseManager.connect (Configuration config) throws ULjException

Parameters
● config The Configuration object containing the specifications for the existing database.

Returns
A Connection object that establishes the connection to the database.

Remarks
Only one application, at most, can connect to a given UltraLite Java edition database concurrently.

See also
● “Configuration interface [UltraLiteJ]” on page 100
● “Connection interface [UltraLiteJ]” on page 102
● “UltraLite and UltraLite Java edition database creation and connection approaches” on page 5

createConfigurationFile method
Creates a Configuration object for a physical database store from a file and returns a ConfigFile object.
Syntax

```java
ConfigFile DatabaseManager.createConfigurationFile(
    String file_name
) throws ULjException
```

Parameters

- **file_name**  The name of file to use or create.

Returns

The ConfigFile object used to configure a database.

See also

- “ConfigFile interface [UltraLiteJ]” on page 85

**createConfigurationFileAndroid method**

Creates a Configuration object for a physical database store from a file on an Android device, and returns a ConfigFileAndroid object.

Syntax

```java
ConfigFileAndroid DatabaseManager.createConfigurationFileAndroid(
    String file_name,
    android.content.Context context
) throws ULjException
```

Parameters

- **file_name**  The name of the database file to use or create. You must have read-write access to the database path. The default path for this file is `/data/data/your-application-package-name/`, where `your-application-package-name` is the package name you assigned to your application. You can include an absolute path with the file name to specify a different location for the database.

- **context**  The Context object from an Android application. This parameter must not be null.

Returns

The ConfigFileAndroid object used to configure a database.

See also

- “ConfigFileAndroid interface [Android] [UltraLiteJ]” on page 86

**createConfigurationNonPersistent method [BlackBerry]**

Creates a Configuration object for a non-persistent database store, and returns a ConfigNonPersist object.
Syntax

```
ConfigNonPersistent DatabaseManager.createConfigurationNonPersistent(
    String db_name
) throws ULjException
```

Parameters
- **db_name**  The name of the non-persistent database.

Returns
The ConfigNonPersistent object used to configure the database.

See also
- “ConfigNonPersistent interface [BlackBerry] [UltraLiteJ]” on page 88

### createConfigurationObjectStore method [BlackBerry]

Creates a Configuration object for a RIM object store, and returns a ConfigObjectStore object.

Syntax

```
ConfigObjectStore DatabaseManager.createConfigurationObjectStore(
    String db_name
) throws ULjException
```

Parameters
- **db_name**  The name of the database.

Returns
The ConfigObjectStore object used to configure the database.

See also
- “ConfigObjectStore interface [BlackBerry] [UltraLiteJ]” on page 89

### createDatabase method

Creates a new database based on a set of configurations, and connects to the database.

Syntax

```
Connection DatabaseManager.createDatabase(
    Configuration config
) throws ULjException
```

Parameters
- **config**  A Configuration object containing the specifications for the new database.
Returns
A Connection object that establishes a connection the new database.

Remarks
This method replaces any existing database on the device that may have the same name.

See also
● “Configuration interface [UltraLiteJ]” on page 100
● “Connection interface [UltraLiteJ]” on page 102

cREATEFILETRANSFER method
Creates a FileTransfer object for transferring files to or from MobiLink.

Syntax
FileTransfer DatabaseManager.createFileTransfer(
    String fileName,
    int streamType,
    String userName,
    String version
) throws ULjException

Parameters
● fileName The name of the server file to transfer. This parameter must not contain any path information.
● streamType One of the constants defined in the SyncParms class used to identify the type of communication stream.
● userName The MobiLink user name.
● version The MobiLink script version.

Returns
The FileTransfer object.

See also
● “SyncParms class [UltraLiteJ]” on page 242

cREATEFILETRANSFERANDROID method [Android]
Creates a FileTransfer object for transferring files to or from MobiLink.

Syntax
FileTransfer DatabaseManager.createFileTransferAndroid(
    android.content.Context context,
String fileName,
int streamType,
String userName,
String version
) throws ULjException

Parameters

- **context**  The Context object from an Android application. This parameter must not be null.
- **fileName**  The name of the server file to transfer. This parameter must not contain any path information.
- **streamType**  One of the constants defined in the SyncParms class used to identify the type of communication stream.
- **userName**  The MobiLink user name.
- **version**  The MobiLink script version.

Returns

The FileTransfer object.

Remarks

This method is recommended for Android. It must be used if a database connection has not been created using the createConfigurationFileAndroid method.

See also

- “DatabaseManager.createConfigurationFileAndroid method [UltraLiteJ]” on page 137

**createObjectStoreTransfer method [BlackBerry]**

Creates a FileTransfer object for transferring UltraLite Java edition databases to or from MobiLink, and storing them into a RIM object store.

Syntax

FileTransfer DatabaseManager.createObjectStoreTransfer(
    String fileName,
    int streamType,
    String userName,
    String version
) throws ULjException

Parameters

- **fileName**  The name of the server file to transfer. This parameter Must not contain any path information.
- **streamType**  One of the constants defined in the SyncParms class used to identify the type of communication stream.
The MobiLink user name.

version

The MobiLink script version.

Returns

The FileTransfer object.

See also

“SyncParms class [UltraLiteJ]” on page 242

createSISHTTPListener method [BlackBerry]

Creates an SISListener object for server-initiated synchronizations.

Syntax

SISListener DatabaseManager.createSISHTTPListener(
    SISRequestHandler handler,
    int port,
    String httpOptions
) throws ULjException

Parameters

- handler A SISRequestHandler object specified to handle server-initiated synchronization requests.
- port An HTTP port to listen for server messages.
- httpOptions The HTTP options for connecting to the server.

Returns

The SISListener object to use for server-initiated synchronizations.

Remarks

4400 is the recommended port setting.

"deviceside=false" is the recommended HTTP option for BlackBerry simulators.

The MobiLink side of the BlackBerry HTTP SISListener requires the target device be associated with a BlackBerry Enterprise Server, such as a BES activated device).

release method

Closes this DatabaseManager object to release all connections and to shutdown all databases.

Syntax

void DatabaseManager.release() throws ULjException
Remarks
On Android, this method releases all connections that were created with this DatabaseManager.
Any uncommitted transactions are rolled back.

**setErrorLanguage method**
Sets the language to use for error messages.

**Syntax**
```java
void DatabaseManager.setDefaultLanguage(String lang)
```

**Parameters**
- `lang` The language code, represented as a double character.

**Remarks**
Recognized languages are EN, DE, FR, JA, ZH. If an unrecognized language is specified, the system
reverts to the default language, "EN".

In J2SE and BlackBerry environments, the current locale is used to determine the default language.

**DecimalNumber interface**
Describes an exact decimal value and provides decimal arithmetic support for Java platforms where
java.math.BigDecimal is not available.

**Syntax**
```java
public interface DecimalNumber
```

**Members**
All members of the DecimalNumber interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add method</td>
<td>Adds two DecimalNumber objects together and returns the sum.</td>
</tr>
<tr>
<td>divide method</td>
<td>Divides the first DecimalNumber object by the second DecimalNumber object returns the quotient.</td>
</tr>
<tr>
<td>getString method</td>
<td>Returns the String representation of the DecimalNumber object.</td>
</tr>
<tr>
<td>isNull method</td>
<td>Determines if the DecimalNumber object is null.</td>
</tr>
<tr>
<td>multiply method</td>
<td>Multiplies two DecimalNumber objects together and returns the product.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>set method</td>
<td>Sets the DecimalNumber object with a String value.</td>
</tr>
<tr>
<td>setNull method</td>
<td>Sets the DecimalNumber object to null.</td>
</tr>
<tr>
<td>subtract method</td>
<td>Subtracts the second DecimalNumber object from the first DecimalNumber object and returns the difference.</td>
</tr>
</tbody>
</table>

### add method

Adds two DecimalNumber objects together and returns the sum.

**Syntax**

```java
DecimalNumber add(DecimalNumber num1, DecimalNumber num2) throws ULjException
```

**Parameters**

- **num1** A number.
- **num2** Another number.

**Returns**

The sum of num1 and num2.

### divide method

Divides the first DecimalNumber object by the second DecimalNumber object returns the quotient.

**Syntax**

```java
DecimalNumber divide(DecimalNumber num1, DecimalNumber num2) throws ULjException
```

**Parameters**

- **num1** A dividend.
- **num2** A divisor.

**Returns**

The quotient of num1 divided by num2.
**getString method**

Returns the String representation of the DecimalNumber object.

Syntax

```java
String DecimalNumber.getString() throws ULjException
```

Returns

The String value.

**isNull method**

Determines if the DecimalNumber object is null.

Syntax

```java
boolean DecimalNumber.isNull()
```

Returns

True if the object is null; otherwise, returns false.

**multiply method**

Multiplies two DecimalNumber objects together and returns the product.

Syntax

```java
DecimalNumber DecimalNumber.multiply(DecimalNumber num1, DecimalNumber num2) throws ULjException
```

Parameters

- **num1** A multiplicand.
- **num2** A multiplier.

Returns

The product of num1 and num2.

**set method**

Sets the DecimalNumber object with a String value.

Syntax

```java
void DecimalNumber.set(String value) throws ULjException
```
Parameters

- **value**  A numerical value represented as a String.

**setNull method**

Sets the DecimalNumber object to null.

**Syntax**

```java
void DecimalNumber.setNull() throws ULjException
```

**subtract method**

Subtracts the second DecimalNumber object from the first DecimalNumber object and returns the difference.

**Syntax**

```java
DecimalNumber subtract(DecimalNumber num1, DecimalNumber num2) throws ULjException
```

**Parameters**

- **num1**  A minuend.
- **num2**  A subtrahend.

**Returns**

The difference between num1 and num2.

**Domain interface**

Describes the Domain object type information for a column in a table.

**Syntax**

```java
public interface Domain
```

**Members**

All members of the Domain interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BIG variable</strong></td>
<td>Denotes the domain ID constant for a 64-bit integer (BIGINT SQL type).</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BINARY variable</td>
<td>Denotes the domain ID constant for a variable-length binary object of maximum size bytes (BINARY(size) SQL type).</td>
</tr>
<tr>
<td>BIT variable</td>
<td>Denotes the domain ID constant for a bit (BIT SQL type).</td>
</tr>
<tr>
<td>DATE variable</td>
<td>Denotes the domain ID constant for a Date (DATE SQL type).</td>
</tr>
<tr>
<td>DOMAIN_MAX variable</td>
<td>Denotes the maximum kinds of Domain types.</td>
</tr>
<tr>
<td>DOUBLE variable</td>
<td>Denotes the domain ID constant for a 8-byte floating point (DOUBLE SQL type).</td>
</tr>
<tr>
<td>INTEGER variable</td>
<td>Denotes the domain ID constant for a 32-bit integer (INTEGER SQL type).</td>
</tr>
<tr>
<td>LONGBINARY variable</td>
<td>Denotes the domain ID constant for an arbitrary long block of binary data (BLOB) (LONG BINARY SQL type).</td>
</tr>
<tr>
<td>LONGBINARYFILE variable</td>
<td>Denotes the domain ID constant for an arbitrary file of data.</td>
</tr>
<tr>
<td>LONGVARCHAR variable</td>
<td>Denotes the domain ID constant for an arbitrary long block of character data (CLOB) (LONG VARCHAR SQL type).</td>
</tr>
<tr>
<td>NUMERIC variable</td>
<td>Denotes the domain ID constant for a numeric value of fixed precision (size) total digits and with scale digits after the decimal (NUMER-IC(precision,scale) SQL type).</td>
</tr>
<tr>
<td>REAL variable</td>
<td>Denotes the domain ID constant for a 4-byte floating point (REAL SQL type).</td>
</tr>
<tr>
<td>SHORT variable</td>
<td>Denotes the domain ID constant for a 16-bit integer (SMALLINT SQL type).</td>
</tr>
<tr>
<td>ST_GEOMETRY variable</td>
<td>Denotes the denotes the domain ID constant for a geometry (GEOMETRY SQL type)</td>
</tr>
<tr>
<td>TIME variable</td>
<td>Denotes the domain ID constant for a Time (TIME SQL type).</td>
</tr>
<tr>
<td>TIMESTAMP variable</td>
<td>Denotes the domain ID constant for a Timestamp (TIMESTAMP SQL type).</td>
</tr>
<tr>
<td>TIMESTAMP_ZONE variable</td>
<td>Denotes the domain ID constant for a timestamps with time zones (DATETIMEOFFSET SQL type).</td>
</tr>
<tr>
<td>TINY variable</td>
<td>Denotes the domain ID constant for a unsigned 8-bit integer (TINYINT SQL type).</td>
</tr>
</tbody>
</table>
### Name | Description
--- | ---
UNSIGNED_BIG variable | Denotes the domain ID constant for a unsigned 64-bit integer (UN-SIGNED BIGINT SQL type).
UNSIGNED_INTEGER variable | Denotes the domain ID constant for a unsigned 32-bit integer (UN-SIGNED INTEGER SQL type).
UNSIGNED_SHORT variable | Denotes the domain ID constant for a unsigned 16-bit integer (UN-SIGNED SMALLINT SQL type).
UUID variable | Denotes the domain ID constant for a UniqueIdentifier (UNIQUEIDENTIFIER SQL type).
VARCHAR variable | Denotes the domain ID constant for a variable-length character string of maximum size bytes (VARCHAR(size) SQL type).

### Remarks
This interface contains constants to denote the various domains, and methods that extract information from a Domain object.

See the Connection interface for an example of creating a schema for a simple database.

Types can be classified as follows:

#### Integer Types:

<table>
<thead>
<tr>
<th>Domain Constant</th>
<th>SQL Type</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>BIT</td>
<td>0 or 1</td>
</tr>
<tr>
<td>TINY</td>
<td>TINYINT</td>
<td>0 to 255 (unsigned integer using 1 byte of storage)</td>
</tr>
<tr>
<td>SHORT</td>
<td>SMALLINT</td>
<td>-32768 to 32767 (signed integer using 2 bytes of storage)</td>
</tr>
<tr>
<td>UNSIGNED_SHORT</td>
<td>UNSIGNED SMALLINT</td>
<td>0 to 65535 (unsigned integer using 2 bytes of storage)</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
<td>$-2^{31}$ to $2^{31} - 1$, or -2147483648 to 2147483647 (signed integer using 4 bytes of storage)</td>
</tr>
<tr>
<td>UNSIGNED_INTEGER</td>
<td>UNSIGNED INTEGER</td>
<td>0 to $2^{32} - 1$, or 0 to 4294967295 (unsigned integer using 4 bytes of storage)</td>
</tr>
<tr>
<td>Domain Constant</td>
<td>SQL Type</td>
<td>Value Range</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BIG</td>
<td>BIGINT</td>
<td>(-2^{63} ) to (2^{63} - 1), or (-9223372036854775808) to (9223372036854775807) (signed integer using 8 bytes of storage)</td>
</tr>
<tr>
<td>UNSIGNED_BIG</td>
<td>UNSIGNED BIGINT</td>
<td>0 to (2^{64} - 1), or 0 to (18446744073709551615) (unsigned integer using 8 bytes of storage)</td>
</tr>
</tbody>
</table>

Non-Integer Numeric Types:

<table>
<thead>
<tr>
<th>Domain Constant</th>
<th>SQL Type</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>(-3.402823e+38) to (3.402823e+38), with numbers close to zero as small as (1.175495e-38) (single precision floating point number using 4 bytes of storage, rounding errors may occur after the sixth digit)</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>(-1.79769313486231e+308) to (1.79769313486231e+308), with numbers close to zero as small as (2.22507385850721e-308) (single precision floating point number using 8 bytes of storage, rounding errors may occur after the fifteenth digit)</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>NUMERIC(precision,scale)</td>
<td>Any decimal numbers with (precision) (size) total digits and with (scale) digits after the decimal point (no rounding within precision)</td>
</tr>
</tbody>
</table>

Character and Binary Types:

<table>
<thead>
<tr>
<th>Domain Constant</th>
<th>SQL Type</th>
<th>Size Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR</td>
<td>VARCHAR(size)</td>
<td>1 to 32767 bytes (characters are stored as 1-3 byte UTF-8 characters). When evaluating expressions, the maximum length for a temporary character value is 2048 bytes.</td>
</tr>
<tr>
<td>LONG-VARCHAR</td>
<td>LONG VARCHAR</td>
<td>Any length (memory permitting). The only operations allowed on LONG VARCHAR columns are to insert, update, or delete them, or to include them in the select-list of a query.</td>
</tr>
<tr>
<td>BINARY</td>
<td>BINARY(size)</td>
<td>1 to 32767 bytes. When evaluating expressions, the maximum length for a temporary character value is 2048 bytes.</td>
</tr>
<tr>
<td>LONG-BINARY</td>
<td>LONG BINARY</td>
<td>Any length (memory permitting). The only operations allowed on LONG BINARY columns are to insert, update, or delete them, or to include them in the select-list of a query.</td>
</tr>
</tbody>
</table>
## Domain interface

### UUID

<table>
<thead>
<tr>
<th>Domain Constant</th>
<th>SQL Type</th>
<th>Size Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>UUID</td>
<td>UNIQUEIDENTIFIER</td>
<td>Always 16 bytes binary with special interpretation.</td>
</tr>
</tbody>
</table>

Date and Time Types:

<table>
<thead>
<tr>
<th>Domain Constant</th>
<th>SQL Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>Year, month, day.</td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
<td>Hour, minute, second, and fraction of a second.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>DATE and TIME.</td>
</tr>
<tr>
<td>TIMESTAMP_ZONE</td>
<td>TIMESTAMP_ZONE</td>
<td>DATE and TIME with time zone.</td>
</tr>
</tbody>
</table>

BIT columns are not nullable by default. All other types are nullable by default.

### BIG variable

Denotes the domain ID constant for a 64-bit integer (BIGINT SQL type).

**Syntax**

```
final short Domain_BIG
```

### BINARY variable

Denotes the domain ID constant for a variable-length binary object of maximum `size` bytes (BINARY(`size`) SQL type).

**Syntax**

```
final short Domain_BINARY
```

### See also

- “Connection interface [UltraLiteJ]” on page 102
- “Domain interface [UltraLiteJ]” on page 145
**BIT variable**
Denotes the domain ID constant for a bit (BIT SQL type).

Syntax
```java
final short Domain.BIT
```

Remarks
BIT columns are not nullable by default.

See also
- “Domain interface [UltraLiteJ]” on page 145

**DATE variable**
Denotes the domain ID constant for a Date (DATE SQL type).

Syntax
```java
final short Domain.DATE
```

See also
- “Domain interface [UltraLiteJ]” on page 145

**DOMAIN_MAX variable**
Denotes the maximum kinds of Domain types.

Syntax
```java
final short Domain.DOMAIN_MAX
```

**DOUBLE variable**
Denotes the domain ID constant for a 8-byte floating point (DOUBLE SQL type).

Syntax
```java
final short Domain.DOUBLE
```

See also
- “Domain interface [UltraLiteJ]” on page 145

**INTEGER variable**
Denotes the domain ID constant for a 32-bit integer (INTEGER SQL type).
Syntax

```java
final short Domain.INTEGER
```

See also

- “Domain interface [UltraLiteJ]” on page 145

LONGBINARY variable

Denotes the domain ID constant for an arbitrary long block of binary data (BLOB) (LONG BINARY SQL type).

Syntax

```java
final short Domain.LONGBINARY
```

See also

- “Domain interface [UltraLiteJ]” on page 145

LONGBINARYFILE variable

Denotes the domain ID constant for an arbitrary file of data.

Syntax

```java
final short Domain.LONGBINARYFILE
```

See also

- “Domain interface [UltraLiteJ]” on page 145

LONGVARCHAR variable

Denotes the domain ID constant for an arbitrary long block of character data (CLOB) (LONG VARCHAR SQL type).

Syntax

```java
final short Domain.LONGVARCHAR
```

See also

- “Domain interface [UltraLiteJ]” on page 145

NUMERIC variable

Denotes the domain ID constant for a numeric value of fixed precision (size) total digits and with scale digits after the decimal (NUMERIC(precision, scale) SQL type).
Syntax

    final short Domain.NUMERIC

See also

- “Domain interface [UltraLiteJ]” on page 145

**REAL variable**

Denotes the domain ID constant for a 4-byte floating point (REAL SQL type).

Syntax

    final short Domain.REAL

See also

- “Domain interface [UltraLiteJ]” on page 145

**SHORT variable**

Denotes the domain ID constant for a 16-bit integer (SMALLINT SQL type).

Syntax

    final short Domain.SHORT

See also

- “Domain interface [UltraLiteJ]” on page 145

**ST_GEOMETRY variable**

Denotes the domain ID constant for a geometry (GEOMETRY SQL type).

Syntax

    final short Domain.ST_GEOMETRY

See also

- “Domain interface [UltraLiteJ]” on page 145

**TIME variable**

Denotes the domain ID constant for a Time (TIME SQL type).

Syntax

    final short Domain.TIME
See also
● “Domain interface [UltraLiteJ]” on page 145

**TIMESTAMP variable**
Denotes the domain ID constant for a Timestamp (TIMESTAMP SQL type).

**Syntax**
```java
final short Domain.TIMESTAMP
```

See also
● “Domain interface [UltraLiteJ]” on page 145

**TIMESTAMP_ZONE variable**
Denotes the domain ID constant for a timestamps with time zones (DATETIMEOFFSET SQL type).

**Syntax**
```java
final short Domain.TIMESTAMP_ZONE
```

See also
● “Domain interface [UltraLiteJ]” on page 145

**TINY variable**
Denotes the domain ID constant for a unsigned 8-bit integer (TINYINT SQL type).

**Syntax**
```java
final short Domain.TINY
```

See also
● “Domain interface [UltraLiteJ]” on page 145

**UNSIGNED_BIG variable**
Denotes the domain ID constant for a unsigned 64-bit integer (UNSIGNED BIGINT SQL type).

**Syntax**
```java
final short Domain.UNSIGNED_BIG
```

See also
● “Domain interface [UltraLiteJ]” on page 145
**UNSIGNED_INTEGER variable**
Denotes the domain ID constant for a unsigned 32-bit integer (UNSIGNED INTEGER SQL type).

Syntax
```java
final short Domain.UNSIGNED_INTEGER
```

See also
- “Domain interface [UltraLiteJ]” on page 145

**UNSIGNED_SHORT variable**
Denotes the domain ID constant for a unsigned 16-bit integer (UNSIGNED SMALLINT SQL type).

Syntax
```java
final short Domain.UNSIGNED_SHORT
```

See also
- “Domain interface [UltraLiteJ]” on page 145

**UUID variable**
Denotes the domain ID constant for a UniqueIdentifier (UNIQUEIDENTIFIER SQL type).

Syntax
```java
final short Domain.UUID
```

See also
- “Domain interface [UltraLiteJ]” on page 145

**VARCHAR variable**
Denotes the domain ID constant for a variable-length character string of maximum `size` bytes (VARCHAR(`size`) SQL type).

Syntax
```java
final short Domain.VARCHAR
```

See also
- “Domain interface [UltraLiteJ]” on page 145
FileTransfer interface

Provides a mechanism to transfer files between the client and a MobiLink server.

Syntax

```java
public interface FileTransfer
```

Members

All members of the FileTransfer interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>downloadFile method</td>
<td>Downloads the file with the specified properties of this object.</td>
</tr>
<tr>
<td>getAuthenticationParms method</td>
<td>Returns parameters provided to a custom user authentication script.</td>
</tr>
<tr>
<td>getAuthStatus method</td>
<td>Returns the authorization status code of the last file transfer attempt.</td>
</tr>
<tr>
<td>getAuthValue method</td>
<td>Returns the value specified in custom user authentication synchronization scripts.</td>
</tr>
<tr>
<td>getFileAuthCode method</td>
<td>Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.</td>
</tr>
<tr>
<td>getLivenessTimeout method</td>
<td>Returns the liveness timeout length, in seconds.</td>
</tr>
<tr>
<td>getLocalFileName method</td>
<td>Determines the local file name.</td>
</tr>
<tr>
<td>getLocalPath method</td>
<td>Specifies where to find or store the file in the local file system.</td>
</tr>
<tr>
<td>getPassword method</td>
<td>Returns the MobiLink password for the user specified with the setUserName method.</td>
</tr>
<tr>
<td>getRemoteKey method</td>
<td>Determines the current remote key value.</td>
</tr>
<tr>
<td>getServerFileName method</td>
<td>Returns the name of the file on the server.</td>
</tr>
<tr>
<td>getStreamErrorCode method</td>
<td>Returns the error code reported by the stream.</td>
</tr>
<tr>
<td>getStreamErrorMessage method</td>
<td>Returns the error message reported by the stream itself.</td>
</tr>
<tr>
<td>getStreamParms method</td>
<td>Returns the parameters used to configure the synchronization stream.</td>
</tr>
<tr>
<td>getUserId method</td>
<td>Returns the MobiLink user name that uniquely identifies the client to the MobiLink server.</td>
</tr>
<tr>
<td>getVersion method</td>
<td>Returns the synchronization script to use.</td>
</tr>
</tbody>
</table>
### Name | Description
--- | ---
`isResumePartialTransfer` method  | Determines whether to resume or discard a previous partial transfer.
`isTransferredFile` method  | Checks whether the file was actually downloaded during the last file transfer attempt.
`setAuthenticationParms` method  | Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
`setLivenessTimeout` method  | Sets the liveness timeout length, in seconds.
`setLocalFileName` method  | Specifies the local file name.
`setLocalPath` method  | Specifies where to find or store the file in the local file system.
`setPassword` method  | Sets the MobiLink password for the user specified with the `setUserName` method.
`setRemoteKey` method  | Specifies the remote key.
`setResumePartialTransfer` method  | Specifies whether to resume or discard a previous partial transfer.
`setServerFileName` method  | Specifies the name of the file on the server.
`setUserName` method  | Sets the MobiLink user name that uniquely identifies the client to the MobiLink server.
`setVersion` method  | Sets the synchronization script to use.
`uploadFile` method  | Uploads the file with the specified properties of this object.

### Remarks

A FileTransfer object is obtained by calling the `DatabaseManager.createFileTransfer` or `DatabaseManager.createObjectStoreTransfer` methods.

The instance returned by the `createFileTransfer` method can be used to transfer any files between MobiLink and the local file system.

For Android devices and simulators, the local file system is either a media card or the internal file system where the application has appropriate permissions. Eg. `/sdcard/Android/data/your.package.name/files/`

The instance returned by the `createObjectStoreTransfer` method can be used to download UltraLite Java edition database files to the local BlackBerry object store, or vice versa.

Only valid, unencrypted UltraLite Java edition database files can be transferred. Attempting to download any files other than UltraLite Java edition databases result in exceptions being thrown.
Note
The application should not simultaneously start two downloads to the same local file.

See also
- “DatabaseManager.createFileTransfer method [UltraLiteJ]” on page 139
- “DatabaseManager.createObjectStoreTransfer method [BlackBerry] [UltraLiteJ]” on page 140

downloadFile method
Downloads the file with the specified properties of this object.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>downloadFile() method</td>
<td>Downloads the file with the specified properties of this object.</td>
</tr>
<tr>
<td>downloadFile(FileTransfer-ProgressListener) method</td>
<td>Download the file specified by the properties of this object with progress events posted to the specified listener.</td>
</tr>
</tbody>
</table>

downloadFile() method
Downloads the file with the specified properties of this object.

Syntax

```java
abstract boolean FileTransfer.downloadFile() throws ULjException
```

Returns

True if download is successful; otherwise, a ULjException is thrown and the method does not return normally.

Remarks

The file specified by the setServerFileName method is downloaded from the MobiLink server to the path specified by the setLocalPath method using the specified stream, userName, password, and script version.

Further options can be specified using the setLocalFileName(), setAuthenticationParms() and setResumePartialTransfer() methods.

To avoid file corruption, for desktop and BlackBerry file system downloads, UltraLiteJ downloads to a temporary file and only replaces the local file once the download has completed.

For BlackBerry object store downloads, UltraLiteJ starts to write to the local store directly. This is because, in this case, it is not possible to create atomic temporary objects. Any existing local database store with the same name is corrupted upon invoking the transferFile method.
A detailed result status can be fetched using the getAuthStatus(), getAuthValue(), getFileAuthCode(), isTransferredFile(), getStreamErrorCode(), and getStreamErrorMessage() methods.

**downloadFile(FileTransferProgressListener) method**

Download the file specified by the properties of this object with progress events posted to the specified listener.

**Syntax**

```java
abstract boolean FileTransfer.downloadFile(
    FileTransferProgressListener listener
) throws ULjException
```

**Parameters**

- **listener** The object that receives file transfer progress events.

**Returns**

True if the download is successful; otherwise, a ULjException is thrown, and the method does not return normally.

**Remarks**

Errors may result in no data being sent to the listener.

**See also**

- “FileTransfer.downloadFile method [UltraLiteJ]” on page 157

**getAuthenticationParms method**

Returns parameters provided to a custom user authentication script.

**Syntax**

```java
abstract String FileTransfer.getAuthenticationParms()
```

**Returns**

The list of authentication parms or null if no parameters are specified.

**See also**

- “FileTransfer.setAuthenticationParms method [UltraLiteJ]” on page 163

**getAuthStatus method**

Returns the authorization status code of the last file transfer attempt.
Syntax
abstract int FileTransfer.getAuthStatus()

Returns
An AuthStatusCode class value.

getAuthValue method
Returns the value specified in custom user authentication synchronization scripts.

Syntax
abstract long FileTransfer.getAuthValue()

Returns
An integer returned from custom user authentication synchronization scripts.

getFileAuthCode method
Returns the return value from the authenticate_file_transfer script for the last file transfer attempt.

Syntax
abstract int FileTransfer.getFileAuthCode()

Returns
An integer returned from the authenticate_file_transfer script for the last file transfer attempt.

getLivenessTimeout method
Returns the liveness timeout length, in seconds.

Syntax
abstract int FileTransfer.getLivenessTimeout()

Returns
The timeout.

See also
   ● “FileTransfer.setLivenessTimeout method [UltraLiteJ]” on page 164

getLocalFileName method
Determines the local file name.
Syntax

abstract String FileTransfer.getLocalFileName()

Returns

The local file name for the downloaded file.

Remarks

For file downloads, this is the name of the downloaded file. For file uploads, this is the name of the file to upload.

See also

● “FileTransfer.setLocalFileName method [UltraLiteJ]” on page 164

getLocalPath method

Specifies where to find or store the file in the local file system.

Syntax

abstract String FileTransfer.getLocalPath()

Returns

The local directory.

See also

● “FileTransfer.setLocalPath method [UltraLiteJ]” on page 165

getPassword method

Returns the MobiLink password for the user specified with the setUserName method.

Syntax

abstract String FileTransfer.getPassword()

Returns

The password for the MobiLink user.

See also

● “FileTransfer.setPassword method [UltraLiteJ]” on page 166

getRemoteKey method

Determines the current remote key value.
Syntax
   abstract String FileTransfer.getRemoteKey()

Returns
   The remote key value or null if the remote key is unspecified.

See also
   ● “FileTransfer.setRemoteKey method [UltraLiteJ]” on page 166

**getServerFileName method**
   Returns the name of the file on the server.

Syntax
   abstract String FileTransfer.getServerFileName()

Returns
   The name of the file in the server side.

Remarks
   For file downloads, this is the name of the file to download. For file uploads, this is the name of the uploaded file.

See also
   ● “FileTransfer.setServerFileName method [UltraLiteJ]” on page 167

**getStreamErrorCode method**
   Returns the error code reported by the stream.

Syntax
   abstract int FileTransfer.getStreamErrorCode()

Returns
   0 if there was no communication stream error; otherwise, returns the response code from the server.

Remarks
   The error code is the HTTP response code.

**getStreamErrorMessage method**
   Returns the error message reported by the stream itself.
**Syntax**

```java
abstract String FileTransfer.getStreamErrorMessage()
```

**Returns**

Null, if no message is available; otherwise, returns the response message.

**Remarks**

This is the HTTP response message.

---

**getStreamParms method**

Returns the parameters used to configure the synchronization stream.

**Syntax**

```java
abstract StreamHTTPParms FileTransfer.getStreamParms()
```

**Returns**

A StreamHTTPParms or StreamHTTPSParms object specifying the parameters for HTTP or HTTPS streams. The object is returned by reference.

**Remarks**

The synchronization stream type is specified when the FileTransfer object is created.

---

**getUserName method**

Returns the MobiLink user name that uniquely identifies the client to the MobiLink server.

**Syntax**

```java
abstract String FileTransfer.getUserName()
```

**Returns**

The MobiLink user name.

**See also**

- “FileTransfer.setUserName method [UltraLiteJ]” on page 168

---

**getVersion method**

Returns the synchronization script to use.

**Syntax**

```java
abstract String FileTransfer.getVersion()
```
Returns

The script version.

See also

●  “FileTransfer.setVersion method [UltraLiteJ]” on page 168

isResumePartialTransfer method

Determines whether to resume or discard a previous partial transfer.

Syntax

abstract boolean FileTransfer.isResumePartialTransfer()

Returns

True if to resume the download; otherwise, returns false.

See also

●  “FileTransfer.setResumePartialTransfer method [UltraLiteJ]” on page 167

isTransferredFile method

Checks whether the file was actually downloaded during the last file transfer attempt.

Syntax

abstract boolean FileTransfer.isTransferredFile()

Returns

True if the file transferred; otherwise, returns false.

Remarks

If the file is already up-to-date when the transferFile() method is invoked, this method returns true while the isTransferredFile() method returns false.

If an error occurs and the transferFile() method throws an exception, the isTransferredFile() method returns false.

setAuthenticationParms method

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).
Syntax

```java
abstract void FileTransfer.setAuthenticationParms(
    String authParms
) throws ULjException
```

Parameters

- `authParms`  A comma separated list of authentication parameters, or the null reference. See the class description of the SyncParms class for more information about comma separated lists.

Remarks

Only the first 255 strings are used and each string should be no longer than 128 characters (longer strings are truncated when sent to MobiLink).

See also

- “FileTransfer.getAuthenticationParms method [UltraLiteJ]” on page 158

**setLivenessTimeout method**

Sets the liveness timeout length, in seconds.

Syntax

```java
abstract void FileTransfer.setLivenessTimeout(
    int timeout
) throws ULjException
```

Parameters

- `timeout`  The new liveness timeout value.

Remarks

The liveness timeout is the length of time the server allows a remote to be idle. If the remote does not communicate with the server for 1 second, the server assumes that the remote has lost the connection, and terminates the file transfer. The remote automatically sends periodic messages to the server to keep the connection alive.

If a negative value is set, an exception is thrown. The value may be changed by the MobiLink server without notice. This change occurs if the value is set too low or too high.

The default value is 100 seconds for BlackBerry/J2SE platforms, and 240 seconds for Android platforms.

See also

- “FileTransfer.getLivenessTimeout method [UltraLiteJ]” on page 159

**setLocalFileName method**

Specifies the local file name.
Syntax
abstract void FileTransfer.setLocalFileName(String localFileName)

Parameters
- localFileName  A string specifying the local file name for the downloaded file. If the value is a null reference, fileName is used. The default is a null reference.

Remarks
For file downloads, this is the name of the downloaded file. For file uploads, this is the name of the file to upload. The file name must not include any drive or path information.

See also
- “FileTransfer.getLocalFileName method [UltraLiteJ]” on page 159
- “FileTransfer.setLocalPath method [UltraLiteJ]” on page 165

setLocalPath method
Specifies where to find or store the file in the local file system.

Syntax
abstract void FileTransfer.setLocalPath(String localPath)

Parameters
- localPath  A string specifying the local directory of the file. The default is a null reference.

Remarks
The syntax of the local directory varies among platforms:
- For a desktop, the syntax is like "C:\ulj\"
- For a BlackBerry file system, the syntax is like "file:///SDCard/ulj/"
- For a BlackBerry object store, this option is ignored
- For an Android file system, the syntax is like "/sdcard/Android/data/your.package.name/files/"

The default local directory also varies depending on the device operating system:
- For a desktop, if the localPath parameter is null, the file is stored in the current directory
- For a BlackBerry file system store, the localPath parameter has no default value, and must be explicitly set.
- For an Android file system store, the localPath parameter has no default value, and must be explicitly set.
See also
  ● “FileTransfer.getLocalPath method [UltraLiteJ]” on page 160
  ● “FileTransfer.setLocalFileName method [UltraLiteJ]” on page 164

setPassword method
  Sets the MobiLink password for the user specified with the setUserName method.

Syntax
  abstract void FileTransfer.setPassword(String password) throws ULjException

Parameters
  ● password  A password for the MobiLink user.

Remarks
  This user name and password is separate from any database user ID and password. This method is used to authenticate the application against the MobiLink server.

  The default is an empty string, suggesting no password.

See also
  ● “FileTransfer.getPassword method [UltraLiteJ]” on page 160
  ● “FileTransfer.setUserName method [UltraLiteJ]” on page 168

setRemoteKey method
  Specifies the remote key.

Syntax
  abstract void FileTransfer.setRemoteKey(String remoteKey)

Parameters
  ● remoteKey  The remote key value or null to leave it unspecified.

Remarks
  The remote key is a parameter passed to the server authenticate_file_upload script.

  The script can use this parameter to determine the name and location of the file to be stored on the server.

  If this value is unspecified, the getFileName() value is used as the remote key.

See also
  ● “FileTransfer.getRemoteKey method [UltraLiteJ]” on page 160
**setResumePartialTransfer method**

Specifies whether to resume or discard a previous partial transfer.

**Syntax**

```java
abstract void FileTransfer.setResumePartialTransfer(boolean resume)
```

**Parameters**

- **resume**  
  Set to true to resume a previous partial download, or false to discard a previous partial download.

**Remarks**

The default is true.

UltraLiteJ has the ability to restart file transfers that fail because of communication errors or user aborts through the FileTransferProgressListener object.

For file downloads, UltraLiteJ processes the download as it is received. If a download is interrupted, then the partially download file is retained and can be resumed during the next file transfer. If the file has been updated on the server, the partial download is discarded and a new download started.

For file uploads, the MobiLink server keeps partially uploaded files so that a subsequent file upload can resume a previous one. However, if the file has been updated locally, the partial upload is discarded and a new upload is started.

**See also**

- “FileTransfer.resumePartialTransfer method [UltraLiteJ]” on page 163

---

**setServerFileName method**

Specifies the name of the file on the server.

**Syntax**

```java
abstract void FileTransfer.setServerFileName(String fileName) throws ULjException
```

**Parameters**

- **fileName**  
  A string specifying the name of the file as recognized by the MobiLink server.

**Remarks**

For file downloads, this is the name of the file to download. For file uploads, this is the name of the uploaded file.

This parameter is initialized when the FileTransfer object is created.
MobiLink first searches for the file in the userName subdirectory followed by the root directory. The root download directory is specified via the MobiLink server's -ftr option, and the root upload directory is specified via the -ftru option.

fileName must not include any drive or path information, or the MobiLink server will be unable to find it. For example, "myfile.txt" is valid, but "somedir\myfile.txt", "..\myfile.txt", and "c:\myfile.txt" are all invalid.

See also
- “FileTransfer.getServerFileName method [UltraLiteJ]” on page 161

**setUserName method**

Sets the MobiLink user name that uniquely identifies the client to the MobiLink server.

**Syntax**

```java
abstract void FileTransfer.setUserName(
    String userName
) throws ULjException
```

**Parameters**

- **userName** The MobiLink user name.

**Remarks**

The MobiLink server uses this value to locate the file on the server side. The MobiLink user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink server.

This parameter is initialized when the FileTransfer object is created.

See also
- “FileTransfer.getUserName method [UltraLiteJ]” on page 162
- “FileTransfer.setPassword method [UltraLiteJ]” on page 166

**setVersion method**

Sets the synchronization script to use.

**Syntax**

```java
abstract void FileTransfer.setVersion(
    String version
) throws ULjException
```

**Parameters**

- **version** The script version.
Remarks
Each synchronization script in the consolidated database is marked with version string. The version string allows an UltraLiteJ application to choose from a set of synchronization scripts.

This parameter is initialized when the FileTransfer object is created.

See also
- “FileTransfer.getVersion method [UltraLiteJ]” on page 162

uploadFile method
Uploads the file with the specified properties of this object.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uploadFile() method</td>
<td>Uploads the file with the specified properties of this object.</td>
</tr>
<tr>
<td>uploadFile(FileTransferProgressListener) method</td>
<td>Upload the file specified by the properties of this object with progress events posted to the specified listener.</td>
</tr>
</tbody>
</table>

uploadFile() method
Uploads the file with the specified properties of this object.

Syntax

\[
abstract boolean FileTransfer.uploadFile() throws ULjException
\]

Returns
True if the upload is successful; otherwise, a ULjException is thrown and the method does not return normally.

Remarks

The file specified by the setLocalFileName and setLocalPath methods is uploaded to the MobiLink server to the file specified by the setServerFileName method using the specified stream, userName, password, and script version.

Further options can be specified using the setAuthenticationParms() and setResumePartialTransfer() methods.

A detailed result status can be fetched using the getAuthStatus(), getAuthValue(), getFileAuthCode(), isTransferredFile(), getStreamErrorCode(), and getStreamErrorMessage() methods.
uploadFile(FileTransferProgressListener) method

Upload the file specified by the properties of this object with progress events posted to the specified listener.

Syntax

abstract boolean FileTransfer.uploadFile(
   FileTransferProgressListener listener
) throws ULjException

Parameters

● listener The object that receives file transfer progress events.

Returns

True if the upload is successful; otherwise, a ULjException is thrown and the method does not return normally.

Remarks

Errors may result in no data being sent to the listener.

See also

● “FileTransfer.uploadFile method [UltraLiteJ]” on page 169

FileTransferProgressData interface

Reports file transfer progress monitoring data.

Syntax

public interface FileTransferProgressData

Members

All members of the FileTransferProgressData interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getBytesTransferred method</td>
<td>Returns the number of bytes transferred so far.</td>
</tr>
<tr>
<td>getFileSize method</td>
<td>Returns the size of the file being transferred.</td>
</tr>
<tr>
<td>getResumedAtSize method</td>
<td>Returns the point in the file where the transfer was resumed.</td>
</tr>
</tbody>
</table>

getBytesTransferred method

Returns the number of bytes transferred so far.
Syntax
abstract long FileTransferProgressData.getBytesTransferred()

Returns
The number of bytes transferred so far.

Remarks
This method counts the number of bytes transferred by the current file transfer session, and adds the bytes transferred by any previous interrupted transfers.

Subtract the value returned by the getResumedAtSize method to determine the number of bytes transferred by the current session.

See also
● “FileTransferProgressData.getResumedAtSize method [UltraLiteJ]” on page 171

getFileSize method
Returns the size of the file being transferred.

Syntax
abstract long FileTransferProgressData.getFileSize()

Returns
The size of the file in bytes.

Remarks
The returned value remains constant for the duration of the file transfer session.

getResumedAtSize method
Returns the point in the file where the transfer was resumed.

Syntax
abstract long FileTransferProgressData.getResumedAtSize()

Returns
The number of bytes previously transferred.

Remarks
The returned value remains constant for the duration of the file transfer session.
FileTransferProgressListener interface

Receives file transfer progress events.

Syntax

```java
public interface FileTransferProgressListener
```

Members

All members of the FileTransferProgressListener interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileTransferProgressed method</td>
<td>Is invoked during a file transfer to inform the user of the transfer progress.</td>
</tr>
</tbody>
</table>

Remarks

Create a new class to receive progress reports during a file transfer.

The following example illustrates a simple SyncObserver interface that implements the FileTransferProgressListener interface:

```java
class MyObserver implements FileTransferProgressListener {
    public boolean fileTransferProgressed( FileTransferProgressData data ) {
        System.out.println(
            "file transfer progress 
            + " + data.getBytesTransferred() + " bytes received = " + data.getBytesTransferred()
        );
        return false; // Always continue file transfer.
    }
    public MyObserver() {} // The default constructor.
}
```

fileTransferProgressed method

Is invoked during a file transfer to inform the user of the transfer progress.

Syntax

```java
boolean FileTransferProgressListener.fileTransferProgressed(FileTransferProgressData data)
```

Parameters

- data A FileTransferProgressData object containing the latest file transfer progress data.

Returns

This method should return true to cancel the transfer; otherwise, return false to continue.
Remarks

The listener is called under the following conditions:

- Before the first disk write
- After every disk write or every 0.5 seconds, whichever is later
- After the file download is complete

Usually, the cancel request is accepted by the UltraLiteJ API. This results in a ULjException object being thrown with the errorCode set to the ULjException.SQLE_INTERRUPTED constant.

However, if the download completed for a BlackBerry object store, UltraLiteJ does not cancel the transfer.

UltraLiteJ API methods should not be invoked during a fileTransferProgressed call.

IndexSchema interface

Specifies the schema of an index and provides constants that are useful for querying system tables.

Syntax

```java
public interface IndexSchema
```

Members

All members of the IndexSchema interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCENDING variable</td>
<td>Denotes that an index is sorted in ascending order for a column.</td>
</tr>
<tr>
<td>DESCENDING variable</td>
<td>Denotes that an index is sorted in descending order for a column.</td>
</tr>
<tr>
<td>PERSISTENT variable</td>
<td>Denotes that an index is persistent.</td>
</tr>
<tr>
<td>PRIMARY_INDEX variable</td>
<td>Denotes that an index is a primary key.</td>
</tr>
<tr>
<td>UNIQUE_INDEX variable</td>
<td>Denotes that an index is a unique index.</td>
</tr>
<tr>
<td>UNIQUE_KEY variable</td>
<td>Denotes that an index is a unique key.</td>
</tr>
</tbody>
</table>

Remarks

This interface only contains index-related constants, including flags and the sort order of indexes.

See also

- “TableSchema.SYS_INDEXES variable [UltraLiteJ]” on page 274
ASCENDING variable

Denotes that an index is sorted in ascending order for a column.

Syntax
final byte IndexSchema.ASCENDING

DESCENDING variable

Denotes that an index is sorted in descending order for a column.

Syntax
final byte IndexSchema.DESCENDING

PERSISTENT variable

Denotes that an index is persistent.

Syntax
final byte IndexSchema.PERSISTENT

Remarks
This value can be logically combined with other flags in the index_flags column of the SYS_INDEXES system table.

See also
● “TableSchema.SYS_INDEXES variable [UltraLiteJ]” on page 274

PRIMARY_INDEX variable

Denotes that an index is a primary key.

Syntax
final byte IndexSchema.PRIMARY_INDEX

Remarks
This value can be logically combined with other flags in the index_flags column of the SYS_INDEXES system table.

See also
● “TableSchema.SYS_INDEXES variable [UltraLiteJ]” on page 274
**UNIQUE_INDEX variable**

Denotes that an index is a unique index.

**Syntax**

```java
final byte IndexSchema.UNIQUE_INDEX
```

**Remarks**

This value can be logically combined with other flags in the index_flags column of the SYS_INDEXES system table.

**See also**

- “TableSchema.SYS_INDEXES variable [UltraLiteJ]” on page 274

**UNIQUE_KEY variable**

Denotes that an index is a unique key.

**Syntax**

```java
final byte IndexSchema.UNIQUE_KEY
```

**Remarks**

This value can be logically combined with other flags in the index_flags column of the SYS_INDEXES system table.

**See also**

- “TableSchema.SYS_INDEXES variable [UltraLiteJ]” on page 274

**PreparedStatement interface**

Provides methods to execute a SQL query to generate a ResultSet object or to execute a prepared SQL statement on a database.

**Syntax**

```java
public interface PreparedStatement
```

**Members**

All members of the PreparedStatement interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close method</td>
<td>Closes the PreparedStatement to release the memory resources associated with it.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>execute method</td>
<td>Executes the prepared SQL statement.</td>
</tr>
<tr>
<td>executeQuery method</td>
<td>Executes the prepared SQL SELECT statement and returns a ResultSet object.</td>
</tr>
<tr>
<td>getBlobOutputStream me-</td>
<td>Returns an OutputStream object.</td>
</tr>
<tr>
<td>thod</td>
<td></td>
</tr>
<tr>
<td>getClobWriter method</td>
<td>Returns a Writer object.</td>
</tr>
<tr>
<td>getOrdinal method</td>
<td>Returns the (base-one) ordinal for the value represented by the name.</td>
</tr>
<tr>
<td>getParameterCount meth-</td>
<td>Gets the number of input parameters for this statement.</td>
</tr>
<tr>
<td>[Android]</td>
<td></td>
</tr>
<tr>
<td>getParameterType method</td>
<td>Gets the domain type of a parameter.</td>
</tr>
<tr>
<td>[Android]</td>
<td></td>
</tr>
<tr>
<td>getPlan method</td>
<td>Returns a text-based description of the SQL query execution plan.</td>
</tr>
<tr>
<td>getPlanTree method</td>
<td>Returns a text-based description of the SQL query execution plan, rep-</td>
</tr>
<tr>
<td></td>
<td>resented as a tree.</td>
</tr>
<tr>
<td>getResultSet method</td>
<td>Returns the ResultSet object for a prepared SQL statement.</td>
</tr>
<tr>
<td>getUpdateCount method</td>
<td>Returns the number of rows inserted, updated or deleted since the last</td>
</tr>
<tr>
<td></td>
<td>execute statement.</td>
</tr>
<tr>
<td>hasResultSet method</td>
<td>Determines if the PreparedStatement object contains a ResultSet object.</td>
</tr>
<tr>
<td>set method</td>
<td>Sets a value to the host variable in the SQL statement.</td>
</tr>
<tr>
<td>setNull method</td>
<td>Sets a null value to the host variable in the SQL statement.</td>
</tr>
</tbody>
</table>

**Remarks**

The following example demonstrates how to create a PreparedStatement object, check if a SELECT statement execution creates a ResultSet object, save any ResultSet object to a local variable, and then close the PreparedStatement:

```java
// 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.
```
When a statement contains expressions, it may contain a host variable wherever a column name could appear. Host variables are entered as either a `?` character (unnamed host variables) or a `:name` (named host variable).

In the following example, there are two host variables that may be set using the PreparedStatement object, and were prepared for the SQL statement in question:

```sql
SELECT * FROM SampleTable WHERE pk > :bound AND pk < ?
```

See also
- “Connection interface [UltraLiteJ]” on page 102
- “Connection.prepareStatement method [UltraLiteJ]” on page 115

### close method
Closes the PreparedStatement to release the memory resources associated with it.

**Syntax**
```java
void PreparedStatement.close() throws ULjException
```

**Remarks**
No further methods can be used on this object. If the PreparedStatement object contains a ResultSet object, both objects are closed.

### execute method
Executes the prepared SQL statement.

**Syntax**
```java
boolean PreparedStatement.execute() throws ULjException
```

**Returns**
True if the execute statement runs successfully; otherwise, returns false.

See also
- “ResultSet interface [UltraLiteJ]” on page 191

### executeQuery method
Executes the prepared SQL SELECT statement and returns a ResultSet object.
Syntax

```java
ResultSet PreparedStatement.executeQuery() throws ULjException
```

**Returns**

The ResultSet object containing the query result of the prepared SQL SELECT statement.

**See also**

- “ResultSet interface [UltraLiteJ]” on page 191

### getBlobOutputStream method

Returns an OutputStream object.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getBlobOutputStream(int)</code></td>
<td>Returns an OutputStream object.</td>
</tr>
<tr>
<td><code>getBlobOutputStream(String)</code></td>
<td>Returns an OutputStream object.</td>
</tr>
</tbody>
</table>

### getBlobOutputStream(int) method

Returns an OutputStream object.

**Syntax**

```java
java.io.OutputStream PreparedStatement.getBlobOutputStream(int ordinal) throws ULjException
```

**Parameters**

- `ordinal` A base-one integer representing the host variable as ordered in the SQL statement.

**Returns**

The OutputStream object for the named value.

### getBlobOutputStream(String) method

Returns an OutputStream object.

**Syntax**

```java
java.io.OutputStream PreparedStatement.getBlobOutputStream(String name) throws ULjException
```
Parameters

- name  A String representing the host variable name.

Returns

The OutputStream object for the named value.

**getClobWriter method**

Returns a Writer object.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getClobWriter(int) method</td>
<td>Returns a Writer object.</td>
</tr>
<tr>
<td>getClobWriter(String) method</td>
<td>Returns a Writer object.</td>
</tr>
</tbody>
</table>

**getClobWriter(int) method**

Returns a Writer object.

**Syntax**

```java
java.io.Writer PreparedStatement.getClobWriter(
    int ordinal
) throws UException
```

**Parameters**

- ordinal  A base-one integer representing the host variable as ordered in the SQL statement.

**Returns**

The Writer object for the named value.

**getClobWriter(String) method**

Returns a Writer object.

**Syntax**

```java
java.io.Writer PreparedStatement.getClobWriter(
    String name
) throws UException
```

**Parameters**

- name  A String representing the host variable name.
Returns
The Writer object for the named value.

**getOrdinal method**
Returns the (base-one) ordinal for the value represented by the name.

**Syntax**
```java
int PreparedStatement.getOrdinal(String name) throws ULjException
```

**Parameters**
- **name** A String representing the table column name.

**Returns**
The (base-one) ordinal for the value represented by the name.

**getParameterCount method [Android]**
Gets the number of input parameters for this statement.

**Syntax**
```java
short PreparedStatement.getParameterCount() throws ULjException
```

**Returns**
The number of input parameters for this statement.

**getParameterType method [Android]**
Gets the domain type of a parameter.

**Syntax**
```java
short PreparedStatement.getParameterType(
    int ordinal
) throws ULjException
```

**Parameters**
- **ordinal** The 1-based ordinal of the parameter.

**Returns**
The domain type of the specified parameter.
**getPlan method**

Returns a text-based description of the SQL query execution plan.

**Syntax**

```java
String PreparedStatement.getPlan() throws ULjException
```

**Returns**

The String representation of the plan.

**Remarks**

This method is intended for use during development.

This plan contains the same information as is presented by the `getPlanTree` method. The difference is in the presentation.

An empty string is returned if there is no plan. Plans exist when the prepared statement is a SQL query.

The plan shows the operations used to execute the query when the plan is obtained before the associated query has been executed. Additionally, the plan shows the number of rows that each operation produced when the plan is obtained after the query has been executed. This plan can be used to gain insight about the execution of the query.

The following is an example of a plan tree, expressed as a String. It is displayed on multiple lines with '|' characters to represent the structure.

```sql
SELECT * FROM tab1, tab2 WHERE col1 > pk2
row: 2 20 10 banana
row: 3 30 10 banana
row: 4 40 10 banana
row: 4 40 30 peach
row: 5 50 10 banana
row: 5 50 30 peach
row: 5 50 40 apple
plan: root:7(inner-join:7(table-scan:5[tab1,prime_key],index-scan:7[tab2,prime_key]))
```

**See also**

- “PreparedStatement.getPlanTree method [UltraLiteJ]” on page 181

**getPlanTree method**

Returns a text-based description of the SQL query execution plan, represented as a tree.

**Syntax**

```java
String PreparedStatement.getPlanTree() throws ULjException
```

**Returns**

The String representation of the plan, represented as a tree.
Remarks

This method is intended for use during development.

This plan contains the same information as is presented by the getPlan method. The difference is in the presentation.

An empty string is returned if there is no plan. Plans exist when the prepared statement is a SQL query.

The plan shows the operations used to execute the query when the plan is obtained before the associated query has been executed. Additionally, the plan shows the number of rows that each operation produced when the plan is obtained after the query has been executed. This plan can be used to gain insight about the execution of the query.

The following is an example of a plan tree, expressed as a String. It is displayed on multiple lines with '|' characters to represent the structure.

```plaintext
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]
```

See also

- “PreparedStatement.getPlan method [UltraLiteJ]” on page 181

**getResultSet method**

Returns the ResultSet object for a prepared SQL statement.

**Syntax**

```
ResultSet PreparedStatement.getResultSet() throws ULjException
```

**Returns**

The ResultSet object containing the query result of the prepared SQL statement.

See also

- “ResultSet interface [UltraLiteJ]” on page 191
**getUpdateCount method**
Returns the number of rows inserted, updated or deleted since the last execute statement.

Syntax

```java
int PreparedStatement.getUpdateCount() throws ULjException
```

Returns

-1 when a statement cannot perform changes; otherwise, returns the number of rows that have been changed.

**hasResultSet method**
Determines if the PreparedStatement object contains a ResultSet object.

Syntax

```java
boolean PreparedStatement.hasResultSet() throws ULjException
```

Returns

True if a ResultSet was found; otherwise, returns false.

See also

- “ResultSet interface [UltraLiteJ]” on page 191

**set method**
Sets a value to the host variable in the SQL statement.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set(int, boolean)</code> method</td>
<td>Sets a boolean value to the host variable in the SQL statement that is defined by ordinal.</td>
</tr>
<tr>
<td><code>set(int, byte[])</code> method</td>
<td>Sets a byte array value to the host variable in the SQL statement that is defined by ordinal.</td>
</tr>
<tr>
<td><code>set(int, Date)</code> method</td>
<td>Sets a java.util.Date to the host variable in the SQL statement that is defined by ordinal.</td>
</tr>
<tr>
<td><code>set(int, DecimalNumber)</code> method</td>
<td>Sets a DecimalNumber object to the host variable in the SQL statement that is defined by ordinal.</td>
</tr>
<tr>
<td><code>set(int, double)</code> method</td>
<td>Sets a double value to the host variable in the SQL statement that is defined by ordinal.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>set(int, float) method</td>
<td>Sets a float value to the host variable in the SQL statement that is defined by ordinal.</td>
</tr>
<tr>
<td>set(int, int) method</td>
<td>Sets an integer value to the host variable in the SQL statement that is defined by ordinal.</td>
</tr>
<tr>
<td>set(int, long) method</td>
<td>Sets a long integer value to the host variable in the SQL statement that is defined by ordinal.</td>
</tr>
<tr>
<td>set(int, String) method</td>
<td>Sets a String value to the host variable in the SQL statement that is defined by ordinal.</td>
</tr>
<tr>
<td>set(int, UUIDValue) method</td>
<td>Sets a UUIDValue value to the host variable in the SQL statement that is defined by ordinal.</td>
</tr>
<tr>
<td>set(String, boolean) method</td>
<td>Sets a boolean value to the host variable in the SQL statement that is defined by name.</td>
</tr>
<tr>
<td>set(String, byte[]) method</td>
<td>Sets a byte array value to the host variable in the SQL statement that is defined by name.</td>
</tr>
<tr>
<td>set(String, Date) method</td>
<td>Sets a java.util.Date to the host variable in the SQL statement that is defined by name.</td>
</tr>
<tr>
<td>set(String, DecimalNumber) method</td>
<td>Sets a DecimalNumber object to the host variable in the SQL statement that is defined by name.</td>
</tr>
<tr>
<td>set(String, double) method</td>
<td>Sets a double value to the host variable in the SQL statement that is defined by name.</td>
</tr>
<tr>
<td>set(String, float) method</td>
<td>Sets a float value to the host variable in the SQL statement that is defined by name.</td>
</tr>
<tr>
<td>set(String, int) method</td>
<td>Sets an integer value to the host variable in the SQL statement that is defined by name.</td>
</tr>
<tr>
<td>set(String, long) method</td>
<td>Sets a long integer value to the host variable in the SQL statement that is defined by name.</td>
</tr>
<tr>
<td>set(String, String) method</td>
<td>Sets a String value to the host variable in the SQL statement that is defined by name.</td>
</tr>
<tr>
<td>set(String, UUIDValue) method</td>
<td>Sets a UUIDValue value to the host variable in the SQL statement that is defined by name.</td>
</tr>
</tbody>
</table>
set(int, boolean) method
Sets a boolean value to the host variable in the SQL statement that is defined by ordinal.

Syntax
```java
void PreparedStatement.set(
    int ordinal,
    boolean value
) throws ULjException
```

Parameters
- **ordinal** A base-one integer representing the host variable as ordered in the SQL statement.
- **value** The value to be set.

set(int, byte[]) method
Sets a byte array value to the host variable in the SQL statement that is defined by ordinal.

Syntax
```java
void PreparedStatement.set(
    int ordinal,
    byte[] value
) throws ULjException
```

Parameters
- **ordinal** A base-one integer representing the host variable as ordered in the SQL statement.
- **value** The value to be set.

set(int, Date) method
Sets a java.util.Date to the host variable in the SQL statement that is defined by ordinal.

Syntax
```java
void PreparedStatement.set(
    int ordinal,
    java.util.Date value
) throws ULjException
```

Parameters
- **ordinal** A base-one integer representing the host variable as ordered in the SQL statement.
- **value** The value to be set.
**set(int, DecimalNumber) method**

Sets a DecimalNumber object to the host variable in the SQL statement that is defined by ordinal.

Syntax

```java
void PreparedStatement.set(
    int ordinal,
    DecimalNumber value
) throws ULjException
```

Parameters

- **ordinal** A base-one integer representing the host variable as ordered in the SQL statement.
- **value** The DecimalNumber value to be set.

**set(int, double) method**

Sets a double value to the host variable in the SQL statement that is defined by ordinal.

Syntax

```java
void PreparedStatement.set(
    int ordinal,
    double value
) throws ULjException
```

Parameters

- **ordinal** A base-one integer representing the host variable as ordered in the SQL statement.
- **value** The value to be set.

**set(int, float) method**

Sets a float value to the host variable in the SQL statement that is defined by ordinal.

Syntax

```java
void PreparedStatement.set(int ordinal, float value) throws ULjException
```

Parameters

- **ordinal** A base-one integer representing the host variable as ordered in the SQL statement.
- **value** The value to be set.

**set(int, int) method**

Sets an integer value to the host variable in the SQL statement that is defined by ordinal.
Syntax

```java
void PreparedStatement.set(int ordinal, int value) throws ULjException
```

Parameters

- **ordinal**  
  A base-one integer representing the host variable as ordered in the SQL statement.

- **value**  
  The value to be set.

**set(int, long) method**

Sets a long integer value to the host variable in the SQL statement that is defined by ordinal.

Syntax

```java
void PreparedStatement.set(int ordinal, long value) throws ULjException
```

Parameters

- **ordinal**  
  A base-one integer representing the host variable as ordered in the SQL statement.

- **value**  
  The value to be set.

**set(int, String) method**

Sets a String value to the host variable in the SQL statement that is defined by ordinal.

Syntax

```java
void PreparedStatement.set(int ordinal, String value) throws ULjException
```

Parameters

- **ordinal**  
  A base-one integer representing the host variable as ordered in the SQL statement.

- **value**  
  The value to be set.

**set(int, UUIDValue) method**

Sets a UUIDValue value to the host variable in the SQL statement that is defined by ordinal.

Syntax

```java
void PreparedStatement.set(int ordinal, UUIDValue value) throws ULjException
```

Parameters

- **ordinal**  
  A base-one integer representing the host variable as ordered in the SQL statement.

- **value**  
  The value to be set.
Parameters

- **ordinal**  A base-one integer representing the host variable as ordered in the SQL statement.
- **value**  The value to be set.

**set(String, boolean) method**

Sets a boolean value to the host variable in the SQL statement that is defined by name.

**Syntax**

```java
void PreparedStatement.set(
    String name,
    boolean value
) throws ULjException
```

**Parameters**

- **name**  A String representing the host variable name.
- **value**  The value to be set.

**set(String, byte[]) method**

Sets a byte array value to the host variable in the SQL statement that is defined by name.

**Syntax**

```java
void PreparedStatement.set(
    String name,
    byte[] value
) throws ULjException
```

**Parameters**

- **name**  A String representing the host variable name.
- **value**  The value to be set.

**set(String, Date) method**

Sets a java.util.Date to the host variable in the SQL statement that is defined by name.

**Syntax**

```java
void PreparedStatement.set(
    String name,
    java.util.Date value
) throws ULjException
```
Parameters

- **name**  A String representing the host variable name.
- **value**  The value to be set.

**set(String, DecimalNumber) method**

Sets a DecimalNumber object to the host variable in the SQL statement that is defined by name.

**Syntax**

```java
void PreparedStatement.set(
    String name,
    DecimalNumber value
) throws ULjException
```

**Parameters**

- **name**  A String representing the host variable name.
- **value**  The DecimalNumber value to be set.

**set(String, double) method**

Sets a double value to the host variable in the SQL statement that is defined by name.

**Syntax**

```java
void PreparedStatement.set(
    String name,
    double value
) throws ULjException
```

**Parameters**

- **name**  A String representing the host variable name.
- **value**  The value to be set.

**set(String, float) method**

Sets a float value to the host variable in the SQL statement that is defined by name.

**Syntax**

```java
void PreparedStatement.set(String name, float value) throws ULjException
```

**Parameters**

- **name**  A String representing the host variable name.
- **value**  The value to be set.
**set(String, int) method**
Sets an integer value to the host variable in the SQL statement that is defined by name.

**Syntax**
```java
void PreparedStatement.set(String name, int value) throws ULjException
```

**Parameters**
- **name**  
  A String representing the host variable name.
- **value**  
  The value to be set.

**set(String, long) method**
Sets a long integer value to the host variable in the SQL statement that is defined by name.

**Syntax**
```java
void PreparedStatement.set(String name, long value) throws ULjException
```

**Parameters**
- **name**  
  A String representing the host variable name.
- **value**  
  The value to be set.

**set(String, String) method**
Sets a String value to the host variable in the SQL statement that is defined by name.

**Syntax**
```java
void PreparedStatement.set(String name, String value) throws ULjException
```

**Parameters**
- **name**  
  A String representing the host variable name.
- **value**  
  The value to be set.

**set(String, UUIDValue) method**
Sets a UUIDValue value to the host variable in the SQL statement that is defined by name.

**Syntax**
```java
void PreparedStatement.set(String name,
                         String value,
                         ) throws ULjException
```
UUIDValue value
) throws ULjException

Parameters

- **name**  A String representing the host variable name.
- **value**  The value to be set.

**setNull method**

Sets a null value to the host variable in the SQL statement.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setNull(int) method</td>
<td>Sets a null value to the host variable in the SQL statement.</td>
</tr>
<tr>
<td>setNull(String) method</td>
<td>Sets a null value to the host variable in the SQL statement that is defined by name.</td>
</tr>
</tbody>
</table>

**setNull(int) method**

Sets a null value to the host variable in the SQL statement.

**Syntax**

```java
void PreparedStatement.setNull(int ordinal) throws ULjException
```

**Parameters**

- **ordinal**  A base-one integer representing the host variable as ordered in the SQL statement.

**setNull(String) method**

Sets a null value to the host variable in the SQL statement that is defined by name.

**Syntax**

```java
void PreparedStatement.setNull(String name) throws ULjException
```

**Parameters**

- **name**  A String representing the host variable name.

**ResultSet interface**

Provides methods to traverse a table by row, and access the column data.
Syntax

```java
public interface ResultSet
```

Members

All members of the ResultSet interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>afterLast method [Android]</td>
<td>Moves the cursor after the last row.</td>
</tr>
<tr>
<td>beforeFirst method [Android]</td>
<td>Moves the cursor before the first row.</td>
</tr>
<tr>
<td>close method</td>
<td>Closes the ResultSet object to release the memory resources associated with it.</td>
</tr>
<tr>
<td>first method [Android]</td>
<td>Moves the cursor to the first row.</td>
</tr>
<tr>
<td>getBlobInputStream method</td>
<td>Returns an InputStream object for long binary types, including file-based long binaries.</td>
</tr>
<tr>
<td>getBoolean method</td>
<td>Returns a boolean value.</td>
</tr>
<tr>
<td>getBytes method</td>
<td>Returns a byte array.</td>
</tr>
<tr>
<td>getClobReader method</td>
<td>Returns a Reader object.</td>
</tr>
<tr>
<td>getDate method</td>
<td>Returns a java.util.Date object.</td>
</tr>
<tr>
<td>getDecimalNumber method</td>
<td>Returns a DecimalNumber object.</td>
</tr>
<tr>
<td>getDouble method</td>
<td>Returns a double value based on the column number.</td>
</tr>
<tr>
<td>getFloat method</td>
<td>Returns a float value.</td>
</tr>
<tr>
<td>getInt method</td>
<td>Returns an integer value.</td>
</tr>
<tr>
<td>getLong method</td>
<td>Returns a long integer value.</td>
</tr>
<tr>
<td>getOrdinal method</td>
<td>Returns the (base-one) ordinal for the value represented by a String.</td>
</tr>
<tr>
<td>ResultSetMetadata method</td>
<td>Returns a ResultSetMetadata object that contains the metadata for the ResultSet object.</td>
</tr>
<tr>
<td>getRowCount method [Android]</td>
<td>Gets the number of rows in the table.</td>
</tr>
<tr>
<td>getSize method</td>
<td>Gets the actual size of a result set column.</td>
</tr>
<tr>
<td>getString method</td>
<td>Returns a String value.</td>
</tr>
</tbody>
</table>
### Name Description

- **getUUIDValue method**  
  Returns a UUIDValue object.

- **isNull method**  
  Tests if the value at the specified column number is null.

- **last method [Android]**  
  Moves the cursor to the last row.

- **next method**  
  Fetches the next row of data in the ResultSet object.

- **previous method**  
  Fetches the previous row of data in the ResultSet object.

- **relative method [Android]**  
  Moves the cursor by an offset of rows from the current cursor position.

### Remarks

A ResultSet object is generated when the execute or executeQuery method is called on a PreparedStatement object with a SQL SELECT statement.

The following example demonstrates how to fetch a row in the ResultSet object, and access data from a specified column:

```java
// 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();
```

### See also

- “PreparedStatement interface [UltraLiteJ]” on page 175
- “PreparedStatement.executeUpdate method [UltraLiteJ]” on page 177
- “PreparedStatement.executeQuery method [UltraLiteJ]” on page 177
- “Connection interface [UltraLiteJ]” on page 102
**afterLast method [Android]**

Moves the cursor after the last row.

**Syntax**

```java
boolean ResultSet.afterLast() throws ULjException
```

**Returns**

True on success; otherwise, returns false.

**beforeFirst method [Android]**

Moves the cursor before the first row.

**Syntax**

```java
boolean ResultSet.beforeFirst() throws ULjException
```

**Returns**

True on success; otherwise, returns false.

**close method**

Closes the ResultSet object to release the memory resources associated with it.

**Syntax**

```java
void ResultSet.close() throws ULjException
```

**Remarks**

Subsequent attempts to fetch rows from a closed ResultSet object throw an error.

**first method [Android]**

Moves the cursor to the first row.

**Syntax**

```java
boolean ResultSet.first() throws ULjException
```

**Returns**

True on success; otherwise, returns false.

**getBlobInputStream method**

Returns an InputStream object for long binary types, including file-based long binaries.
Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getBlobInputStream(int) method</td>
<td>Returns an InputStream object for long binary types, including file-based long binaries.</td>
</tr>
<tr>
<td>getBlobInputStream(String) method</td>
<td>Returns an InputStream object for long binary types, including file-based long binaries.</td>
</tr>
</tbody>
</table>

**getBlobInputStream(int) method**

Returns an InputStream object for long binary types, including file-based long binaries.

**Syntax**

```java
java.io.InputStream ResultSet.getBlobInputStream(
    int ordinal
) throws ULjException
```

**Parameters**

- **ordinal** A base-one integer representing the column number as ordered in the SQL statement.

**Returns**

The InputStream object representation of the named value.

**getBlobInputStream(String) method**

Returns an InputStream object for long binary types, including file-based long binaries.

**Syntax**

```java
java.io.InputStream ResultSet.getBlobInputStream(
    String name
) throws ULjException
```

**Parameters**

- **name** A String representing the table column name.

**Returns**

The InputStream object representation of the named value.

**getBoolean method**

Returns a boolean value.
### Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getBoolean(int) method</td>
<td>Returns a boolean value.</td>
</tr>
<tr>
<td>getBoolean(String) method</td>
<td>Returns a boolean value.</td>
</tr>
</tbody>
</table>

### getBoolean(int) method

Returns a boolean value.

**Syntax**

```java
boolean ResultSet.getBoolean(int ordinal) throws ULjException
```

**Parameters**

- `ordinal` A base-one integer representing the column number as ordered in the SQL statement.

**Returns**

The boolean representation of the named value.

### getBoolean(String) method

Returns a boolean value.

**Syntax**

```java
boolean ResultSet.getBoolean(String name) throws ULjException
```

**Parameters**

- `name` A String representing the table column name in the ResultSet object.

**Returns**

The boolean representation of the named value.

### getBytes method

Returns a byte array.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getBytes(int) method</td>
<td>Returns a byte array.</td>
</tr>
<tr>
<td>getBytes(String) method</td>
<td>Returns a byte array.</td>
</tr>
</tbody>
</table>
**getBytes(int) method**

Returns a byte array.

Syntax

```
byte[] ResultSet.getBytes(int ordinal) throws ULjException
```

Parameters

- **ordinal** A base-one integer representing the column number as ordered in the SQL statement.

Returns

The byte array representation of the named value.

**getBytes(String) method**

Returns a byte array.

Syntax

```
byte[] ResultSet.getBytes(String name) throws ULjException
```

Parameters

- **name** A String representing the table column name.

Returns

The byte array representation of the named value.

**getClobReader method**

Returns a Reader object.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getClobReader(int) method</td>
<td>Returns a Reader object.</td>
</tr>
<tr>
<td>getClobReader(String) method</td>
<td>Returns a Reader object.</td>
</tr>
</tbody>
</table>

**getClobReader(int) method**

Returns a Reader object.

Syntax

```
java.io.Reader ResultSet.getClobReader(int ordinal) throws ULjException
```
Parameters

- **ordinal** A base-one integer representing the column number as ordered in the SQL statement.

Returns

The Reader object representation of the named value.

**getClobReader(String) method**

Returns a Reader object.

Syntax

```java
java.io.Reader ResultSet.getClobReader(String name) throws ULjException
```

Parameters

- **name** A String representing the table column name.

Returns

The Reader object representation of the named value.

**getDate method**

Returns a java.util.Date object.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDate(int) method</td>
<td>Returns a java.util.Date object.</td>
</tr>
<tr>
<td>getDate(String) method</td>
<td>Returns a java.util.Date object.</td>
</tr>
</tbody>
</table>

**getDate(int) method**

Returns a java.util.Date object.

Syntax

```java
java.util.Date ResultSet.getDate(int ordinal) throws ULjException
```

Parameters

- **ordinal** A base-one integer representing the column number as ordered in the SQL statement.

Returns

The java.util.Date object representation of the named value.
**getDateTime(String) method**

Returns a java.util.Date object.

**Syntax**

```java
java.util.Date ResultSet.getDateTime(String name) throws ULjException
```

**Parameters**

- `name` A String representing the table column name.

**Returns**

The java.util.date object representation of the named value.

---

**getDecimalNumber method**

Returns a DecimalNumber object.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDecimalNumber(int) method</td>
<td>Returns a DecimalNumber object.</td>
</tr>
<tr>
<td>getDecimalNumber(String) method</td>
<td>Returns a DecimalNumber object.</td>
</tr>
</tbody>
</table>

**getDecimalNumber(int) method**

Returns a DecimalNumber object.

**Syntax**

```java
DecimalNumber ResultSet.getDecimalNumber(
    int ordinal
) throws ULjException
```

**Parameters**

- `ordinal` A base-one integer representing the column number as ordered in the SQL statement.

**Returns**

The DecimalNumber object representation of the named value.

**See also**

- “DecimalNumber interface [UltraLiteJ]” on page 142
getDecimalNumber(String) method

Returns a DecimalNumber object.

Syntax

```
DecimalNumber ResultSet.getDecimalNumber(
    String name
) throws ULjException
```

Parameters

- **name** A String representing the table column name.

Returns

The DecimalNumber object representation of the named value.

See also

- “DecimalNumber interface [UltraLiteJ]” on page 142

getDouble method

Returns a double value based on the column number.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDouble(int) method</td>
<td>Returns a double value based on the column number.</td>
</tr>
<tr>
<td>getDouble(String) method</td>
<td>Returns a double value based on the column name.</td>
</tr>
</tbody>
</table>

gelloworld(int) method

Returns a double value based on the column number.

Syntax

```
double ResultSet.getDouble(int ordinal) throws ULjException
```

Parameters

- **ordinal** A base-one integer representing the column number as ordered in the SQL statement.

Returns

The double representation of the named value.
**getDouble(String) method**

Returns a double value based on the column name.

**Syntax**

double ResultSet.getDouble(String name) throws ULjException

**Parameters**

- **name**  
  A String representing the table column name.

**Returns**

The double representation of the named value.

---

**getFloat method**

Returns a float value.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getFloat(int) method</td>
<td>Returns a float value.</td>
</tr>
<tr>
<td>getFloat(String) method</td>
<td>Returns a float value.</td>
</tr>
</tbody>
</table>

---

**getFloat(int) method**

Returns a float value.

**Syntax**

float ResultSet.getFloat(int ordinal) throws ULjException

**Parameters**

- **ordinal**  
  A base-one integer representing the column number as ordered in the SQL statement.

**Returns**

The float representation of the named value.

---

**getFloat(String) method**

Returns a float value.

**Syntax**

float ResultSet.getFloat(String name) throws ULjException
Parameters

- **name**  A String representing the table column name.

Returns

The float representation of the named value.

**getInt method**

Returns an integer value.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getInt(int) method</td>
<td>Returns an integer value.</td>
</tr>
<tr>
<td>getInt(String) method</td>
<td>Returns an integer value.</td>
</tr>
</tbody>
</table>

**getInt(int) method**

Returns an integer value.

**Syntax**

```java
int ResultSet.getInt(int ordinal) throws ULjException
```

**Parameters**

- **ordinal**  A base-one integer representing the column number as ordered in the SQL statement.

**Returns**

The integer representation of the named value.

**getInt(String) method**

Returns an integer value.

**Syntax**

```java
int ResultSet.getInt(String name) throws ULjException
```

**Parameters**

- **name**  A String representing the table column name.

**Returns**

The integer representation of the named value.
**getLong method**

Returns a long integer value.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getLong(int) method</td>
<td>Returns a long integer value.</td>
</tr>
<tr>
<td>getLong(String) method</td>
<td>Returns a long integer value.</td>
</tr>
</tbody>
</table>

**getLong(int) method**

Returns a long integer value.

**Syntax**

```java
long ResultSet.getLong(int ordinal) throws ULjException
```

**Parameters**

- **ordinal**  A base-one integer representing the column number as ordered in the SQL statement.

**Returns**

The long integer representation of the named value.

**getLong(String) method**

Returns a long integer value.

**Syntax**

```java
long ResultSet.getLong(String name) throws ULjException
```

**Parameters**

- **name**  A String representing the table column name.

**Returns**

The long integer representation of the named value.

**getOrdinal method**

Returns the (base-one) ordinal for the value represented by a String.

**Syntax**

```java
int ResultSet.getOrdinal(String name) throws ULjException
```
Parameters

- **name**  A String representing the table column name.

Returns

The ordinal value.

**getResultSetMetadata method**

Returns a ResultSetMetadata object that contains the metadata for the ResultSet object.

Syntax

```
ResultSetMetadata ResultSet.getResultSetMetadata() throws ULjException
```

Returns

The ResultSetMetadata object.

**getRowCount method [Android]**

Gets the number of rows in the table.

Syntax

```
long ResultSet.getRowCount(long threshold) throws ULjException
```

Parameters

- **threshold**  The limit on the number of rows to count. 0 indicates no limit.

Returns

The number of rows in the table.

Remarks

This method is equivalent to executing "SELECT COUNT(*) FROM table".

**getSize method**

Gets the actual size of a result set column.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSize(int) method</td>
<td>Gets the actual size of a result set column.</td>
</tr>
<tr>
<td>getSize(String) method</td>
<td>Gets the actual size of a result set column.</td>
</tr>
</tbody>
</table>
**getSize(int) method**

Gets the actual size of a result set column.

**Syntax**

```java
int ResultSet.getSize(int ordinal) throws ULEException
```

**Parameters**

- `ordinal` A base-one integer representing the column number as ordered in the SQL statement.

**Returns**

The actual size of a result set column.

**getSize(String) method**

Gets the actual size of a result set column.

**Syntax**

```java
int ResultSet.getSize(String name) throws ULEException
```

**Parameters**

- `name` A String representing the table column name.

**Returns**

The actual size of a result set column.

**getString method**

Returns a String value.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getString(int) method</code></td>
<td>Returns a String value.</td>
</tr>
<tr>
<td><code>getString(String) method</code></td>
<td>Returns a String value.</td>
</tr>
</tbody>
</table>

**getString(int) method**

Returns a String value.

**Syntax**

```java
String ResultSet.getString(int ordinal) throws ULEException
```
Parameters

- **ordinal**  A base-one integer representing the column number as ordered in the SQL statement.

Returns

The String representation of the named value.

**getString(String) method**

Returns a String value.

Syntax

```java
String ResultSet.getString(String name) throws ULjException
```

Parameters

- **name**  A String representing the table column name.

Returns

The String representation of the named value.

**getUUIDValue method**

Returns a UUIDValue object.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getUUIDValue(int) method</td>
<td>Returns a UUIDValue object.</td>
</tr>
<tr>
<td>getUUIDValue(String) method</td>
<td>Returns a UUIDValue object.</td>
</tr>
</tbody>
</table>

**getUUIDValue(int) method**

Returns a UUIDValue object.

Syntax

```java
UUIDValue ResultSet.getUUIDValue(int ordinal) throws ULjException
```

Parameters

- **ordinal**  A base-one integer representing the column number as ordered in the SQL statement.

Returns

The UUIDValue object representation of the named value.
getUUIDValue(String) method
Returns a UUIDValue object.

Syntax
UUIDValue ResultSet.getUUIDValue(String name) throws ULjException

Parameters
- name A String representing the table column name.

Returns
The UUIDValue object representation of the named value.

isNull method
Tests if the value at the specified column number is null.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isNull(int) method</td>
<td>Tests if the value at the specified column number is null.</td>
</tr>
<tr>
<td>isNull(String) method</td>
<td>Tests if the value at the specified column name is null.</td>
</tr>
</tbody>
</table>

isNull(int) method
Tests if the value at the specified column number is null.

Syntax
boolean ResultSet.isNull(int ordinal) throws ULjException

Parameters
- ordinal A base-one integer representing the column number as ordered in the SQL statement.

Returns
True if the value is null; otherwise, returns false.
isNull(String) method
Tests if the value at the specified column name is null.

Syntax
boolean ResultSet.isNull(String name) throws ULjException

Parameters
• name A String representing the table column name.

Returns
True if the value is null; otherwise, returns false.

last method [Android]
Moves the cursor to the last row.

Syntax
boolean ResultSet.last() throws ULjException

Returns
True on success; otherwise, returns false.

next method
Fetches the next row of data in the ResultSet object.

Syntax
boolean ResultSet.next() throws ULjException

Returns
True when the next row is successfully fetched; otherwise, returns false.

See also
• “ResultSetMetadata interface [UltraLiteJ]” on page 209

previous method
Fetches the previous row of data in the ResultSet object.

Syntax
boolean ResultSet.previous() throws ULjException
Returns
True when the previous row is successfully fetched; otherwise, returns false.

**relative method [Android]**
Moves the cursor by an offset of rows from the current cursor position.

**Syntax**
```java
boolean ResultSet.relative(int offset) throws ULjException
```

**Parameters**
- **offset** The number of rows to move.

Returns
True on success; otherwise, returns false.

**ResultSetMetadata interface**
Associated with a ResultSet object and contains a method that provides column information.

**Syntax**
```java
public interface ResultSetMetadata
```

**Members**
All members of the ResultSetMetadata interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAliasName method</td>
<td>Returns the alias name for a column.</td>
</tr>
<tr>
<td>getColumnCount method</td>
<td>Returns the total number of columns in the ResultSet object.</td>
</tr>
<tr>
<td>getCorrelationName method</td>
<td>Returns the correlation name for a column.</td>
</tr>
<tr>
<td>getDomainName method</td>
<td>Returns the name of the domain.</td>
</tr>
<tr>
<td>getDomainPrecision method</td>
<td>Returns the precision of the domain value.</td>
</tr>
<tr>
<td>getDomainScale method</td>
<td>Returns the scale of the domain value.</td>
</tr>
<tr>
<td>getDomainSize method</td>
<td>Returns the size of the domain value.</td>
</tr>
<tr>
<td>getDomainType method</td>
<td>Returns the type of the domain.</td>
</tr>
<tr>
<td>getQualifiedName method</td>
<td>Returns the qualified name for a column.</td>
</tr>
</tbody>
</table>
**Name** | **Description**
---|---
getTableColumnName method | Returns the column name in the table or the derived table.
getTableName method | Returns the table name for a column.
getWrittenName method | Returns the written name for a column.

**Remarks**

This interface is obtained with the ResultSet.getResultSetMetadata method.

When a column in the select list of the ResultSet object is a simple name or a compound name (tablename.column-name, correlation-name.column-name), then, the following information about that name can be extracted if it exists:

- Alias name
- Correlation name
- Qualified version of the name
- Table name
- The name as written

For each column in the select list of the ResultSet object, the following information about the column domain can be obtained:

- Column Type: an integer from the Domain interface
- Name of the domain
- Size of the domain, for VARCHAR and BINARY domains
- Scale and precision, for NUMERIC domains

**See also**

- “Domain interface [UltraLiteJ]” on page 145
- “ResultSet interface [UltraLiteJ]” on page 191
- “ResultSet.getResultSetMetadata method [UltraLiteJ]” on page 204

**getAliasName method**

Returns the alias name for a column.

**Syntax**

```java
String ResultSetMetadata.getAliasName(int column_no) throws ULjException
```
Parameters

- column_no  The (base 1) number of the column in the select list.

Returns

Null if there is not an alias name for the column; otherwise, returns the alias name for a column.

Remarks

The alias name ([AS] name) may be specified to reference a column.

**getColumnCount method**

Returns the total number of columns in the ResultSet object.

Syntax

```java
int ResultSetMetadata.getColumnCount() throws ULjException
```

Returns

The number of columns.

**getCorrelationName method**

Returns the correlation name for a column.

Syntax

```java
String ResultSetMetadata.getCorrelationName(
    int column_no
) throws ULjException
```

Parameters

- column_no  The (base 1) number of the column in the select list.

Returns

Null if there is no correlation name for the column; otherwise, returns the correlation name for a column.

Remarks

The correlation name specified ([AS] correlation-name) in the FROM clause to specify a table expression, such as a derived table.

**getDomainName method**

Returns the name of the domain.
Syntax

```java
String ResultSetMetadata.getDomainName(
    int column_no
) throws ULjException
```

Parameters

- `column_no` The (base 1) number of the column in the select list.

Returns

The domain name.

**getDomainPrecision method**

Returns the precision of the domain value.

Syntax

```java
int ResultSetMetadata.getDomainPrecision(
    int column_no
) throws ULjException
```

Parameters

- `column_no` The (base 1) number of the column in the select list.

Returns

The precision.

**getDomainScale method**

Returns the scale of the domain value.

Syntax

```java
int ResultSetMetadata.getDomainScale(int column_no) throws ULjException
```

Parameters

- `column_no` The (base 1) number of the column in the select list.

Returns

The scale.

**getDomainSize method**

Returns the size of the domain value.
ResultSetMetadata interface

Syntax

```java
int ResultSetMetadata.getDomainSize(int column_no) throws ULjException
```

Parameters

- **column_no**  The (base 1) number of the column in the select list.

Returns

The size.

**getDomainType method**

Returns the type of the domain.

Syntax

```java
short ResultSetMetadata.getDomainType(int column_no) throws ULjException
```

Parameters

- **column_no**  The (base 1) number of the column in the select list.

Returns

The domain type expressed as an integer.

**getQualifiedName method**

Returns the qualified name for a column.

Syntax

```java
String ResultSetMetadata.getQualifiedName(int column_no) throws ULjException
```

Parameters

- **column_no**  The (base 1) number of the column in the select list.

Returns

Null if there is no qualified name for the column; otherwise, returns the qualified name for a column.

Remarks

When the ResultSet column refers to a column in a table, the name returned is a compound name consisting of a correlation name (or table name if the correlation name was not given) followed by the name of the column in the table.

When the ResultSet column does not refer to a column in a table and an alias was specified, the alias name is returned.
**getTableColumnName method**

Returns the column name in the table or the derived table.

**Syntax**

```java
String ResultSetMetadata.getTableColumnName(int column_no) throws ULjException
```

**Parameters**

- **column_no**  
The (base 1) number of the column in the select list.

**Returns**

Null if there is no table name for the column; otherwise, returns the table name for a column.

**getTableName method**

Returns the table name for a column.

**Syntax**

```java
String ResultSetMetadata.getTableName(int column_no) throws ULjException
```

**Parameters**

- **column_no**  
The (base 1) number of the column in the select list.

**Returns**

Null if there is no table name for the column; otherwise, returns the table name for a column.

**Remarks**

The table is the name of the table that ResultSet column references (possibly as through a correlation name).

**getWrittenName method**

Returns the written name for a column.

**Syntax**

```java
String ResultSetMetadata.getWrittenName(int column_no) throws ULjException
```

**Parameters**

- **column_no**  
The (base 1) number of the column in the select list.
Returns
Null if there is no written name for the column; otherwise, returns the written name for a column.

Remarks
The written name is the simple or compound name specified as the indicated column in the select list.

SISListener interface [BlackBerry]
Listens for server-initiated synchronization messages.

Syntax
public interface SISListener

Members
All members of the SISListener interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startListening method</td>
<td>Creates and starts the listening thread.</td>
</tr>
<tr>
<td>stopListening method</td>
<td>Stops the listening thread.</td>
</tr>
</tbody>
</table>

Remarks
The application creates an instance of the SISListener interface using the appropriate DatabaseManager.createSISHTTPLListener method.

See also
● “DatabaseManager.createSISHTTPLListener method [BlackBerry] [UltraLiteJ]” on page 141

startListening method
Creates and starts the listening thread.

Syntax
void SISListener.startListening()

stopListening method
Stops the listening thread.

Syntax
void SISListener.stopListening()
SISRequestHandler interface [BlackBerry]
Handles server-initiated synchronization requests.

Syntax

```java
public interface SISRequestHandler
```

Members
All members of the SISRequestHandler interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onError method</td>
<td>Handles server-initiated synchronization errors that occur while listening.</td>
</tr>
<tr>
<td>onRequest method</td>
<td>Handles server-initiated synchronization requests on worker threads.</td>
</tr>
</tbody>
</table>

See also
- “SISListener interface [BlackBerry] [UltraLiteJ]” on page 215

onError method
Handles server-initiated synchronization errors that occur while listening.

Syntax

```java
void SISRequestHandler.onError(String text)
```

Parameters
- `text` The string representation of the exception.

Remarks
To stop listening, explicitly call the stopListening method in the SISListener interface.

onRequest method
Handles server-initiated synchronization requests on worker threads.

Syntax

```java
void SISRequestHandler.onRequest(String text)
```

Parameters
- `text` The string sent by the request.
SQLInfo interface [Android]

Represents information about an executed SQL statement.

Syntax

public interface SQLInfo

Members

All members of the SQLInfo interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getMessage method</td>
<td>Returns the message associated with the SQL code.</td>
</tr>
<tr>
<td>getParameter method</td>
<td>Returns the specified parameter.</td>
</tr>
<tr>
<td>getParameterCount method</td>
<td>Returns the number of parameters of the message.</td>
</tr>
<tr>
<td>getSQLCode method</td>
<td>Gets the code (SQLCODE) for the executed SQL statement.</td>
</tr>
<tr>
<td>getSQLCount method</td>
<td>Returns a value that depends on the result of the executed SQL statement.</td>
</tr>
</tbody>
</table>

getMessage method

Returns the message associated with the SQL code.

Syntax

String SQLInfo.getMessage()

ggetParameter method

Returns the specified parameter.

Syntax

String SQLInfo.getParameter(short param_no)

Parameters

- param_no  A one-based parameter number.

Returns

The parameter.
**getParameterCount method**

Returns the number of parameters of the message.

**Syntax**

```
short SQLInfo.getParameterCount()
```

**Returns**

The number of parameters.

**getSQLCode method**

Gets the code (SQLCODE) for the executed SQL statement.

**Syntax**

```
int SQLInfo.getSQLCode()
```

**Returns**

The SQLCODE value.

**getSQLCount method**

Returns a value that depends on the result of the executed SQL statement.

**Syntax**

```
int SQLInfo.getSQLCount()
```

**Returns**

The number of rows affected by the statement after an INSERT, UPDATE, or DELETE statement is executed. The return value is the offset into the associated dynamic SQL statement that corresponds to the error if a SQLE_SYNTAX_ERROR occurs.

---

**StreamHTTPParms interface**

Represents HTTP stream parameters that define how to communicate with a MobiLink server using HTTP.

**Syntax**

```
public interface StreamHTTPParms
```

**Derived classes**

- “StreamHTTPSParms interface [UltraLiteJ]” on page 231
Members

All members of the StreamHTTPParms interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addCustomHTTPHeader method [BlackBerry]</td>
<td>Adds a message header to each HTTP request.</td>
</tr>
<tr>
<td>getCustomHTTPHeaders method [BlackBerry]</td>
<td>Returns a Hashtable object containing the HTTP headers specified using the addCustomHTTPHeader method.</td>
</tr>
<tr>
<td>getE2eePublicKey method [BlackBerry]</td>
<td>Returns the name of the file containing the end-to-end public key.</td>
</tr>
<tr>
<td>getE2eeType method [BlackBerry]</td>
<td>Returns the end-to-end encryption type that is in use.</td>
</tr>
<tr>
<td>getHost method</td>
<td>Returns the host name of the MobiLink server.</td>
</tr>
<tr>
<td>getHTTPPassword method [BlackBerry]</td>
<td>Returns the HTTP password.</td>
</tr>
<tr>
<td>getHTTPUserId method [BlackBerry]</td>
<td>Returns the HTTP user ID.</td>
</tr>
<tr>
<td>getOutputBufferSize method</td>
<td>Returns the size, in bytes, of the output buffer used to store data before it is sent to the MobiLink server.</td>
</tr>
<tr>
<td>getPort method</td>
<td>Returns the port number used to connect to the MobiLink server.</td>
</tr>
<tr>
<td>getURLSuffix method</td>
<td>Returns the URL suffix of the MobiLink server.</td>
</tr>
<tr>
<td>isRestartable method</td>
<td>Determines whether restartable HTTP is used.</td>
</tr>
<tr>
<td>setE2eePublicKey method [BlackBerry]</td>
<td>Specifies the name of the file containing the end-to-end public key.</td>
</tr>
<tr>
<td>setE2eeType method [BlackBerry]</td>
<td>Specifies the type of end-to-end encryption to use.</td>
</tr>
<tr>
<td>setHost method</td>
<td>Sets the host name of the MobiLink server.</td>
</tr>
<tr>
<td>setHTTPUserIdAndPassword method [BlackBerry]</td>
<td>Sets the user ID and password used for Basic HTTP authentication as described in RFC 2617.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>setOutputBufferSize method</td>
<td>Sets the size, in bytes, of the output buffer used to store data before it is sent to the MobiLink server.</td>
</tr>
<tr>
<td>setPort method</td>
<td>Sets the port number used to connect to the MobiLink server.</td>
</tr>
<tr>
<td>setRestartable method</td>
<td>Enables or disables restartable HTTP.</td>
</tr>
<tr>
<td>setURLSuffix method</td>
<td>Specifies the URL suffix to connect to the MobiLink server.</td>
</tr>
<tr>
<td>setZlibCompression method</td>
<td>Enables or disables ZLIB compression.</td>
</tr>
<tr>
<td>setZlibDownloadWindowSize method</td>
<td>Sets the download window size for ZLIB compression.</td>
</tr>
<tr>
<td>setZlibUploadWindowSize method</td>
<td>Sets the upload window size for ZLIB compression.</td>
</tr>
<tr>
<td>zlibCompressionEnabled method</td>
<td>Determines if ZLIB compression is enabled.</td>
</tr>
<tr>
<td>E2EE_RSA variable [BlackBerry]</td>
<td>Specifies RSA-based end-to-end encryption when passed to the setE2eeType method.</td>
</tr>
</tbody>
</table>

**Remarks**

The following example sets the stream parameters to communicate with a MobiLink server on host name "MyMLHost". The server started with the following parameters: ":-x http(port=1234)"

```java
SyncParms syncParms = myConnection.createSyncParms(
    SyncParms.HTTP_STREAM,
    "MyUniqueMLUserID",
    "MyMLScriptVersion"
);
StreamHTTPParms httpParms = syncParms.getInputStreamParms();
httpParms.setHost("MyMLHost");
httpParms.setPort(1234);
```

Instances implementing this interface are returned by the SyncParms.getInputStreamParms method.

**See also**

- “SyncParms class [UltraLiteJ]” on page 242
- “SyncParms.getInputStreamParms method [UltraLiteJ]” on page 248

**addCustomHTTPHeader method [BlackBerry]**

Adds a message header to each HTTP request.

**Syntax**

```java
void StreamHTTPParms.addCustomHTTPHeader(String name, String value)
```
Parameters

- **name**  The header name.
- **value**  The header value.

Remarks

If this method is called more than once with the same name parameter, the values are concatenated into a comma-separated list.

The following standard headers cannot be changed using this method:

- Connection
- Content-Length
- User-Agent
- Content-Type

Other headers can be modified by the Java VM.

Specify custom cookies by calling this method with Cookie as the header name.

For Android smartphones, use the setExtraParameters method to specify a custom_header parameter.

See also

- “StreamHTTPParms.setExtraParameters method [Android] [UltraLiteJ]” on page 225

**getCustomHTTPHeaders method [BlackBerry]**

Returns a HasTable object containing the HTTP headers specified using the addCustomHTTPHeader method.

**Syntax**

```java
java.util.HasTable StreamHTTPParms.getCustomHTTPHeaders()
```

**Returns**

A HasTable containing the HTTP headers specified by the addCustomHTTPHeader method. The key is the header name, and the value is the header value.

See also

- “StreamHTTPParms.addCustomHTTPHeader method [BlackBerry] [UltraLiteJ]” on page 220

**getE2eePublicKey method [BlackBerry]**

Returns the name of the file containing the end-to-end public key.
String `StreamHTTPParms.getE2eePublicKey()`

Returns
The name of the file containing the end-to-end public key.

See also
- “StreamHTTPParms.setE2eePublicKey method [BlackBerry] [UltraLiteJ]” on page 224

**getE2eeType method [BlackBerry]**

Returns the end-to-end encryption type that is in use.

Syntax
```java
short StreamHTTPParms.getE2eeType()
```

Returns
The end-to-end encryption type.

**getExtraParameters method [Android]**

Gets the extra MobiLink client network protocol options.

Syntax
```java
String StreamHTTPParms.getExtraParameters()
```

Returns
The extra protocol options that have been set.

**getHost method**

Returns the host name of the MobiLink server.

Syntax
```java
String StreamHTTPParms.getHost()
```

Returns
The name of the host.

See also
- “StreamHTTPParms.setHost method [UltraLiteJ]” on page 226
- “StreamHTTPParms.getPort method [UltraLiteJ]” on page 224
- “StreamHTTPParms.setPort method [UltraLiteJ]” on page 227
getHTTPPassword method [BlackBerry]

Returns the HTTP password.

Syntax

```java
String StreamHTTPParms.getHTTPPassword()
```

Returns

The password previously set by the setHTTPUserIdAndPassword method.

See also

- “StreamHTTPParms.setHTTPUserIdAndPassword method [BlackBerry] [UltraLiteJ]” on page 226

getHTTPUserId method [BlackBerry]

Returns the HTTP user ID.

Syntax

```java
String StreamHTTPParms.getHTTPUserId()
```

Returns

The user ID previously set by the setHTTPUserIdAndPassword method.

See also

- “StreamHTTPParms.setHTTPUserIdAndPassword method [BlackBerry] [UltraLiteJ]” on page 226

getOutputBufferSize method

Returns the size, in bytes, of the output buffer used to store data before it is sent to the MobiLink server.

Syntax

```java
int StreamHTTPParms.getOutputStreamBufferSize()
```

Returns

The integer containing the buffer size.

Remarks

Increasing this value may reduce the number of network flushes needed to send a large upload at the cost of increased memory use. In HTTP, each flush sends a large (approximately 250 bytes) HTTP header; reducing the number of flushes can reduce the bandwidth use.

See also

- “StreamHTTPParms.setOutputBufferSize method [UltraLiteJ]” on page 227
**getPort method**

Returns the port number used to connect to the MobiLink server.

Syntax

```java
int StreamHTTPParms.getPort()
```

Returns

The port number of MobiLink server.

See also

- “StreamHTTPParms.setPort method [UltraLiteJ]” on page 227

**getURLSuffix method**

Returns the URL suffix of the MobiLink server.

Syntax

```java
String StreamHTTPParms.getURLSuffix()
```

Returns

The String containing the URL suffix.

See also

- “StreamHTTPParms.setURLSuffix method [UltraLiteJ]” on page 228

**isRestartable method**

Determines whether restartable HTTP is used.

Syntax

```java
boolean StreamHTTPParms.isRestartable()
```

Returns

True if restartable HTTP is enabled; otherwise, returns false.

See also

- “StreamHTTPParms.setRestartable method [UltraLiteJ]” on page 228

**setE2eePublicKey method [BlackBerry]**

Specifies the name of the file containing the end-to-end public key.
Syntax
```
void StreamHTTPParms.setE2eePublicKey(String public_key)
```

Parameters
- **public_key** The name of the RSA public key file used in the encryption. This name must be DER-encoded.

Remarks
By default, this value is null, indicating that end-to-end encryption is not used.

This method corresponds to the e2 ee_public_key protocol option.

The public key can be stored on either an SD card or the device object store.

When using an SD card, the public_key parameter should have the following form:
```
file:///path
```

`path` is the absolute path to the file on the card. For example, `file:///SDCard/ulj/public_key.der` is a valid public_key parameter.

When using the object store, use the UltraLite Java Edition Database Transfer utility to download the file from the MobiLink server. The key must have a `der` file extension.

See also
- “StreamHTTPParms.getE2eePublicKey method [BlackBerry] [UltraLiteJ]” on page 221
- “e2ee_public_key” [MobiLink - Client Administration]
- “UltraLite Java Edition Database Transfer utility” [UltraLite - Database Management and Reference]

### setE2eeType method [BlackBerry]

Specifies the type of end-to-end encryption to use.

Syntax
```
void StreamHTTPParms.setE2eeType(short type)
```

Parameters
- **type** Pass the StreamHTTPParms.E2EE_RSA constant. StreamHTTPParms.E2EE_RSA is the default.

See also
- “StreamHTTPParms.E2EE_RSA variable [BlackBerry] [UltraLiteJ]” on page 230

### setExtraParameters method [Android]

Sets extra MobiLink client network protocol options.
Syntax

```java
void StreamHTTPParms.setExtraParameters(String parms)
```

Parameters

- **parms** A semicolon delimited list of protocol options.

Remarks

These options are appended to the list that is built from the settings resulting from the methods of this class.

Options that are set by this method override the same options that are set by other methods. For example, if "host=abc" is contained in the extra parameters, and the setHost("xyz") method is called, then the host option is "abc".

**setHost method**

Sets the host name of the MobiLink server.

Syntax

```java
void StreamHTTPParms.setHost(String v)
```

Parameters

- **v** The name of the host.

Remarks

The default is null, which indicates a localhost.

See also

- “StreamHTTPParms.getHost method [UltraLiteJ]” on page 222
- “StreamHTTPParms.getPort method [UltraLiteJ]” on page 224
- “StreamHTTPParms.setPort method [UltraLiteJ]” on page 227

**setHTTPUserIdAndPassword method [BlackBerry]**

Sets the user ID and password used for Basic HTTP authentication as described in RFC 2617.

Syntax

```java
void StreamHTTPParms.setHTTPUserIdAndPassword(String userid, String password)
```

Parameters

- **userid** The userid to use.
password  The password to use.

Remarks
With Basic HTTP authentication, passwords are included in HTTP headers in clear text; however, you can use HTTPS to encrypt the headers and protect this password.

See also
- RFC 2617: HTTP Authentication

**setOutputBufferSize method**
Sets the size, in bytes, of the output buffer used to store data before it is sent to the MobiLink server.

Syntax
```java
void StreamHTTPParms.setOutputBufferSize(int size)
```

Parameters
- size  The new buffer size.

Remarks
The default is 4096. Valid values range between 512 and 32768. Increasing this value may cause the Java runtime to send chunked HTTP, which the MobiLink server cannot process.

If the MobiLink server outputs an "unknown transfer encoding" error, try decreasing this value.

See also
- “StreamHTTPParms.getOutputBufferSize method [UltraLiteJ]” on page 223

**setPort method**
Sets the port number used to connect to the MobiLink server.

Syntax
```java
void StreamHTTPParms.setPort(int v)
```

Parameters
- v  A port number ranging from 1 to 65535. Out of range values revert to default value.

Remarks
The default port is 80 for HTTP synchronizations and 443 for HTTPS synchronizations.

See also
- “StreamHTTPParms.getPort method [UltraLiteJ]” on page 224
**setRestartable method**

Enables or disables restartable HTTP.

**Syntax**

```java
void StreamHTTPParms.setRestartable(boolean isRestartable)
```

**Parameters**

- **isRestartable** Set to true to enable restartable HTTP. The default value is false.

**Remarks**

When restartable HTTP is enabled, UltraLiteJ can tolerate network interruptions so that synchronizations do not fail as often on unreliable networks.

To use restartable HTTP, both UltraLiteJ and the MobiLink server must have applied CR#690250.

**See also**

- “StreamHTTPParms.isRestartable method [UltraLiteJ]” on page 224

**setURLSuffix method**

Specifies the URL suffix to connect to the MobiLink server.

**Syntax**

```java
void StreamHTTPParms.setURLSuffix(String v)
```

**Parameters**

- **v** The URL suffix string.

**Remarks**

UltraLiteJ forms URLs in the following format:

```
[http|https]://host-name:port-number/url-suffix
```

By default, `url-suffix` is "Mobilink/". You can set the URL suffix to the default by setting `v` to null.

The following code illustrates how to specify a URL suffix that instructs a BlackBerry smartphone to only connect using a Wi-Fi connection:

```java
myHTTPParms.setURLSuffix(";deviceside=true;interface=wifi");
```

The following code illustrates how to specify a URL that instructs a BlackBerry smartphone to synchronize over the HTTPS protocol with a BlackBerry Enterprise Server (BES), which may be required by some BlackBerry smartphones:

```java
myHTTPParms.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.

See also
- “StreamHTTPParms.getUrlSuffix method [UltraLiteJ]” on page 224

**setZlibCompression method**

Enables or disables ZLIB compression.

**Syntax**

```java
void StreamHTTPParms.setZlibCompression(boolean enable)
```

**Parameters**

- `enable` Set to true to enable ZLIB compression, or false to disable ZLIB compression.

**Remarks**

By default, ZLIB compression is disabled.

This method corresponds to the compression=zlib protocol option.

See also
- “compression” [MobiLink - Client Administration]

**setZlibDownloadWindowSize method**

Sets the download window size for ZLIB compression.

**Syntax**

```java
void StreamHTTPParms.setZlibDownloadWindowSize(int size)
```

**Parameters**

- `size` The compression window size specification. This parameter is the base two logarithm of the window size (the size of the history buffer). For Android, specify a valid range from 9 to 15, inclusively. For BlackBerry, specify a valid range from 10 to 15, inclusively.

**Remarks**

This method corresponds to the zlib_download_window_size protocol option.

See also
- “zlib_download_window_size” [MobiLink - Client Administration]
setZlibUploadWindowSize method

Sets the upload window size for ZLIB compression.

Syntax

```java
void StreamHTTPParms.setZlibUploadWindowSize(int size)
```

Parameters

- **size**  
The compression window size specification. This parameter is the base two logarithm of the window size (the size of the history buffer). For Android, specify a valid range from 9 to 15, inclusively. For BlackBerry, specify a valid range from 8 to 15, inclusively.

Remarks

This method corresponds to the zlib_upload_window_size protocol option.

See also

- “zlib_upload_window_size” [MobiLink - Client Administration]

zlibCompressionEnabled method

Determines if ZLIB compression is enabled.

Syntax

```java
boolean StreamHTTPParms.zlibCompressionEnabled()
```

Returns

True if enabled; otherwise, returns false.

See also

- “StreamHTTPParms.setZlibCompression method [UltraLiteJ]” on page 229

E2EE_RSA variable [BlackBerry]

Specifies RSA-based end-to-end encryption when passed to the setE2eeType method.

Syntax

```java
final short StreamHTTPParms.E2EE_RSA
```

See also

- “StreamHTTPParms.setE2eeType method [BlackBerry] [UltraLiteJ]” on page 225
StreamHTTPSParms interface

Represents HTTPS stream parameters that define how to communicate with a MobiLink server using secure HTTPS connections.

Syntax

public interface StreamHTTPSParms

Base classes

- “StreamHTTPSParms interface [UltraLiteJ]” on page 218

Members

All members of the StreamHTTPSParms interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addCustomHTTPHeader method [BlackBerry]</td>
<td>Adds a message header to each HTTP request.</td>
</tr>
<tr>
<td>getCertificateCompany method</td>
<td>Returns the certificate company name for verification of secure connections.</td>
</tr>
<tr>
<td>getCertificateName method</td>
<td>Returns the certificate common name for verification of secure connections.</td>
</tr>
<tr>
<td>getCertificateUnit method</td>
<td>Returns the certificate unit name for verification of secure connections.</td>
</tr>
<tr>
<td>getCustomHTTPHeaders method [BlackBerry]</td>
<td>Returns a Hashtable object containing the HTTP headers specified using the addCustomHTTPHeader method.</td>
</tr>
<tr>
<td>getE2eePublicKey method [BlackBerry]</td>
<td>Returns the name of the file containing the end-to-end public key.</td>
</tr>
<tr>
<td>getE2eeType method [BlackBerry]</td>
<td>Returns the end-to-end encryption type that is in use.</td>
</tr>
<tr>
<td>getHost method</td>
<td>Returns the host name of the MobiLink server.</td>
</tr>
<tr>
<td>getHTTPPassword method [BlackBerry]</td>
<td>Returns the HTTP password.</td>
</tr>
<tr>
<td>getHTTPUserId method [BlackBerry]</td>
<td>Returns the HTTP user ID.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>getOutputBufferSize</code> method</td>
<td>Returns the size, in bytes, of the output buffer used to store data before it is sent to the MobiLink server.</td>
</tr>
<tr>
<td><code>getPort</code> method</td>
<td>Returns the port number used to connect to the MobiLink server.</td>
</tr>
<tr>
<td><code>getTrustedCertificates</code> method</td>
<td>Returns the name of the file containing a list of trusted root certificates used for secure synchronization.</td>
</tr>
<tr>
<td><code>getURLSuffix</code> method</td>
<td>Returns the URL suffix of the MobiLink server.</td>
</tr>
<tr>
<td><code>isRestartable</code> method</td>
<td>Determines whether restartable HTTP is used.</td>
</tr>
<tr>
<td><code>setCertificateCompany</code> method</td>
<td>Sets the certificate company name for verification of secure connections.</td>
</tr>
<tr>
<td><code>setCertificateName</code> method</td>
<td>Sets the certificate common name for verification of secure connections.</td>
</tr>
<tr>
<td><code>setCertificateUnit</code> method</td>
<td>Sets the certificate unit name for verification of secure connections.</td>
</tr>
<tr>
<td><code>setE2eePublicKey</code> method [BlackBerry]</td>
<td>Specifies the name of the file containing the end-to-end public key.</td>
</tr>
<tr>
<td><code>setE2eeType</code> method [BlackBerry]</td>
<td>Specifies the type of end-to-end encryption to use.</td>
</tr>
<tr>
<td><code>setHost</code> method</td>
<td>Sets the host name of the MobiLink server.</td>
</tr>
<tr>
<td><code>setHTTPUserIdAndPassword</code> method [BlackBerry]</td>
<td>Sets the user ID and password used for Basic HTTP authentication as described in RFC 2617.</td>
</tr>
<tr>
<td><code>setOutputBufferSize</code> method</td>
<td>Sets the size, in bytes, of the output buffer used to store data before it is sent to the MobiLink server.</td>
</tr>
<tr>
<td><code>setPort</code> method</td>
<td>Sets the port number used to connect to the MobiLink server.</td>
</tr>
<tr>
<td><code>setRestartable</code> method</td>
<td>Enables or disables restartable HTTP.</td>
</tr>
<tr>
<td><code>setTrustedCertificates</code> method</td>
<td>Sets a file containing a list of trusted root certificates used for secure synchronization.</td>
</tr>
<tr>
<td><code>setURLSuffix</code> method</td>
<td>Specifies the URL suffix to connect to the MobiLink server.</td>
</tr>
</tbody>
</table>
**Name** | **Description**
--- | ---
setZlibCompression method | Enables or disables ZLIB compression.
setZlibDownloadWindowSize method | Sets the download window size for ZLIB compression.
setZlibUploadWindowSize method | Sets the upload window size for ZLIB compression.
zlibCompressionEnabled method | Determines if ZLIB compression is enabled.
E2EE_RSA variable [BlackBerry] | Specifies RSA-based end-to-end encryption when passed to the setE2eeType method.

**Remarks**

The following example sets the stream parameters to communicate with a MobiLink server on host name "MyMLHost". The server started with the following parameters: "-x https(port=1234;certificate=RSAServer.crt;certificate_password=x)"

```java
SyncParms syncParms = myConnection.createSyncParms(
    SyncParms.HTTPS_STREAM,
    "MyUniqueMLUserID",
    "MyMLScriptVersion"
);
StreamHTTPSParms httpsParms = (StreamHTTPSParms) syncParms.getStreamParms();
httpsParms.setHost("MyMLHost");
httpsParms.setPort(1234);
```

The above example assumes that the certificate in RSAServer.crt is chained to a trusted root certificate already installed on the client host or device.

For J2SE, you can deploy the required trusted root certificate using one of the following methods:

1. Install the trusted root certificate in the lib/security/cacerts key store of the JRE.

2. Build your own key store using the Java keytool utility and setting the javax.net.ssl.trustStore Java system property to its location (set the javax.net.ssl.trustStorePassword method to an appropriate value)

3. Use the setTrustedCertificates(String) parameter to point to the deployed certificate file.

To enhance security, the setCertificateName, setCertificateCompany, and setCertificateUnit methods should be used to turn on validation of the MobiLink server certificate.

Instances implementing this interface are returned by the SyncParms.getStreamParms method when the SyncParms object is created for HTTPS synchronization.
getCertificateCompany method

Returns the certificate company name for verification of secure connections.

Syntax

String StreamHTTPSParms.getCertificateCompany()

Returns

The certificate company name.

getcertificateName method

Returns the certificate common name for verification of secure connections.

Syntax

String StreamHTTPSParms.getCertificateName()

Returns

The certificate name.

getcertificateUnit method

Returns the certificate unit name for verification of secure connections.

Syntax

String StreamHTTPSParms.getCertificateUnit()

Returns

The organization unit name.

getTrustedCertificates method

Returns the name of the file containing a list of trusted root certificates used for secure synchronization.
Syntax

    String StreamHTTPSParms.getTrustedCertificates()

Returns

    The file name of the trusted root certificates file.

See also

    ● “StreamHTTPSParms.setTrustedCertificates method [UltraLiteJ]” on page 236

**setCertificateCompany method**

    Sets the certificate company name for verification of secure connections.

Syntax

    void StreamHTTPSParms.setCertificateCompany(String val)

Parameters

    ● val  The company name.

Remarks

    The default is null, indicating that the company name does not get verified in the certificate.

**setCertificateName method**

    Sets the certificate common name for verification of secure connections.

Syntax

    void StreamHTTPSParms.setCertificateName(String val)

Parameters

    ● val  The certificate common name.

Remarks

    The default is null, indicating that the common name does not get verified in the certificate.

**setCertificateUnit method**

    Sets the certificate unit name for verification of secure connections.

Syntax

    void StreamHTTPSParms.setCertificateUnit(String val)
Parameters

- **val** The company unit name.

Remarks

The default is null, indicating that the organization unit name does not get verified in the certificate.

**setTrustedCertificates method**

Sets a file containing a list of trusted root certificates used for secure synchronization.

Syntax

```java
void StreamHTTPSParms.setTrustedCertificates(
    String filename
) throws ULjException
```

Parameters

- **filename** The file name of the trusted root certificate.

Remarks

This method supports any X.509 format that the Java Runtime Environment on the platform allows. The JKS KeyStore type for storing root certificates is used by J2SE, and the BKS KeyStore is used by Android smartphones.

This method can only be used on the J2SE platform, and Android smartphones.

Certificates are used according to the following rules of precedence:

1. If this method is called, then the certificates from the specified file are used.
2. If this method is not called and certificates were set in the database by the ulinit or ulload utilities, then those certificates are used.
3. If certificates are not specified by either this method or by the ulinit or ulload utilities, and you are on Android, then certificates are read from the operating system's trusted certificate store. This certificate store is used by web browsers when they connect to secure web servers via HTTPS.

See also

- “StreamHTTPSParms.getTrustedCertificates method [UltraLiteJ]” on page 234

**SyncObserver interface**

Receives synchronization progress information.

Syntax

```java
public interface SyncObserver
```
Members

All members of the SyncObserver interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syncProgress method</td>
<td>Informs the user of progress.</td>
</tr>
</tbody>
</table>

Remarks

Create a new class that performs synchronization, and implement it using the SyncParms.setSyncObserver method to receive synchronization progress reports.

The following example illustrates a simple SyncObserver object implementation:

```java
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.
}
```

The above class can be enabled with the following method call:

```java
SyncParms.setSyncObserver(new MyObserver());
```

See also

- “SyncParms class [UltraLiteJ]” on page 242
- “SyncParms.setSyncObserver method [UltraLiteJ]” on page 257

syncProgress method

Informs the user of progress.

Syntax

```java
boolean SyncObserver.syncProgress(int state, SyncResult data)
```

Parameters

- **state** One of the SyncObserver.States constants, representing the current state of the synchronization.
- **data** A SyncResult object containing the latest synchronization results.

Returns

return True to cancel the synchronization; otherwise, returns false to continue synchronization.
Remarks

This method is invoked during synchronization.

The various states, which are signaled as packets, are received and sent. Since multiple tables may be uploaded or downloaded in a single packet, calls to this method for any given synchronization may skip a number of states.

Note

With the exception of the SyncResult methods, no other UltraLiteJ API methods should be invoked during a syncProgress call.

See also

- “SyncObserver.States interface [UltraLiteJ]” on page 238
- “SyncParms.setSyncObserver method [UltraLiteJ]” on page 257
- “SyncResult class [UltraLiteJ]” on page 260

SyncObserver.States interface

Defines the synchronization states that can be signaled to an observer.

Syntax

public interface SyncObserver.States

Members

All members of the SyncObserver.States interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMITTING_DOWNLOAD variable</td>
<td>Denotes that the downloaded rows are being committed to the database.</td>
</tr>
<tr>
<td>CONNECTING variable</td>
<td>Denotes that a synchronization is starting.</td>
</tr>
<tr>
<td>DISCONNECTING variable</td>
<td>Denotes that the synchronization stream is disconnecting.</td>
</tr>
<tr>
<td>DONE variable</td>
<td>Denotes that synchronization is complete.</td>
</tr>
<tr>
<td>ERROR variable</td>
<td>Denotes that synchronization is complete but an error occurred.</td>
</tr>
<tr>
<td>FINISHING_UPLOAD variable</td>
<td>Denotes that the upload is finalizing.</td>
</tr>
<tr>
<td>RECEIVING_DATA variable</td>
<td>Denotes that schema information or row data is being received.</td>
</tr>
<tr>
<td>RECEIVING_TABLE variable</td>
<td>Denotes that a new table is being downloaded.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RECEIVING_UPLOAD_ACK variable</td>
<td>Denotes that an upload acknowledgement is being downloaded.</td>
</tr>
<tr>
<td>ROLLING_BACK_DOWNLOAD variable</td>
<td>Denotes that the downloaded rows are being committed to the database.</td>
</tr>
<tr>
<td>SENDING_DATA variable</td>
<td>Denotes that schema information or row data is being sent.</td>
</tr>
<tr>
<td>SENDING_DOWNLOAD_ACK variable</td>
<td>Denotes that an acknowledgement of a complete download is being sent.</td>
</tr>
<tr>
<td>SENDING_HEADER variable</td>
<td>Denotes that the synchronization stream has been opened and the header is about to be sent.</td>
</tr>
<tr>
<td>SENDING_TABLE variable</td>
<td>Denotes that a new table is being uploaded.</td>
</tr>
<tr>
<td>STARTING variable</td>
<td>Denotes that a synchronization is starting.</td>
</tr>
</tbody>
</table>

**See also**

- “SyncParms.setSyncObserver method [UltraLiteJ]” on page 257
- “SyncObserver interface [UltraLiteJ]” on page 236

**COMMITTING_DOWNLOAD variable**

Denotes that the downloaded rows are being committed to the database.

**Syntax**

```java
final int SyncObserver.States.COMMITTING_DOWNLOAD
```

**CONNECTING variable**

Denotes that a synchronization is starting.

**Syntax**

```java
final int SyncObserver.States.CONNECTING
```

**Remarks**

No actions have taken place yet.

**DISCONNECTING variable**

Denotes that the synchronization stream is disconnecting.
Syntax
    final int SyncObserver.States.DISCONNECTING

**DONE variable**

Denotes that synchronization is complete.

Syntax
    final int SyncObserver.States.DONE

Remarks
    No other states are reported.

**ERROR variable**

Denotes that synchronization is complete but an error occurred.

Syntax
    final int SyncObserver.States.ERROR

**FINISHING_UPLOAD variable**

Denotes that the upload is finalizing.

Syntax
    final int SyncObserver.States.FINISHING_UPLOAD

**RECEIVING_DATA variable**

Denotes that schema information or row data is being received.

Syntax
    final int SyncObserver.States.RECEIVING_DATA

**RECEIVING_TABLE variable**

Denotes that a new table is being downloaded.

Syntax
    final int SyncObserver.States.RECEIVING_TABLE
RECEIVING_UPLOAD_ACK variable
Denotes that an upload acknowledgement is being downloaded.

Syntax
final int SyncObserver.States.RECEIVING_UPLOAD_ACK

ROLLING_BACK_DOWNLOAD variable
Denotes that the downloaded rows are being committed to the database.

Syntax
final int SyncObserver.States.ROLLING_BACK_DOWNLOAD

Remarks
Denotes that synchronization is rolling back the download because an error was encountered during the download.

SENDING_DATA variable
Denotes that schema information or row data is being sent.

Syntax
final int SyncObserver.States.SENDING_DATA

SENDING_DOWNLOAD_ACK variable
Denotes that an acknowledgement of a complete download is being sent.

Syntax
final int SyncObserver.States.SENDINGDOWNLOAD_ACK

SENDING_HEADER variable
Denotes that the synchronization stream has been opened and the header is about to be sent.

Syntax
final int SyncObserver.States.SENDING_HEADER

Remarks
Denotes that the synchronization stream has opened, and that the header is about to be sent.
**SENDING_TABLE variable**

Denotes that a new table is being uploaded.

**Syntax**

```java
final int SyncObserver.States.SENDING_TABLE
```

**STARTING variable**

Denotes that a synchronization is starting.

**Syntax**

```java
final int SyncObserver.States.STARTING
```

**Remarks**

No actions have taken place yet.

**SyncParms class**

Maintains the parameters used during the database synchronization process.

**Syntax**

```java
public class SyncParms
```

**Members**

All members of the SyncParms class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAcknowledgeDownload method</td>
<td>Determines if the client sends download acknowledgments.</td>
</tr>
<tr>
<td>getAdditionalParms method [Android]</td>
<td>Returns the additional synchronization parameters.</td>
</tr>
<tr>
<td>getAuthenticationParms method</td>
<td>Returns parameters provided to a custom user authentication script.</td>
</tr>
<tr>
<td>getKeepPartialDownload method [Android]</td>
<td>Determines if partial downloads are enabled.</td>
</tr>
<tr>
<td>getLivenessTimeout method</td>
<td>Returns the liveness timeout length, in seconds.</td>
</tr>
<tr>
<td>getNewPassword method</td>
<td>Returns the new MobiLink password for the user specified with the setUserName method.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getPassword method</td>
<td>Returns the MobiLink password for the user specified with the setUserName method.</td>
</tr>
<tr>
<td>getPublications method</td>
<td>Returns the publications to be synchronized.</td>
</tr>
<tr>
<td>getResumePartialDownload method [Android]</td>
<td>Determines if partial downloads are to be resumed.</td>
</tr>
<tr>
<td>getStreamParms method</td>
<td>Returns the parameters used to configure the synchronization stream.</td>
</tr>
<tr>
<td>getSyncObserver method</td>
<td>Returns the currently specified SyncObserver object.</td>
</tr>
<tr>
<td>getSyncResult method</td>
<td>Returns the SyncResult object that contains the status of the synchronization.</td>
</tr>
<tr>
<td>getTableOrder method</td>
<td>Returns the order in which tables should be uploaded to the consolidated database.</td>
</tr>
<tr>
<td>getUserName method</td>
<td>Returns the MobiLink user name that uniquely identifies the client to the MobiLink server.</td>
</tr>
<tr>
<td>getVersion method</td>
<td>Returns the script version to use.</td>
</tr>
<tr>
<td>isDownloadOnly method</td>
<td>Determines if the synchronization is download-only.</td>
</tr>
<tr>
<td>isPingOnly method</td>
<td>Determines whether the client pings the MobiLink server or performs a synchronization.</td>
</tr>
<tr>
<td>isUploadOnly method</td>
<td>Determines if the synchronization is upload-only.</td>
</tr>
<tr>
<td>setAcknowledgeDownload method</td>
<td>Specifies whether the client should send download acknowledgements.</td>
</tr>
<tr>
<td>setAdditionalParms method [Android]</td>
<td>Specifies additional synchronization parameters as a semicolon-separated list of name=value pairs.</td>
</tr>
<tr>
<td>setAuthenticationParms method</td>
<td>Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).</td>
</tr>
<tr>
<td>setDownloadOnly method</td>
<td>Sets the synchronization as download-only.</td>
</tr>
<tr>
<td>setKeepPartialDownload method [Android]</td>
<td>Specifies whether partial downloads should be allowed while synchronizing.</td>
</tr>
<tr>
<td>setLivenessTimeout method</td>
<td>Sets the liveness timeout length, in seconds.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>setNewPassword method</td>
<td>Sets a new MobiLink password for the user specified with setUserName method.</td>
</tr>
<tr>
<td>setPassword method</td>
<td>Sets the MobiLink password for the user specified with the setUserName method.</td>
</tr>
<tr>
<td>setPingOnly method</td>
<td>Sets the client to ping the MobiLink server rather than perform a synchronization.</td>
</tr>
<tr>
<td>setPublications method</td>
<td>Sets the publications to be synchronized.</td>
</tr>
<tr>
<td>setResumePartialDownload method</td>
<td>Specifies whether to resume or discard a previous partial download.</td>
</tr>
<tr>
<td>[Android] setResumePartialDownload method</td>
<td></td>
</tr>
<tr>
<td>setSyncObserver method</td>
<td>Sets a SyncObserver object to monitor the progress of the synchronization.</td>
</tr>
<tr>
<td>setTableOrder method</td>
<td>Sets the order in which tables should be uploaded to the consolidated database.</td>
</tr>
<tr>
<td>setUploadOnly method</td>
<td>Sets the synchronization as upload-only.</td>
</tr>
<tr>
<td>setUserName method</td>
<td>Sets the MobiLink user name that uniquely identifies the client to the MobiLink server.</td>
</tr>
<tr>
<td>setVersion method</td>
<td>Sets the synchronization script to use.</td>
</tr>
<tr>
<td>HTTP_STREAM variable</td>
<td>Creates a SyncParms object for HTTP synchronizations.</td>
</tr>
<tr>
<td>HTTPS_STREAM variable</td>
<td>Creates a SyncParms object for secure HTTPS synchronizations.</td>
</tr>
</tbody>
</table>

**Remarks**

This interface is invoked with the Connection.createSyncParms method.

You can only set one synchronization command at a time. These commands are specified using the setDownloadOnly, setPingOnly, and setUploadOnly methods. By setting one of these methods to true, you set the other methods to false.

The UserName and Version parameters must be set. The UserName must be unique for each client database.

The communication stream is configured using the getStreamParms method based on the type of SyncParms object. For example, the following code prepares and performs an HTTP synchronization:

```java
SyncParms syncParms = myConnection.createSyncParms(
    SyncParms.HTTP_STREAM,
    "MyUniqueMLUserID",
```
"MyMLScriptVersion"
);
syncParms.setPassword("ThePWDforMyUniqueMLUserID");
syncParms.getStreamParms().setHost("MyMLHost");
myConnection.synchronize(syncParms);

Comma Separated Lists

AuthenticationParms, Publications, and TableOrder parameters are all specified using a string value that contains a comma separated list of values. Values within the list may be quoted using either single quotes or double quotes, but there are no escape characters. Leading and trailing spaces in values are ignored unless quoted. For example, the following code specifies Table A, then Table B,D, then Table C:

```
syncParms.setTableOrder( "'Table A',"Table B,D",Table C );
```

See also
- “SyncParms.getStreamParms method [UltraLiteJ]” on page 248
- “SyncParms.setUserName method [UltraLiteJ]” on page 258
- “Connection.createSyncParms method [UltraLiteJ]” on page 109
- “StreamHTTPParms interface [UltraLiteJ]” on page 218
- “StreamHTTPSParms interface [UltraLiteJ]” on page 231

getAcknowledgeDownload method

Determines if the client sends download acknowledgments.

Syntax

```
abstract boolean SyncParms.getAcknowledgeDownload()
```

Returns

True if the client sends download acknowledgments; otherwise, returns false.

See also
- “SyncParms.setAcknowledgeDownload method [UltraLiteJ]” on page 251

getAdditionalParms method [Android]

Returns the additional synchronization parameters.

Syntax

```
abstract String SyncParms.getAdditionalParms()
```

Returns

The list of additional parms or null if no parameters are specified.
See also
● “SyncParms.setAdditionalParms method [Android] [UltraLiteJ]” on page 251

getAuthenticationParms method
Returns parameters provided to a custom user authentication script.

Syntax
abstract String SyncParms.getAuthenticationParms()

Returns
The list of authentication parms or null if no parameters are specified.

See also
● “SyncParms.setAuthenticationParms method [UltraLiteJ]” on page 252

getKeepPartialDownload method [Android]
Determines if partial downloads are enabled.

Syntax
abstract boolean SyncParms.getKeepPartialDownload()

Returns
True if partial downloads are enabled; otherwise, returns false.

See also
● “SyncParms.setKeepPartialDownload method [Android] [UltraLiteJ]” on page 253

getLivenessTimeout method
Returns the liveness timeout length, in seconds.

Syntax
abstract int SyncParms.getLivenessTimeout()

Returns
The timeout.

See also
● “SyncParms.setLivenessTimeout method [UltraLiteJ]” on page 254
getNewPassword method

Returns the new MobiLink password for the user specified with the setUserName method.

Syntax

```java
abstract String SyncParms.getNewPassword()
```

Returns

The new password set after the next synchronization.

See also

- “SyncParms.setUserName method [UltraLiteJ]” on page 258
- “SyncParms.setNewPassword method [UltraLiteJ]” on page 255

getPassword method

Returns the MobiLink password for the user specified with the setUserName method.

Syntax

```java
abstract String SyncParms.getPassword()
```

Returns

The password for the MobiLink user.

See also

- “SyncParms.setPassword method [UltraLiteJ]” on page 255

getPublications method

Returns the publications to be synchronized.

Syntax

```java
abstract String SyncParms.getPublications()
```

Returns

The set of publications to synchronize.

See also

- “SyncParms.setPublications method [UltraLiteJ]” on page 256

getResumePartialDownload method [Android]

Determines if partial downloads are to be resumed.
Syntax
abstract boolean SyncParms.getResumePartialDownload()

Returns
True if partial downloads are to be resumed; otherwise, returns false.

See also
● “SyncParms.setResumePartialDownload method [Android] [UltraLiteJ]” on page 257

getStreamParms method
Returns the parameters used to configure the synchronization stream.

Syntax
abstract StreamHTTPParms SyncParms.getStreamParms()

Returns
A StreamHTTPParms or StreamHTTPSParms object specifying the parameters for HTTP or HTTPS synchronization streams. The object is returned by reference.

Remarks
The synchronization stream type is specified when the SyncParms object is created.

See also
● “Connection.createSyncParms method [UltraLiteJ]” on page 109
● “StreamHTTPParms interface [UltraLiteJ]” on page 218
● “StreamHTTPSParms interface [UltraLiteJ]” on page 231

getSyncObserver method
Returns the currently specified SyncObserver object.

Syntax
abstract SyncObserver SyncParms.getSyncObserver()

Returns
The SyncObserver object, or null if an observer was not specified.

See also
● “SyncParms.setSyncObserver method [UltraLiteJ]” on page 257
**getSyncResult method**

Returns the SyncResult object that contains the status of the synchronization.

**Syntax**

```java
abstract SyncResult SyncParms.getSyncResult()
```

**Returns**

The SyncResult object representing the result of the last call to the Connection.synchronize method.

**Remarks**

The following example illustrates how to get the result set of the last call to the Connection.synchronize method:

```java
conn.synchronize( mySyncParms );
SyncResult result = mySyncParms.getSyncResult();
display("*** Synchronized *** sent=" + result.getSentRowCount()
       + ", received=" + result.getReceivedRowCount() );
```

**Note**

This method does not return the result of the last SYNCHRONIZE SQL statement. To obtain the SyncResult object for the last SYNCHRONIZE SQL statement, use the getSyncResult method on the Connection object passed in.

**See also**

- “SyncResult class [UltraLiteJ]” on page 260
- “Connection.getSyncResult method [UltraLiteJ]” on page 114

---

**getTableOrder method**

Returns the order in which tables should be uploaded to the consolidated database.

**Syntax**

```java
abstract String SyncParms.getTableOrder()
```

**Returns**

A comma separated list of table names; otherwise, returns null if a table order was not specified. See the class description for more information about comma separated lists.

**See also**

- “SyncParms.setTableOrder method [UltraLiteJ]” on page 257
**getUserName method**

Returns the MobiLink user name that uniquely identifies the client to the MobiLink server.

**Syntax**

```
abstract String SyncParms.getUserName()
```

**Returns**

The MobiLink user name.

**See also**

- “SyncParms.setUserName method [UltraLiteJ]” on page 258

**getVersion method**

Returns the script version to use.

**Syntax**

```
abstract String SyncParms.getVersion()
```

**Returns**

The script version.

**See also**

- “SyncParms.setVersion method [UltraLiteJ]” on page 259

**isDownloadOnly method**

Determines if the synchronization is download-only.

**Syntax**

```
abstract boolean SyncParms.isDownloadOnly()
```

**Returns**

True if uploads are disabled; otherwise, returns false.

**See also**

- “SyncParms.setDownloadOnly method [UltraLiteJ]” on page 253

**isPingOnly method**

Determines whether the client pings the MobiLink server or performs a synchronization.
Syntax
    abstract boolean SyncParms.isPingOnly()

Returns
    True if the client only pings the server; otherwise, returns false.

See also
    ● “SyncParms.setPingOnly method [UltraLiteJ]” on page 256

**isUploadOnly method**
    Determines if the synchronization is upload-only.

Syntax
    abstract boolean SyncParms.isUploadOnly()

Returns
    True if downloads are disabled; otherwise, returns false.

See also
    ● “SyncParms.setUploadOnly method [UltraLiteJ]” on page 258

**setAcknowledgeDownload method**
    Specifies whether the client should send download acknowledgements.

Syntax
    abstract void SyncParms.setAcknowledgeDownload(boolean ack)

Parameters
    ● ack Set to true to have the client acknowledge a download; otherwise, set to false.

Remarks
    The default is false.

See also
    ● “SyncParms.getAcknowledgeDownload method [UltraLiteJ]” on page 245

**setAdditionalParms method [Android]**
    Specifies additional synchronization parameters as a semicolon-separated list of name=value pairs.
Syntax

```java
abstract void SyncParms.setAdditionalParms(String v) throws ULjException
```

Parameters

- **v**  A string, in the form of a semicolon-separated list of name=value pairs.

Remarks

Use this method to specify several additional synchronization parameters that cannot be specified using existing methods of the SyncParms class.

The following example illustrates how to set the AllowDownloadDupRows, CheckpointStore, and DisableConcurrency parameters on a SyncParms object:

```java
SyncParms parms;
...
parms.setAdditionalParms("AllowDownloadDupRows=1;CheckpointStore=1;DisableConcurrency=1");
```

See also

- “SyncParms.getAdditionalParms method [Android] [UltraLiteJ]” on page 245

### setAuthenticationParms method

Specifies parameters for a custom user authentication script (MobiLink authenticate_parameters connection event).

Syntax

```java
abstract void SyncParms.setAuthenticationParms(String v) throws ULjException
```

Parameters

- **v**  A comma separated list of authentication parameters, or the null reference. See the class description for more information about comma separated lists.

Remarks

Only the first 255 strings are used and each string should be no longer than the MobiLink server's limit for authentication parameters. (currently 4000 UTF8 bytes)

Strings longer than 21K characters are truncated when sent to MobiLink, and strings that exceed the server's limit for authentication parameters cause a server-side synchronization error.

See also

- “SyncParms.getAuthenticationParms method [UltraLiteJ]” on page 246
**setDownloadOnly method**

Sets the synchronization as download-only.

**Syntax**

```java
abstract void SyncParms.setDownloadOnly(boolean v)
```

**Parameters**

- `v` Set to true to disable uploads, or set false to enable uploads.

**Remarks**

The default is false. Specifying true automatically calls the setPingOnly and setUploadOnly methods, and sets them to false.

**See also**

- “SyncParms.isDownloadOnly method [UltraLiteJ]” on page 250
- “SyncParms.setPingOnly method [UltraLiteJ]” on page 256
- “SyncParms.setUploadOnly method [UltraLiteJ]” on page 258

**setKeepPartialDownload method [Android]**

Specifies whether partial downloads should be allowed while synchronizing.

**Syntax**

```java
abstract void SyncParms.setKeepPartialDownload(
  boolean c
) throws ULjException
```

**Parameters**

- `c` Set to true to enable partial downloads.

**Remarks**

The default setting is false. Set to true to enable and save partial downloads while synchronizing; otherwise, set to false to disable partial downloads and roll back downloads if any errors occur.

UltraLite has the ability to resume partial downloads that fail due to communication errors or when the user aborts through the SyncObserver object. UltraLite processes the download as it is received. If a download is interrupted, then the partial download transaction remains in the database and can be resumed during the next synchronization.

To indicate that UltraLite should save partial downloads, set to true; otherwise, the download is rolled back if an error occurs.

If a partial download was kept, then the SyncResult.getPartialDownloadRetained method returns true when the Connection.synchronize method exits.
If the `KeepPartialDownload` synchronization parameter is set to true, then you can resume a partial download. To resume a partial download, call the `Connection.synchronize` method with the `setResumePartialDownload` method set to true.

It is recommended that you leave the `KeepPartialDownload` synchronization parameter set to true in case another communications error occurs. No upload is performed if a download is skipped.

The download you receive during a resumed download is as old as when the download originally began. If you need the most up to date data, then you can do another download immediately after the resumed download completes.

When resuming a download, many of the synchronization parameters that are specified by the `SyncParms` class are not relevant. For example, the `Publications` parameter is not used. You receive requested publications during the initial download. Only the `setResumePartialDownload` and `setUserName` methods need to be used. The `setKeepPartialDownload` method can be used if desired.

If you have a partial download and it is no longer needed, then you can call the `Connection.rollbackPartialDownload` to roll back the failed download transaction. Also, if you attempt to synchronize again and do not specify the `ResumePartialDownload` parameter, then the partial download is rolled back before the next synchronization begins.

See also

- “`SyncParms.getKeepPartialDownload` method [Android] [UltraLiteJ]” on page 246
- “`SyncParms.setResumePartialDownload` method [Android] [UltraLiteJ]” on page 257
- “`SyncParms.setUserName` method [UltraLiteJ]” on page 258
- “Resumption of failed downloads” [MobiLink - Server Administration]

### setLivenessTimeout method

Sets the liveness timeout length, in seconds.

**Syntax**

```java
abstract void SyncParms.setLivenessTimeout(
    int seconds
) throws ULjException
```

**Parameters**

- **seconds** The new liveness timeout value.

**Remarks**

The liveness timeout is the length of time the server allows a remote to be idle. If the remote does not communicate with the server for 1 seconds, the server assumes that the remote has lost the connection, and terminates the sync. The remote automatically sends periodic messages to the server to keep the connection alive.

If a negative value is set, an exception is thrown. The value may be changed by the MobiLink server without notice. This change occurs if the value is set too low or too high.
The default value is 100 seconds for BlackBerry/J2SE platforms, and 240 seconds for Android platforms.

See also
  ● “SyncParms.getLivenessTimeout method [UltraLiteJ]” on page 246

**setNewPassword method**

Sets a new MobiLink password for the user specified with setUserName method.

**Syntax**

```java
abstract void SyncParms.setNewPassword(String v)
```

**Parameters**

  ● **v**  A new password for MobiLink user.

**Remarks**

The new password takes effect after the next synchronization.

The default is null, suggesting that the password does not get replaced.

See also
  ● “SyncParms.getNewPassword method [UltraLiteJ]” on page 247
  ● “SyncParms.setPassword method [UltraLiteJ]” on page 255
  ● “SyncParms.setUserName method [UltraLiteJ]” on page 258

**setPassword method**

Sets the MobiLink password for the user specified with the setUserName method.

**Syntax**

```java
abstract void SyncParms.setPassword(String v) throws ULjException
```

**Parameters**

  ● **v**  A password for the MobiLink user.

**Remarks**

This user name and password is separate from any database user ID and password. This method is used to authenticate the application against the MobiLink server.

The default is an empty string, suggesting no password.
setPingOnly method

Sets the client to ping the MobiLink server rather than perform a synchronization.

Syntax

abstract void SyncParms.setPingOnly(boolean v)

Parameters

• v Set to true to only ping the server, or set false to perform a synchronization.

Remarks

The default is false. Specifying true automatically calls the setDownloadOnly and setUploadOnly methods, and sets them to false.

See also

• “SyncParms.isPingOnly method [UltraLiteJ]” on page 250
• “SyncParms.setDownloadOnly method [UltraLiteJ]” on page 253
• “SyncParms.setUploadOnly method [UltraLiteJ]” on page 258

setPublications method

Sets the publications to be synchronized.

Syntax

abstract void SyncParms.setPublications(String pubs) throws ULjException

Parameters

• pubs A comma separated list of publication names. See the class description for more information about comma separated lists.

Remarks

The default is set to the Connection.SYNC_ALL constant, which is used to denote the synchronization of all tables in the database. To synchronize all publications, set this method to the Connection.SYNC_ALL_PUBS constant.

See also

• “SyncParms.getPublications method [UltraLiteJ]” on page 247
• “Connection.SYNC_ALL variable [UltraLiteJ]” on page 128
• “Connection.SYNC_ALL_PUBS variable [UltraLiteJ]” on page 128
setResumePartialDownload method [Android]

Specifies whether to resume or discard a previous partial download.

Syntax

```java
abstract void SyncParms.setResumePartialDownload(
    boolean c
) throws ULjException
```

Parameters

- **c** Set to true to resume a previous partial download.

Exceptions

- **ULjException class** SQLE_SYNC_INFO_INVALID is thrown by the Connection.synchronize method when more than one of the following synchronization parameters (DownloadOnly, PingOnly, ResumePartialDownload, or UploadOnly) is set to true.

Remarks

Set to true to resume a previous partial download, or false to discard a previous partial download. The default setting is false.

See also

- “SyncParms.getResumePartialDownload method [Android] [UltraLiteJ]” on page 247

setSyncObserver method

Sets a SyncObserver object to monitor the progress of the synchronization.

Syntax

```java
abstract void SyncParms.setSyncObserver(SyncObserver so)
```

Parameters

- **so** A SyncObserver object.

Remarks

The default is null, suggesting no observer.

See also

- “SyncObserver interface [UltraLiteJ]” on page 236

setTableOrder method

Sets the order in which tables should be uploaded to the consolidated database.
Syntax

abstract void SyncParms.setTableOrder(String v) throws ULjException

Parameters

- v A comma separated list of table names in the order they should be synchronized, or null, indicating no table order. See the class description for more information about comma separated lists.

Remarks

The primary table should be listed first, along with all tables containing foreign key relationships in the consolidated database.

All tables selected for synchronization by the Publications parameter are synchronized whether they are specified in the TableOrder parameter or not. Unspecified tables are synchronized by order of the foreign key relations in the client database. They are synchronized after the specified tables.

The default is a null reference, which does not override the default ordering of tables.

See also

- “SyncParms.getTableOrder method [UltraLiteJ]” on page 249
- “SyncParms.setPublications method [UltraLiteJ]” on page 256

setUploadOnly method

Sets the synchronization as upload-only.

Syntax

abstract void SyncParms.setUploadOnly(boolean v)

Parameters

- v Set to true to disable downloads, or set or false to enable downloads.

Remarks

The default is false. Specifying true automatically calls the setDownloadOnly and setPingOnly methods, and sets them to false.

See also

- “SyncParms.isUploadOnly method [UltraLiteJ]” on page 251
- “SyncParms.setDownloadOnly method [UltraLiteJ]” on page 253
- “SyncParms.setPingOnly method [UltraLiteJ]” on page 256

setUserName method

Sets the MobiLink user name that uniquely identifies the client to the MobiLink server.
Syntax

abstract void SyncParms.setUserName(String v) throws ULjException

Parameters

v  The MobiLink user name.

Remarks

This value is used to determine the following:

- Download content
- Whether to record the synchronization state
- Whether to recover from interruptions during synchronization.

This user name and password is separate from any database user ID and password. This method is used to authenticate the application against the MobiLink server.

This parameter is initialized when the SyncParms object is created.

See also

- “SyncParms.getUserName method [UltraLiteJ]” on page 250
- “SyncParms.setPassword method [UltraLiteJ]” on page 255
- “SyncParms.setNewPassword method [UltraLiteJ]” on page 255
- “Connection.createSyncParms method [UltraLiteJ]” on page 109

setVersion method

Sets the synchronization script to use.

Syntax

abstract void SyncParms.setVersion(String v) throws ULjException

Parameters

v  The script version.

Remarks

Each synchronization script in the consolidated database is marked with a version string. For example, there can be two different download_cursor scripts, and each one is identified by different version strings. The version string allows an application to choose from a set of synchronization scripts.

This parameter is initialized when the SyncParms object is created.

See also

- “SyncParms.getVersion method [UltraLiteJ]” on page 250
- “Connection.createSyncParms method [UltraLiteJ]” on page 109
HTTP_STREAM variable

Creates a SyncParms object for HTTP synchronizations.

Syntax

```java
final int SyncParms.HTTP_STREAM
```

See also

- “Connection.createSyncParms method [UltraLiteJ]” on page 109

HTTPS_STREAM variable

Creates a SyncParms object for secure HTTPS synchronizations.

Syntax

```java
final int SyncParms.HTTPS_STREAM
```

See also

- “Connection.createSyncParms method [UltraLiteJ]” on page 109

SyncResult class

Reports status-related information about a specified database synchronization.

Syntax

```java
public class SyncResult
```

Members

All members of the SyncResult class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAuthMessage method</td>
<td>Returns the authorization message of the last synchronization attempt as specified by custom user authentication synchronization scripts.</td>
</tr>
<tr>
<td>getAuthStatus method</td>
<td>Returns the authorization status code of the last synchronization attempt.</td>
</tr>
<tr>
<td>getAuthValue method</td>
<td>Returns the value specified in custom user authentication synchronization scripts.</td>
</tr>
<tr>
<td>getCurrentTableName method</td>
<td>Returns the name of the table currently being synchronized.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>getIgnoredRows</code> method</td>
<td>Determines if any uploaded rows were ignored during the last synchronization.</td>
</tr>
<tr>
<td><code>getPartialDownloadRetained method</code> [Android]</td>
<td>Checks whether a partial download was retained during the last synchronization.</td>
</tr>
<tr>
<td><code>getReceivedByteCount</code> method</td>
<td>Returns the number of bytes received during data synchronization.</td>
</tr>
<tr>
<td><code>getReceivedDeletes</code> method</td>
<td>Returns the number of received rows that have been deleted.</td>
</tr>
<tr>
<td><code>getReceivedIgnoredDeletes method</code></td>
<td>Returns the number of received delete rows that have been ignored.</td>
</tr>
<tr>
<td><code>getReceivedIgnoredUpdates method</code></td>
<td>Returns the number of received update rows that have been ignored.</td>
</tr>
<tr>
<td><code>getReceivedInserts</code> method</td>
<td>Returns the number of received rows that have been inserted.</td>
</tr>
<tr>
<td><code>getReceivedRowCount</code> method</td>
<td>Returns the number of rows received.</td>
</tr>
<tr>
<td><code>getReceivedTruncateDeletes method</code></td>
<td>Returns the number of rows that have been deleted by a downloaded truncate operation.</td>
</tr>
<tr>
<td><code>getReceivedUpdates</code> method</td>
<td>Returns the number of received rows that have been applied as updates.</td>
</tr>
<tr>
<td><code>getSentByteCount</code> method</td>
<td>Returns the number of bytes sent during data synchronization.</td>
</tr>
<tr>
<td><code>getSentDeletes</code> method</td>
<td>Returns the number of deleted rows sent.</td>
</tr>
<tr>
<td><code>getSentInserts</code> method</td>
<td>Returns the number of inserted rows sent.</td>
</tr>
<tr>
<td><code>getSentUpdates</code> method</td>
<td>Returns the number of updated rows sent.</td>
</tr>
<tr>
<td><code>getStreamErrorCode</code> method</td>
<td>Returns the error code reported by the stream itself.</td>
</tr>
<tr>
<td><code>getStreamErrorMessage</code> method</td>
<td>Returns the error message reported by the stream itself.</td>
</tr>
<tr>
<td><code>getSyncedTableCount</code> method</td>
<td>Returns the number of synchronized tables so far.</td>
</tr>
<tr>
<td><code>getTotalDownloadRowCount</code> method</td>
<td>Returns the total number of rows to be received in the download.</td>
</tr>
<tr>
<td><code>getTotalTableCount</code> method</td>
<td>Returns the number of tables to be synchronized.</td>
</tr>
<tr>
<td><code>isUploadOK</code> method</td>
<td>Determines if the last upload synchronization was successful.</td>
</tr>
</tbody>
</table>
See also

- “SyncParms.getSyncResult method [UltraLiteJ]” on page 249

**getAuthMessage method**

Returns the authorization message of the last synchronization attempt as specified by custom user authentication synchronization scripts.

**Syntax**

```java
abstract String SyncResult.getAuthMessage()
```

**Returns**

A string containing information regarding the authentication of the last synchronization.

**Remarks**

Blank or empty messages are returned as null.

An authentication message may be returned for any authorization status code. See the authentication_message MobiLink named system parameter for more detail.

**getAuthStatus method**

Returns the authorization status code of the last synchronization attempt.

**Syntax**

```java
abstract int SyncResult.getAuthStatus()
```

**Returns**

An AuthStatusCode value.

**getAuthValue method**

Returns the value specified in custom user authentication synchronization scripts.

**Syntax**

```java
abstract int SyncResult.getAuthValue()
```

**Returns**

An integer returned from custom user authentication synchronization scripts.

**getCurrentTableName method**

Returns the name of the table currently being synchronized.
SyncResult class

Syntax
abstract String SyncResult GetCurrentTableName()

Returns
The table name.

**getIgnoredRows method**
Determines if any uploaded rows were ignored during the last synchronization.

Syntax
abstract boolean SyncResult getIgnoredRows()

Returns
True if any uploaded rows were ignored during the last synchronization; otherwise, returns false if no rows were ignored.

**getPartialDownloadRetained method [Android]**
Checks whether a partial download was retained during the last synchronization.

Syntax
abstract boolean SyncResult getPartialDownloadRetained()

Returns
True if a download was interrupted and the partial download was retained, or false if the download was not interrupted or if the partial download was rolled back.

**getReceivedByteCount method**
Returns the number of bytes received during data synchronization.

Syntax
abstract long SyncResult getReceivedByteCount()

Returns
The number of bytes.

**getReceivedDeletes method**
Returns the number of received rows that have been deleted.
Syntax
abstract long SyncResult.getReceivedDeletes()

Returns
The number of downloaded rows applied as deletes.

See also
- “SyncResult.getReceivedIgnoredDeletes method [UltraLiteJ]” on page 264
- “SyncResult.getReceivedIgnoredUpdates method [UltraLiteJ]” on page 264
- “SyncResult.getReceivedInserts method [UltraLiteJ]” on page 265
- “SyncResult.getReceivedTruncateDeletes method [UltraLiteJ]” on page 265
- “SyncResult.getReceivedUpdates method [UltraLiteJ]” on page 266

getReceivedIgnoredDeletes method
Returns the number of received delete rows that have been ignored.

Syntax
abstract long SyncResult.getReceivedIgnoredDeletes()

Returns
The number of downloaded delete rows that were ignored.

See also
- “SyncResult.getReceivedIgnoredUpdates method [UltraLiteJ]” on page 264
- “SyncResult.getReceivedDeletes method [UltraLiteJ]” on page 263
- “SyncResult.getReceivedInserts method [UltraLiteJ]” on page 265
- “SyncResult.getReceivedTruncateDeletes method [UltraLiteJ]” on page 265
- “SyncResult.getReceivedUpdates method [UltraLiteJ]” on page 266

getReceivedIgnoredUpdates method
Returns the number of received update rows that have been ignored.

Syntax
abstract long SyncResult.getReceivedIgnoredUpdates()

Returns
The number of downloaded update rows that were ignored. For UltraLite Java edition databases, this value is always 0.

Remarks
Received update rows are ignored only when duplicate primary keys are allowed in the download. (which is possible on Android only) Duplicate downloaded updates otherwise result in a failed synchronization.
getReceivedInserts method

Returns the number of received rows that have been inserted.

Syntax

```java
abstract long SyncResult.getReceivedInserts()
```

Returns

The number of rows applied as inserts.

See also

- “SyncResult.getReceivedDeletes method [UltraLiteJ]” on page 263
- “SyncResult.getReceivedIgnoredDeletes method [UltraLiteJ]” on page 264
- “SyncResult.getReceivedInserts method [UltraLiteJ]” on page 265
- “SyncResult.getReceivedTruncateDeletes method [UltraLiteJ]” on page 265
- “SyncResult.getReceivedUpdates method [UltraLiteJ]” on page 266

getReceivedRowCount method

Returns the number of rows received.

Syntax

```java
abstract long SyncResult.getReceivedRowCount()
```

Returns

The number of rows received.

Remarks

This count includes rows that may be ignored when the download is applied.

See also

- “SyncResult.getTotalDownloadRowCount method [UltraLiteJ]” on page 269

getReceivedTruncateDeletes method

Returns the number of rows that have been deleted by a downloaded truncate operation.
Syntax

abstract long SyncResult.getReceivedTruncateDeletes()

Returns

The number of rows truncated.

Remarks

Each downloaded truncate operation appears as a single row in the row download as counted by the getReceivedRowCount method but may result in zero or many rows being truncated, as counted by this method.

See also

- “SyncResult.getReceivedDeletes method [UltraLiteJ]” on page 263
- “SyncResult.getReceivedIgnoredDeletes method [UltraLiteJ]” on page 264
- “SyncResult.getReceivedIgnoredUpdates method [UltraLiteJ]” on page 264
- “SyncResult.getReceivedInserts method [UltraLiteJ]” on page 265
- “SyncResult.getReceivedUpdates method [UltraLiteJ]” on page 266

getReceivedUpdates method

Returns the number of received rows that have been applied as updates.

Syntax

abstract long SyncResult.getReceivedUpdates()

Returns

The number of rows applied as updates.

See also

- “SyncResult.getReceivedDeletes method [UltraLiteJ]” on page 263
- “SyncResult.getReceivedIgnoredDeletes method [UltraLiteJ]” on page 264
- “SyncResult.getReceivedIgnoredUpdates method [UltraLiteJ]” on page 264
- “SyncResult.getReceivedInserts method [UltraLiteJ]” on page 265
- “SyncResult.getReceivedTruncate Deletes method [UltraLiteJ]” on page 265

getSentByteCount method

Returns the number of bytes sent during data synchronization.

Syntax

abstract long SyncResult.getSentByteCount()

Returns

The number of bytes sent.
**getSentDeletes method**

Returns the number of deleted rows sent.

**Syntax**

```java
abstract long SyncResult.getSentDeletes()
```

**Returns**

The number of deleted rows sent.

**Remarks**

The number of inserts, updates, and deletes may differ than the number of operations performed on the tables being synchronized because all operations on a given row are coalesced into one.

**See also**

- “SyncResult.getSentInserts method [UltraLiteJ]” on page 267
- “SyncResult.getSentUpdates method [UltraLiteJ]” on page 267

**getSentInserts method**

Returns the number of inserted rows sent.

**Syntax**

```java
abstract long SyncResult.getSentInserts()
```

**Returns**

The number of inserted rows sent.

**Remarks**

The number of inserts, updates, and deletes may differ than the number of operations performed on the tables being synchronized because all operations on a given row are coalesced into one.

**See also**

- “SyncResult.getSentDeletes method [UltraLiteJ]” on page 267
- “SyncResult.getSentUpdates method [UltraLiteJ]” on page 267

**getSentUpdates method**

Returns the number of updated rows sent.

**Syntax**

```java
abstract long SyncResult.getSentUpdates()
```
Returns

The number of updated rows sent.

Remarks

The number of inserts, updates, and deletes may differ than the number of operations performed on the tables being synchronized because all operations on a given row are coalesced into one.

See also

● “SyncResult.getSentDeletes method [UltraLiteJ]” on page 267
● “SyncResult.getSentInserts method [UltraLiteJ]” on page 267

getStreamErrorCode method

Returns the error code reported by the stream itself.

Syntax

abstract int SyncResult.getStreamErrorCode()

Returns

0 if there was no communication stream error; otherwise, returns the response code from the server.

Remarks

This method returns the HTTP response code.

getStreamErrorMessage method

Returns the error message reported by the stream itself.

Syntax

abstract String SyncResult.getStreamErrorMessage()

Returns

Null if no message is available; otherwise, returns the response message.

Remarks

This method returns the HTTP response message.

getSyncedTableCount method

Returns the number of synchronized tables so far.

Syntax

abstract int SyncResult.getSyncedTableCount()
Returns
The number of tables synchronized.

**getTotalDownloadRowCount method**
Returns the total number of rows to be received in the download.

Syntax
```java
abstract long SyncResult.getTotalDownloadRowCount()
```

Returns
The number of rows to be received in the download. This number includes any rows that do not apply, such as deletes for rows that are not on the client.

Remarks
This number includes duplicate rows that are ignored. This value is not set until the synchronization enters the SyncObserver.State.RECEIVING_TABLE state for the first table.

**getTotalTableCount method**
Returns the number of tables to be synchronized.

Syntax
```java
abstract int SyncResult.getTotalTableCount()
```

Returns
The number of tables to synchronize.

**isUploadOK method**
Determines if the last upload synchronization was successful.

Syntax
```java
abstract boolean SyncResult.isUploadOK()
```

Returns
True if the last upload synchronization was successful; otherwise, returns false.

**SyncResult.AuthStatusCode interface**
Enumerates the authorization codes returned by the MobiLink server.
Syntax

```java
public interface SyncResult.AuthStatusCode
```

Members

All members of the SyncResult.AuthStatusCode interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPIRED variable</td>
<td>User ID or password has expired.</td>
</tr>
<tr>
<td>IN_USE variable</td>
<td>User ID is already in use.</td>
</tr>
<tr>
<td>INVALID variable</td>
<td>Bad user ID or password.</td>
</tr>
<tr>
<td>UNKNOWN variable</td>
<td>Authorization status is unknown.</td>
</tr>
<tr>
<td>VALID variable</td>
<td>User ID and password were valid at time of synchronization.</td>
</tr>
<tr>
<td>VALID_BUT_EXPIRES_SOON variable</td>
<td>User ID and password were valid at time of synchronization but expire soon.</td>
</tr>
</tbody>
</table>

See also

- “SyncResult.getAuthStatus method [UltraLiteJ]” on page 262

**EXPIRED variable**

User ID or password has expired.

Syntax

```java
final int SyncResult.AuthStatusCode.EXPIRED
```

Remarks

Authorization fails.

**IN_USE variable**

User ID is already in use.

Syntax

```java
final int SyncResult.AuthStatusCode.IN_USE
```

Remarks

Authorization fails.
**INVALID variable**

Bad user ID or password.

**Syntax**

```java
final int SyncResult.AuthStatusCode.INVALID
```

**Remarks**

Authorization fails.

**UNKNOWN variable**

Authorization status is unknown.

**Syntax**

```java
final int SyncResult.AuthStatusCode.UNKNOWN
```

**Remarks**

This code suggests that a synchronization has not been performed.

**VALID variable**

User ID and password were valid at time of synchronization.

**Syntax**

```java
final int SyncResult.AuthStatusCode.VALID
```

**VALID_BUT_EXPIRES_SOON variable**

User ID and password were valid at time of synchronization but expire soon.

**Syntax**

```java
final int SyncResult.AuthStatusCode.VALID_BUT_EXPIRES_SOON
```

### TableSchema interface

Specifies the schema of a table and provides constants defining the names of system tables.

**Syntax**

```java
public interface TableSchema
```

**Members**

All members of the TableSchema interface, including all inherited members.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_ARTICLES variable</td>
<td>Contains the name of the system table containing information about publication articles.</td>
</tr>
<tr>
<td>SYS_COLUMNS variable</td>
<td>Contains the name of the system table containing information about the table columns in the database.</td>
</tr>
<tr>
<td>SYS_FKEY_COLUMNS variable</td>
<td>Contains the name of the system table containing information about foreign key columns.</td>
</tr>
<tr>
<td>SYS_FOREIGN_KEYS variable</td>
<td>Contains the name of the system table containing information about foreign keys in the database.</td>
</tr>
<tr>
<td>SYS_INDEX_COLUMNS variable</td>
<td>Contains the name of the system table containing information about the index columns in the database.</td>
</tr>
<tr>
<td>SYS_INDEXES variable</td>
<td>Contains the name of the system table containing information about the table indexes in the database.</td>
</tr>
<tr>
<td>SYS_INTERNAL variable</td>
<td>Contains the name of the system table containing internal information.</td>
</tr>
<tr>
<td>SYS_PRIMARY_INDEX variable</td>
<td>Contains the name of the primary key index of system tables.</td>
</tr>
<tr>
<td>SYS_PUBLICATIONS variable</td>
<td>Contains the name of the system table containing information about database publications.</td>
</tr>
<tr>
<td>SYS_TABLES variable</td>
<td>Contains the name of the system table containing information about the tables in the database.</td>
</tr>
<tr>
<td>SYS_ULDATA variable</td>
<td>Contains the name of the system table containing information about system values.</td>
</tr>
<tr>
<td>SYS_ULDATA_INTERNAL variable</td>
<td>Contains the type for internal system data.</td>
</tr>
<tr>
<td>SYS_ULDATA_OPTION variable</td>
<td>Contains the type for option system data.</td>
</tr>
<tr>
<td>SYS_ULDATA_PROPERTY variable</td>
<td>Contains the type for property system data.</td>
</tr>
<tr>
<td>TABLE_IS_DOWNLOAD_ONLY variable</td>
<td>Denotes that a table is a download-only table (a table that is uploaded when synchronized).</td>
</tr>
<tr>
<td>TABLE_IS_NOSYNC variable</td>
<td>Denotes that a table is a non-synchronizing table.</td>
</tr>
</tbody>
</table>
### TABLE_IS_SYSTEM variable
Denotes that a table is a system table.

### Remarks
This interface only contains table-related constants. They include system table names, table flags, and types of data in the `sysuldata` system table.

### SYS_ARTICLES variable
Contains the name of the system table containing information about publication articles.

**Syntax**
```java
final String TableSchema.SYS_ARTICLES
```

### SYS_COLUMNS variable
Contains the name of the system table containing information about the table columns in the database.

**Syntax**
```java
final String TableSchema.SYS_COLUMNS
```

### SYS_FKEY_COLUMNS variable
Contains the name of the system table containing information about foreign key columns.

**Syntax**
```java
final String TableSchema.SYS_FKEY_COLUMNS
```

### SYS_FOREIGN_KEYS variable
Contains the name of the system table containing information about foreign keys in the database.

**Syntax**
```java
final String TableSchema.SYS_FOREIGN_KEYS
```

### SYS_INDEX_COLUMNS variable
Contains the name of the system table containing information about the index columns in the database.

**Syntax**
```java
final String TableSchema.SYS_INDEX_COLUMNS
```
SYS_INDEXES variable
Contains the name of the system table containing information about the table indexes in the database.

Syntax
final String TableSchema.SYS_INDEXES

SYS INTERNAL variable
Contains the name of the system table containing internal information.

Syntax
final String TableSchema.SYS_INTERNAL

SYS_PRIMARY_INDEX variable
Contains the name of the primary key index of system tables.

Syntax
final String TableSchema.SYS_PRIMARY_INDEX

SYS_PUBLICATIONS variable
Contains the name of the system table containing information about database publications.

Syntax
final String TableSchema.SYS_PUBLICATIONS

SYS_TABLES variable
Contains the name of the system table containing information about the tables in the database.

Syntax
final String TableSchema.SYS_TABLES

SYS_ULDATA variable
Contains the name of the system table containing information about system values.

Syntax
final String TableSchema.SYS_ULDATA
SYS_ULDATA_INTERNAL variable
Contains the type for internal system data.

Syntax
final String TableSchema.SYS_ULDATA_INTERNAL

SYS_ULDATA_OPTION variable
Contains the type for option system data.

Syntax
final String TableSchema.SYS_ULDATA_OPTION

SYS_ULDATA_PROPERTY variable
Contains the type for property system data.

Syntax
final String TableSchema.SYS_ULDATA_PROPERTY

TABLE_IS_DOWNLOAD_ONLY variable
Denotes that a table is a download-only table (a table that is uploaded when synchronized).

Syntax
final short TableSchema.TABLE_IS_DOWNLOAD_ONLY

Remarks
This value can be logically combined with other flags in the table_flags column of the SYS_TABLES table except the TABLE_IS_NOSYNC flag.

TABLE_IS_NOSYNC variable
Denotes that a table is a non-synchronizing table.

Syntax
final short TableSchema.TABLE_IS_NOSYNC

Remarks
This value can be logically combined with other flags in the table_flags column of the SYS_TABLES table except the TABLE_IS_DOWNLOAD_ONLY flag.
TABLE_IS_SYSTEM variable

Denotes that a table is a system table.

Syntax

```java
final short TableSchema.TABLE_IS_SYSTEM
```

Remarks

This value can be logically combined with other flags in the table_flags column of the SYS_TABLES table.

ULjEvent interface [Android]

Represents an UltraLiteJ API system event.

Syntax

```java
public interface ULjEvent
```

Members

All members of the ULjEvent interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getParameter method</td>
<td>Returns a named parameter for the event.</td>
</tr>
<tr>
<td>getType method</td>
<td>Returns the event type.</td>
</tr>
<tr>
<td>COMMIT_EVENT variable</td>
<td>Indicates the &quot;Commit&quot; event type.</td>
</tr>
<tr>
<td>SYNC_COMPLETE_EVENT variable</td>
<td>Indicates the &quot;Synchronization complete&quot; event type.</td>
</tr>
<tr>
<td>TABLE_MODIFIED_EVENT variable</td>
<td>Indicates the &quot;Table modified&quot; event type.</td>
</tr>
</tbody>
</table>

getParameter method

Returns a named parameter for the event.

Syntax

```java
String ULjEvent.getParameter(String name) throws ULjException
```

Parameters

- **name** The event name to get the value for.
**getType method**

Returns the event type.

**Syntax**

```
short ULjEvent.getType()
```

**COMMIT_EVENT variable**

Indicates the "Commit" event type.

**Syntax**

```
final short ULjEvent.COMMIT_EVENT
```

**SYNC_COMPLETE_EVENT variable**

Indicates the "Synchronization complete" event type.

**Syntax**

```
final short ULjEvent.SYNC_COMPLETE_EVENT
```

**TABLE_MODIFIED_EVENT variable**

Indicates the "Table modified" event type.

**Syntax**

```
final short ULjEvent.TABLE_MODIFIED_EVENT
```

**ULjException class**

Supersedes the exceptions thrown by the database.

**Syntax**

```
public class ULjException
```

**Members**

All members of the ULjException class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCausingException method</td>
<td>Returns the ULjException object when an exception is caused.</td>
</tr>
<tr>
<td>getErrorCode method</td>
<td>Returns the error code associated with the exception.</td>
</tr>
</tbody>
</table>
### getCausingException method

Returns the ULjException object when an exception is caused.

**Syntax**

```java
abstract ULjException ULjException.getCausingException()
```

**Returns**

Null if no causing exceptions exist; otherwise, returns the ULjException object.

### getErrorCode method

Returns the error code associated with the exception.

**Syntax**

```java
abstract int ULjException.getErrorCode()
```

**Returns**

The error code.

### getParameter method [Android]

Returns the specified error parameter.

**Syntax**

```java
abstract String ULjException.getParameter(short param_no)
```

**Parameters**

- `param_no`  A one-based parameter number.

**Returns**

The error parameter.
**getParameterCount method [Android]**

Returns the number of error parameters.

**Syntax**

```
abstract short ULjException.getParameterCount()
```

**Returns**

The number of error parameters.

**getSqlOffset method**

Returns the error offset within the SQL string.

**Syntax**

```
abstract int ULjException.getSqlOffset()
```

**Returns**

-1 when there is no SQL string associated with the error message; otherwise, returns the zero-base offset within that string where the error occurred.

---

**Unsigned64 class**

Implements unsigned 64-bit binary values.

**Syntax**

```
public class Unsigned64
```

**Members**

All members of the Unsigned64 class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add method</td>
<td>Adds two values together and places the result in self.</td>
</tr>
<tr>
<td>compare method</td>
<td>Compares two long values.</td>
</tr>
<tr>
<td>divide method</td>
<td>Divides two values and places the result in self.</td>
</tr>
<tr>
<td>multiply method</td>
<td>Multiplies two values together and places the result in self.</td>
</tr>
<tr>
<td>remainder method</td>
<td>Returns the remainder when one value is divided by another.</td>
</tr>
<tr>
<td>subtract method</td>
<td>Subtracts two values and places the result in self.</td>
</tr>
</tbody>
</table>
Remarks

The intent of this class is to keep values as long integers and interpret them using the static methods in this class.

This class cannot be instantiated.

**add method**

Adds two values together and places the result in self.

**Syntax**

```
final long Unsigned64.add(long v1, long v2)
```

**Parameters**

- **v1** The first operand
- **v2** The second operand

**Returns**

The sum of the operands.

**compare method**

Compares two long values.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compare(int, int) method</td>
<td>Compares two integer values.</td>
</tr>
<tr>
<td>compare(long, long) method</td>
<td>Compares two long values.</td>
</tr>
</tbody>
</table>

**compare(int, int) method**

Compares two integer values.

**Syntax**

```
final byte Unsigned64.compare(int v1, int v2)
```

**Parameters**

- **v1** The first value to be compared.
- **v2** The second value to be compared.
Returns
-1 when \( v_2 \) is greater than \( v_1 \), 0 when \( v_1 \) equals \( v_2 \), or 1 when \( v_2 \) is less than \( v_1 \).

**compare(long, long) method**
Compares two long values.

**Syntax**
```
final byte Unsigned64.compare(long v1, long v2)
```

**Parameters**
- \( \textit{v1} \) The first value to be compared.
- \( \textit{v2} \) The second value to be compared.

**Returns**
-1 when \( v_2 \) is greater than \( v_1 \), 0 when \( v_1 \) equals \( v_2 \), or 1 when \( v_2 \) is less than \( v_1 \).

**divide method**
Divides two values and places the result in self.

**Syntax**
```
final long Unsigned64.divide(long v1, long v2)
```

**Parameters**
- \( \textit{v1} \) The first operand
- \( \textit{v2} \) The second operand

**Returns**
The first operand divided by the second operand.

**multiply method**
Multiplies two values together and places the result in self.

**Syntax**
```
final long Unsigned64.multiply(long v1, long v2)
```

**Parameters**
- \( \textit{v1} \) The first operand.
- \( \textit{v2} \) The second operand.
Returns
The product of v1 and v2.

**remainder method**
Returns the remainder when one value is divided by another.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remainder(long, long) method</td>
<td>Returns the remainder when one value is divided by another.</td>
</tr>
<tr>
<td>remainder(long, long, long) method</td>
<td>Returns the remainder when one value multiplied by a given quotient is subtracted from another value (v1 - quot * v2).</td>
</tr>
</tbody>
</table>

**remainder(long, long) method**
Returns the remainder when one value is divided by another.

**Syntax**
```
final long Unsigned64.remainder(long v1, long v2)
```

**Parameters**
- **v1** The value to be divided.
- **v2** The value to divide by.

**Returns**
The remainder, represented as a long integer.

**remainder(long, long, long) method**
Returns the remainder when one value multiplied by a given quotient is subtracted from another value (v1 - quot * v2).

**Syntax**
```
final long Unsigned64.remainder(long v1, long v2, long quot)
```

**Parameters**
- **v1** The value to be divided.
- **v2** The value to divide by.
The value of the quotient.

**Returns**

The remainder, represented as a long integer.

---

**subtract method**

Subtracts two values and places the result in self.

**Syntax**

```java
final long Unsigned64.subtract(long v1, long v2)
```

**Parameters**

- **v1** The first operand.
- **v2** The second operand.

**Returns**

The result of v2 being subtracted from v1.

---

**UUIDValue interface**

Describes a unique identifier (UUID or Universally Unique IDentifier) object.

**Syntax**

```java
public interface UUIDValue
```

**Members**

All members of the UUIDValue interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getString method</td>
<td>Returns the String representation of the UUIDValue object.</td>
</tr>
<tr>
<td>isNull method</td>
<td>Determines if the UUIDValue object is null.</td>
</tr>
<tr>
<td>set method</td>
<td>Sets the UUIDValue object with a String value.</td>
</tr>
<tr>
<td>setNull method</td>
<td>Sets the UUIDValue object to null.</td>
</tr>
</tbody>
</table>

**Remarks**

Such entities are useful when a unique identifier is required and where the value can be arbitrary.

A UUIDValue can also be created by the SQL INSERT statement when no value is supplied for a column in a table where the column was created with the DEFAULT NEWID() clause.
A Connection object can be used to create a UUIDValue using the createUUIDValue method.

See also
- “Connection.createUUIDValue method [UltraLiteJ]” on page 110

**getString method**

Returns the String representation of the UUIDValue object.

**Syntax**

```java
String UUIDValue.getString() throws ULjException
```

**Returns**

The String value.

**isNull method**

Determines if the UUIDValue object is null.

**Syntax**

```java
boolean UUIDValue.isNull()
```

**Returns**

True if the object is null; otherwise, returns false.

**set method**

Sets the UUIDValue object with a String value.

**Syntax**

```java
void UUIDValue.set(String value) throws ULjException
```

**Parameters**

- **value**  A numerical value represented as a String.

**setNull method**

Sets the UUIDValue object to null.

**Syntax**

```java
void UUIDValue.setNull() throws ULjException
```
ValidateDatabaseProgressData interface [Android]

Reports ValidateDatabase progress data.

Syntax

```java
public interface ValidateDatabaseProgressData
```

Members

All members of the ValidateDatabaseProgressData interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getParms method</td>
<td>Returns a parameter array associated with the status ID.</td>
</tr>
<tr>
<td>getStatusId method</td>
<td>Returns the status ID of the validation operation.</td>
</tr>
</tbody>
</table>

See also

- “UltraLite Validate Database utility (ulvalid)” [UltraLite - Database Management and Reference]

getParms method

Returns a parameter array associated with the status ID.

Syntax

```java
abstract String[] ValidateDatabaseProgressData.getParms()
```

Returns

The array of parameters as a fixed size; unused parameters are null.

See also

- “ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ]” on page 286

getStatusId method

Returns the status ID of the validation operation.

Syntax

```java
abstract short ValidateDatabaseProgressData.getStatusId()
```

Returns

A status ID constant.
ValidateDatabaseProgressData.StatusId interface [Android]

Specifies possible status IDs for the UltraLite Validate Database utility.

Syntax

```java
public interface ValidateDatabaseProgressData.StatusId
```

Members

All members of the ValidateDatabaseProgressData.StatusId interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UL_VALID_BAD_ROWID variable</td>
<td>There is an invalid row identifier in the index.</td>
</tr>
<tr>
<td>UL_VALID_CHECKING_INDEX variable</td>
<td>Checking an index.</td>
</tr>
<tr>
<td>UL_VALID_CHECKING_PAGE variable</td>
<td>Send a periodic status message while checking database pages.</td>
</tr>
<tr>
<td>UL_VALID_CHECKING_TABLE variable</td>
<td>Checking a table.</td>
</tr>
<tr>
<td>UL_VALID_CONNECT_ERROR variable</td>
<td>Error connecting to the database.</td>
</tr>
<tr>
<td>UL_VALID_CORRUPT_PAGE variable</td>
<td>A page is corrupt.</td>
</tr>
<tr>
<td>UL_VALID_CORRUPT_PAGE_TABLE variable</td>
<td>Page table is corrupt.</td>
</tr>
<tr>
<td>UL_VALID_DATABASE_ERROR variable</td>
<td>An error occurred accessing the database.</td>
</tr>
<tr>
<td>UL_VALID_END variable</td>
<td>End validation.</td>
</tr>
<tr>
<td>UL_VALID_FAILED_CHECKSUM variable</td>
<td>Page checksum failed.</td>
</tr>
<tr>
<td>UL_VALID_INTERRUPTED variable</td>
<td>Validation process interrupted.</td>
</tr>
<tr>
<td>UL_VALID_NO_ERROR variable</td>
<td>No error occurred.</td>
</tr>
<tr>
<td>UL_VALID_ROWCOUNT_MISMATCH variable</td>
<td>The number of rows in the index is different from the table row count.</td>
</tr>
<tr>
<td>UL_VALID_START variable</td>
<td>Start validation.</td>
</tr>
<tr>
<td>UL_VALID_STARTUP_ERROR variable</td>
<td>Error starting the database for low-level access.</td>
</tr>
</tbody>
</table>
UL_VALID_BAD_ROWID variable
There is an invalid row identifier in the index.

Syntax
final short ValidateDatabaseProgressData.StatusId.UL_VALID_BAD_ROWID

Remarks
The first parameter returned by the ValidateDatabaseProgressData.getParms method tracks the table name. The second parameter tracks the index name.

See also
● “ValidateDatabaseProgressData.getParms method [UltraLiteJ]” on page 285

UL_VALID_CHECKING_INDEX variable
Checking an index.

Syntax
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CHECKING_INDEX

Remarks
The first parameter returned by the ValidateDatabaseProgressData.getParms method stores the table name. The second parameter stores the index name.

See also
● “ValidateDatabaseProgressData.getParms method [UltraLiteJ]” on page 285

UL_VALID_CHECKING_PAGE variable
Send a periodic status message while checking database pages.

Syntax
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CHECKING_PAGE

Remarks
The first parameter returned by the ValidateDatabaseProgressData.getParms method tracks a number associated with the page. The order is not defined.

See also
● “ValidateDatabaseProgressData.getParms method [UltraLiteJ]” on page 285
**UL_VALID_CHECKING_TABLE variable**

Checking a table.

**Syntax**

```java
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CHECKING_TABLE
```

**Remarks**

The first parameter returned by the ValidateDatabaseProgressData.getParms method tracks the table name.

**See also**

- “ValidateDatabaseProgressData.getParms method [UltraLiteJ]” on page 285

**UL_VALID_CONNECT_ERROR variable**

Error connecting to the database.

**Syntax**

```java
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CONNECT_ERROR
```

**UL_VALID_CORRUPT_PAGE variable**

A page is corrupt.

**Syntax**

```java
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CORRUPT_PAGE
```

**Remarks**

The first parameter returned by the ValidateDatabaseProgressData.getParms method tracks a number associated with the page.

**See also**

- “ValidateDatabaseProgressData.getParms method [UltraLiteJ]” on page 285

**UL_VALID_CORRUPT_PAGE_TABLE variable**

Page table is corrupt.

**Syntax**

```java
final short ValidateDatabaseProgressData.StatusId.UL_VALID_CORRUPT_PAGE_TABLE
```
**UL_VALID_DATABASE_ERROR variable**

An error occurred accessing the database.

Syntax

```java
final short ValidateDatabaseProgressData.StatusId.UL_VALID_DATABASE_ERROR
```

Remarks

Check the SQLCODE for more information.

See also

- “SQL Anywhere error messages sorted by SQLCODE” [Error Messages]

**UL_VALID_END variable**

End validation.

Syntax

```java
final short ValidateDatabaseProgressData.StatusId.UL_VALID_END
```

Remarks

The first parameter returned by the ValidateDatabaseProgressData.getParms method tracks the resulting SQLCODE, which indicates success or failure.

See also

- “ValidateDatabaseProgressData.getParms method [UltraLiteJ]” on page 285
- “SQL Anywhere error messages sorted by SQLCODE” [Error Messages]

**UL_VALID FAILED_CHECKSUM variable**

Page checksum failed.

Syntax

```java
final short ValidateDatabaseProgressData.StatusId.UL_VALID_FAILED_CHECKSUM
```

Remarks

The first parameter returned by the ValidateDatabaseProgressData.getParms method tracks a number associated with the page.

See also

- “ValidateDatabaseProgressData.getParms method [UltraLiteJ]” on page 285
ULVALID_INTERRUPTED variable
Validation process interrupted.

Syntax
final short ValidateDatabaseProgressData.StatusId.ULVALID_INTERRUPTED

ULVALID_NO_ERROR variable
No error occurred.

Syntax
final short ValidateDatabaseProgressData.StatusId.ULVALID_NO_ERROR

ULVALID_ROWCOUNT_MISMATCH variable
The number of rows in the index is different from the table row count.

Syntax
final short ValidateDatabaseProgressData.StatusId.ULVALID_ROWCOUNT_MISMATCH

Remarks
The first parameter returned by the ValidateDatabaseProgressData.getParms method tracks the table
name. The second parameter tracks index name.

See also
● “ValidateDatabaseProgressData.getParms method [UltraLiteJ]” on page 285

ULVALID_START variable
Start validation.

Syntax
final short ValidateDatabaseProgressData.StatusId.ULVALID_START

ULVALID_STARTUP_ERROR variable
Error starting the database for low-level access.

Syntax
final short ValidateDatabaseProgressData.StatusId.ULVALID_STARTUP_ERROR
ValidateDatabaseProgressListener interface [Android]

Receives ValidateDatabase progress events.

Syntax

public interface ValidateDatabaseProgressListener

Members

All members of the ValidateDatabaseProgressListener interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>validateProgressed method</td>
<td>Gets invoked during a ValidateDatabase operation to inform the user of the validation progress.</td>
</tr>
</tbody>
</table>

validateProgressed method

Gets invoked during a ValidateDatabase operation to inform the user of the validation progress.

Syntax

boolean ValidateDatabaseProgressListener.validateProgressed(
    ValidateDatabaseProgressData data
)

Parameters

- **data**  A ValidateDatabaseProgressData object containing the latest validate progress data.

Returns

True to cancel the validation process; otherwise, returns false.
Index

A
accessing schema information
  UltraLiteJ about, 19
add method
  DecimalNumber interface [UltraLiteJ API], 143
  Unsigned64 class [UltraLiteJ API], 280
addCustomHTTPHeader method [BlackBerry]
  StreamHTTPParms interface [UltraLiteJ API], 220
AfterLast method
  UltraLiteJ example, 17
afterLast method [Android]
  ResultSet interface [UltraLiteJ API], 194
Android
  about UltraLiteJ applications, 3
  coding examples, 30
  connecting to a database, 5
  creating a database, 5
  CustDB synchronization, 30
  database schemas, 19
  database stores, 5
  deploying applications, 27
  disconnecting from an UltraLite database, 26
  handling errors, 19
  Network protocol options, 23
  setup considerations, 4
  synchronizing with MobiLink, 21
  tutorial, 41
applications
  deploying BlackBerry, 49
  developing for Android and BlackBerry, 3, 27, 28
architectures
  UltraLiteJ, 79
ASCENDING variable
  IndexSchema interface [UltraLiteJ API], 174
AutoCommit mode
  UltraLiteJ development, 14

B
BeforeFirst method
  UltraLiteJ example, 17
beforeFirst method [Android]
  ResultSet interface [UltraLiteJ API], 194
BIG variable
  Domain interface [UltraLiteJ API], 149
BINARY variable
  Domain interface [UltraLiteJ API], 149
BIT variable
  Domain interface [UltraLiteJ API], 150
BlackBerry
  about UltraLiteJ applications, 3
  adding synchronization function, 64
  coding examples, 30
  concurrent synchronization, 22
  connecting to a database, 5
  creating a database, 5
  creating an Eclipse project, 49
  creating an UltraLiteJ application, 49
  CustDB synchronization, 23
  data synchronization, 22
  database schemas, 19
  database stores, 5
  deploying applications, 27, 28
  disconnecting from an UltraLite Java edition database, 26
  handling errors, 19
  JDE Component Package, 49
  Network protocol options, 23
  setup considerations, 4
  Signature Tool, 49
  synchronizing with MobiLink, 21
  tutorial, 49

cancelWaitForEvent method [Android]
  Connection interface [UltraLiteJ API], 106
changeEncryptionKey method
  Connection interface [UltraLiteJ API], 107
close method
  PreparedStatement interface [UltraLiteJ API], 177
  ResultSet interface [UltraLiteJ API], 194
code listing
  BlackBerry application tutorial, 72
COLUMN_DEFAULT_AUTOFILENAME variable
  ColumnSchema interface [UltraLiteJ API], 80
COLUMN_DEFAULT_AUTOCURRENT variable
  ColumnSchema interface [UltraLiteJ API], 81
COLUMN_DEFAULT_CONSTANT variable
  ColumnSchema interface [UltraLiteJ API], 81
COLUMN_DEFAULT_CURRENT_DATE variable
  ColumnSchema interface [UltraLiteJ API], 82
COLUMN_DEFAULT_CURRENT_TIME variable
  ColumnSchema interface [UltraLiteJ API], 82
DatabaseManager class [UltraLiteJ API], 136
CONNECTED variable
Connection interface [UltraLiteJ API], 121
connecting
UltraLite and UltraLite Java edition databases, 5
CONNECTING variable
SyncObserver.States interface [UltraLiteJ API], 239
Connection interface [UltraLiteJ API]
cancelWaitForEvent method [Android], 106
changeEncryptionKey method, 107
commit method, 107
CONNECTED variable, 121
createDecimalNumber method, 107
createSyncParms method, 109
createUUIDValue method, 110
description, 102
dropDatabase method, 110
dropDatabase method [BlackBerry], 111
getDatabaseId method [BlackBerry], 111
getDatabaseInfo method, 111
getDatabaseProperty method, 111
getLastDownloadTime method, 112
getLastIdentity method, 112
getLastWarning method, 113
getOption method [BlackBerry], 113
getState method, 114
getSyncObserver method, 114
getSyncResult method, 114
isSynchronizationDeleteDisabled method [BlackBerry], 115
NOT_CONNECTED variable, 121
OPTION_BLOB_FILE_BASE_DIR variable [BlackBerry], 121
OPTION_DATABASE_ID variable [BlackBerry], 122
OPTION_DATE_FORMAT variable, 122
OPTION_DATE_ORDER variable, 122
OPTION_MAX_HASH_SIZE variable, 123
OPTION_MLREMOTE_ID variable [BlackBerry], 123
OPTION_MLSERVER_VERSION variable [BlackBerry], 124
OPTION_NEAREST_CENTURY variable, 124
OPTION_PRECISION variable, 124
OPTION_SCALE variable, 125
OPTION_TIME_FORMAT variable, 125
OPTION_TIMESTAMP_FORMAT variable, 126
OPTION_TIMESTAMP_INCREMENT variable, 126
OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT variable, 127
prepareStatement method, 115
PROPERTY_DATABASE_NAME variable, 127
PROPERTY_PAGE_SIZE variable, 128
registerForEvent method [Android], 116
release method, 116
resetLastDownloadTime method, 116
rollback method, 117
rollbackPartialDownload method [Android], 117
setDatabaseId method, 117
setOption method, 118
setSyncObserver method, 119
SYNC_ALL variable, 128
SYNC_ALL_DB_PUB_NAME variable, 128
SYNC_ALL_PUBS variable, 128
synchronize method, 119
ULVF_DATABASE variable [Android], 129
ULVF_EXPRESS variable [Android], 129
ULVF_FULL_VALIDATE variable [Android], 129
ULVF_INDEX variable [Android], 129
ULVF_TABLE variable [Android], 130
unregisterForEvent method [Android], 119
validateDatabase method [Android], 120
waitForEvent method [Android], 121
Connection objects
UltraLiteJ, 5
createConfigurationFile method
DatabaseManager class [UltraLiteJ API], 136
createConfigurationFileAndroid method
DatabaseManager class [UltraLiteJ API], 137
createConfigurationNonPersistent method [BlackBerry]
DatabaseManager class [UltraLiteJ API], 137
createDatabase method
DatabaseManager class [UltraLiteJ API], 138
createDecimalNumber method
Connection interface [UltraLiteJ API], 107
createFileTransfer method
DatabaseManager class [UltraLiteJ API], 139
createFileTransferAndroid method [Android]
DatabaseManager class [UltraLiteJ API], 139
createObjectStoreTransfer method [BlackBerry]
DatabaseManager class [UltraLiteJ API], 140
createSISHTTPListener method [BlackBerry]
  DatabaseManager class [UltraLiteJ API], 141
createSyncParms method
  Connection interface [UltraLiteJ API], 109
createUUIDValue method
  Connection interface [UltraLiteJ API], 110
CustDB
  Android sample, 30
  BlackBerry sample, 23

D

data modification
  UltraLiteJ with SQL, 8
data synchronization
  UltraLite Java edition databases, 22
database schemas
  UltraLiteJ API access, 19
database stores
  Android and BlackBerry stores, 5
DatabaseInfo interface [UltraLiteJ API]
  description, 130
  getCommitCount method [BlackBerry], 131
  getDbFormat method [BlackBerry], 131
  getDbSize method [BlackBerry], 131
  getLogSize method [BlackBerry], 132
  getNumberRowsToUpload method, 132
  getPageReads method, 133
  getPageSize method, 133
  getPageWrites method, 133
  getRelease method, 134
DatabaseManager class [UltraLiteJ API]
  connect method, 136
  createConfigurationFile method, 136
  createConfigurationFileAndroid method, 137
  createConfigurationNonPersistent method [BlackBerry], 137
  createConfigurationObjectStore method [BlackBerry], 138
  createDatabase method, 138
  createFileTransfer method, 139
  createFileTransferAndroid method [Android], 139
  createObjectStoreTransfer method [BlackBerry], 140
  createSISHTTPListener method [BlackBerry], 141
  description, 134
  release method, 141
setErrorLanguage method, 142
DatabaseManager objects
  UltraLiteJ, 5
DATE variable
  Domain interface [UltraLiteJ API], 150
DecimalNumber interface [UltraLiteJ API]
  add method, 143
  description, 142
  divide method, 143
  getString method, 144
  isNull method, 144
  multiply method, 144
  set method, 144
  setNull method, 145
  subtract method, 145
deploying
  UltraLiteJ applications, 27, 28
DESCENDING variable
  SyncObserver.States interface [UltraLiteJ API], 239
divide method
  DecimalNumber interface [UltraLiteJ API], 143
Unsigned64 class [UltraLiteJ API], 281
DML
  UltraLiteJ, 14
Domain interface [UltraLiteJ API]
  BIG variable, 149
  BINARY variable, 149
  BIT variable, 150
  DATE variable, 150
  description, 145
  DOMAIN_MAX variable, 150
  DOUBLE variable, 150
  INTEGER variable, 150
  LONGBINARY variable, 151
  LONGBINARYFILE variable, 151
  LONGVARCHAR variable, 151
  NUMERIC variable, 151
  REAL variable, 152
  SHORT variable, 152
  ST_GEOMETRY variable, 152
  TIME variable, 152
TIMESTAMP variable, 153
TIMESTAMP_ZONE variable, 153
TINY variable, 153
UNSIGNED_BIG variable, 153
UNSIGNED_INTEGER variable, 154
UNSIGNED_SHORT variable, 154
UUID variable, 154
VARCHAR variable, 154
DOMAIN_MAX variable
  Domain interface [UltraLiteJ API], 150
DONE variable
  SyncObserver.States interface [UltraLiteJ API], 240
DOUBLE variable
  Domain interface [UltraLiteJ API], 150
downloadFile method
  FileTransfer interface [UltraLiteJ API], 157
dropDatabase method
  Connection interface [UltraLiteJ API], 110

E
E2EE_RSA variable [BlackBerry]
  StreamHTTPParms interface [UltraLiteJ API], 230
extension/Shutdown method [BlackBerry]
  Connection interface [UltraLiteJ API], 111
enableAesDBEncryption method
  ConfigPersistent interface [UltraLiteJ API], 93
enableObfuscation method
  ConfigPersistent interface [UltraLiteJ API], 93
error handling
  UltraLiteJ, 19
ERROR variable
  SyncObserver.States interface [UltraLiteJ API], 240
errors
  UltraLiteJ API handling , 19
example code
  CreateDb, 31
  CreateSales, 34
  DumpSchema, 36
  encrypted, 36
  LoadDb, 31
  ReadInnerJoin, 33
  ReadSeq, 32
  Reorg, 35
  SalesReport, 34
  SortTransactions, 35
  Sync, 21, 39
  UltraLiteJ, 30
execute method
  PreparedStatement interface [UltraLiteJ API], 177
executeQuery method
  PreparedStatement interface [UltraLiteJ API], 177
EXPIRED variable
  SyncResult.AuthStatusCode interface [UltraLiteJ API], 270
F
FileTransfer interface [UltraLiteJ API]
  description, 155
downloadFile method, 157
getAuthenticationParms method, 158
getAuthStatus method, 158
getAuthValue method, 159
getAuthCode method, 159
getLivenessTimeout method, 159
getLocalFileName method, 159
getLocalPath method, 160
getPassword method, 160
getRemoteKey method, 160
getServerFileName method, 161
getStreamErrorCode method, 161
getStreamErrorMessage method, 161
getStreamParms method, 162
getUserName method, 162
.getVersion method, 162
isResumePartialTransfer method, 163
isTransferredFile method, 163
setAuthenticationParms method, 163
setLivenessTimeout method, 164
setLocalFileName method, 164
setLocalPath method, 165
setPassword method, 166
setRemoteKey method, 166
setResumePartialTransfer method, 167
setServerFileName method, 167
setUserName method, 168
setVersion method, 168
uploadFile method, 169
FileTransferProgressData interface [UltraLiteJ API]
  description, 170
getBytesTransferred method, 170
getSizeTransferred method, 171
getResumedAtSize method, 171
fileTransferProgressed method
  FileTransferProgressListener interface [UltraLiteJ API], 172
FileTransferProgressListener interface [UltraLiteJ API]
  description, 172
  fileTransferProgressed method, 172
FINISHING_UPLOAD variable
  SyncObserver.States interface [UltraLiteJ API], 240
First method
  UltraLiteJ example, 17
  first method [Android]
    ResultSet interface [UltraLiteJ API], 194
G
getAcknowledgeDownload method
  SyncParms class [UltraLiteJ API], 245
getAdditionalParms method [Android]
  SyncParms class [UltraLiteJ API], 245
getAddressName method
  ResultSetMetadata interface [UltraLiteJ API], 210
getAuthenticationParms method
  FileTransfer interface [UltraLiteJ API], 158
  SyncParms class [UltraLiteJ API], 246
getAuthMessage method
  SyncResult class [UltraLiteJ API], 262
getAuthStatus method
  FileTransfer interface [UltraLiteJ API], 158
  SyncResult class [UltraLiteJ API], 262
getAuthValue method
  FileTransfer interface [UltraLiteJ API], 159
  SyncResult class [UltraLiteJ API], 262
getBlobInputStream method
  ResultSet interface [UltraLiteJ API], 194
getBlobOutputStream method
  PreparedStatement interface [UltraLiteJ API], 179
cGetBlobReader method
  ResultSet interface [UltraLiteJ API], 197
cGetBlobWriter method
  PreparedStatement interface [UltraLiteJ API], 179
cGetColumnCount method
  ResultSetMetadata interface [UltraLiteJ API], 211
cGetCommitCount method [BlackBerry]
  DatabaseInfo interface [UltraLiteJ API], 131
cGetConnectionString method [Android]
  ConfigPersistent interface [UltraLiteJ API], 94
cGetCorrelationName method
  ResultSetMetadata interface [UltraLiteJ API], 211
cGetCurrentTableName method
  SyncResult class [UltraLiteJ API], 262
cGetCustomHTTPHeaders method [BlackBerry]
  StreamHTTPParms interface [UltraLiteJ API], 221
cGetDatabaseId method [BlackBerry]
  Connection interface [UltraLiteJ API], 111
cGetDatabaseInfo method
  Connection interface [UltraLiteJ API], 111
cGetDatabaseName method
  Configuration interface [UltraLiteJ API], 100
cGetDatabaseProperty method
  Connection interface [UltraLiteJ API], 111
cGetDate method
  ResultSet interface [UltraLiteJ API], 198
cGetDbFormat method [BlackBerry]
  DatabaseInfo interface [UltraLiteJ API], 131
cGetDbSize method [BlackBerry]
  DatabaseInfo interface [UltraLiteJ API], 131
cGetDecimalNumber method
  ResultSet interface [UltraLiteJ API], 199
cGetDomainName method
  ResultSetMetadata interface [UltraLiteJ API], 211
cGetDomainPrecision method
  ResultSetMetadata interface [UltraLiteJ API], 212
getDomainScale method
   ResultSetMetadata interface [UltraLiteJ API], 212
getDomainSize method
   ResultSetMetadata interface [UltraLiteJ API], 212
getDomainType method
   ResultSetMetadata interface [UltraLiteJ API], 213
getDouble method
   ResultSet interface [UltraLiteJ API], 200
getE2eePublicKey method [BlackBerry]
   StreamHTTPParms interface [UltraLiteJ API], 221
getE2eeType method [BlackBerry]
   StreamHTTPParms interface [UltraLiteJ API], 222
getEncryptionKey method
   ConfigPersistent interface [UltraLiteJ API], 94
getErrorCode method
   ULjException class [UltraLiteJ API], 278
getExtraParameters method [Android]
   StreamHTTPParms interface [UltraLiteJ API], 222
getFileAuthCode method
   FileTransfer interface [UltraLiteJ API], 159
getFileName method
   FileTransfer interface [UltraLiteJ API], 159
getFileSize method
   FileTransferProgressData interface [UltraLiteJ API], 171
getFloat method
   ResultSet interface [UltraLiteJ API], 201
getHost method
   StreamHTTPParms interface [UltraLiteJ API], 222
getHTTPPassword method [BlackBerry]
   StreamHTTPParms interface [UltraLiteJ API], 223
getHTTPUserld method [BlackBerry]
   StreamHTTPParms interface [UltraLiteJ API], 223
getIgnoredRows method
   SyncResult class [UltraLiteJ API], 263
getInt method
   ResultSet interface [UltraLiteJ API], 202
getKeepPartialDownload method [Android]
   SyncParms class [UltraLiteJ API], 246
getLastDownloadTime method
   Connection interface [UltraLiteJ API], 112
getLastIdentity method
   Connection interface [UltraLiteJ API], 112
getLastWarning method
   Connection interface [UltraLiteJ API], 113
getLazyLoadIndexes method [BlackBerry]
   ConfigPersistent interface [UltraLiteJ API], 94
getLivenessTimeout method
   FileTransfer interface [UltraLiteJ API], 159
getLocalFileName method
   FileTransfer interface [UltraLiteJ API], 159
getLocalPath method
   FileTransfer interface [UltraLiteJ API], 160
getLogSize method [BlackBerry]
   DatabaseInfo interface [UltraLiteJ API], 132
getLong method
   ResultSet interface [UltraLiteJ API], 203
getMessage method
   SQLInfo interface [Android] [UltraLiteJ API], 217
getNewPassword method
   SyncParms class [UltraLiteJ API], 247
getNumberOfRowsToUpload method
   DatabaseInfo interface [UltraLiteJ API], 132
getOption method [BlackBerry]
   Connection interface [UltraLiteJ API], 113
getOrdinal method
   PreparedStatement interface [UltraLiteJ API], 180
getResult method
   ResultSet interface [UltraLiteJ API], 203
getOutputBufferSize method
   StreamHTTPParms interface [UltraLiteJ API], 223
getPageReads method
   DatabaseInfo interface [UltraLiteJ API], 133
getPageSize method
   DatabaseInfo interface [UltraLiteJ API], 133
getPageWrited method
   DatabaseInfo interface [UltraLiteJ API], 133
getParameter method
   SQLInfo interface [Android] [UltraLiteJ API], 217
ULjEvent interface [Android] [UltraLiteJ API], 276
getParameter method [Android]
   ULjException class [UltraLiteJ API], 278
getParameterCount method
   SQLInfo interface [Android] [UltraLiteJ API], 218
getParameterCount method [Android]
   PreparedStatement interface [UltraLiteJ API], 180
ULjException class [UltraLiteJ API], 279
getParameterType method [Android]
   PreparedStatement interface [UltraLiteJ API], 180
getParms method
   ValidateDatabaseProgressData interface [Android]
   SyncResult class [UltraLiteJ API], 263
getPassword method
   FileTransfer interface [UltraLiteJ API], 160
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Class/Interface</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getPlan method</td>
<td>PreparedStatement interface [UltraLiteJ API]</td>
<td>181</td>
</tr>
<tr>
<td>getPlanTree method</td>
<td>PreparedStatement interface [UltraLiteJ API]</td>
<td>181</td>
</tr>
<tr>
<td>getPort method</td>
<td>StreamHTTPParms interface [UltraLiteJ API]</td>
<td>224</td>
</tr>
<tr>
<td>getPublications method</td>
<td>SyncParms class [UltraLiteJ API]</td>
<td>247</td>
</tr>
<tr>
<td>getQualifiedName method</td>
<td>ResultSetMetadata interface [UltraLiteJ API]</td>
<td>213</td>
</tr>
<tr>
<td>getReceivedByteCount method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>263</td>
</tr>
<tr>
<td>getReceivedDeletes method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>263</td>
</tr>
<tr>
<td>getReceivedIgnoredDeletes method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>264</td>
</tr>
<tr>
<td>getReceivedIgnoredUpdates method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>264</td>
</tr>
<tr>
<td>getReceivedInserts method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>265</td>
</tr>
<tr>
<td>getReceivedRowCount method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>265</td>
</tr>
<tr>
<td>getReceivedTruncateDeletes method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>265</td>
</tr>
<tr>
<td>getReceivedUpdates method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>266</td>
</tr>
<tr>
<td>getRelease method</td>
<td>DatabaseInfo interface [UltraLiteJ API]</td>
<td>134</td>
</tr>
<tr>
<td>getRemoteKey method</td>
<td>FileTransfer interface [UltraLiteJ API]</td>
<td>160</td>
</tr>
<tr>
<td>getResultSet method</td>
<td>PreparedStatement interface [UltraLiteJ API]</td>
<td>182</td>
</tr>
<tr>
<td>getResultSetMetadata method</td>
<td>ResultSet interface [UltraLiteJ API]</td>
<td>204</td>
</tr>
<tr>
<td>getResumedAtSize method</td>
<td>FileTransferProgressData interface [UltraLiteJ API]</td>
<td>171</td>
</tr>
<tr>
<td>getResumePartialDownload method</td>
<td>SyncParms class [UltraLiteJ API]</td>
<td>247</td>
</tr>
<tr>
<td>getRowCount method</td>
<td>ResultSet interface [UltraLiteJ API]</td>
<td>204</td>
</tr>
<tr>
<td>getRowScoreFlushSize method</td>
<td>ConfigPersistent interface [UltraLiteJ API]</td>
<td>95</td>
</tr>
<tr>
<td>getRowScoreMaximum method</td>
<td>ConfigPersistent interface [UltraLiteJ API]</td>
<td>95</td>
</tr>
<tr>
<td>getSentByteCount method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>266</td>
</tr>
<tr>
<td>getSentDeletes method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>267</td>
</tr>
<tr>
<td>getSentInserts method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>267</td>
</tr>
<tr>
<td>getSentUpdates method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>267</td>
</tr>
<tr>
<td>getServerFileName method</td>
<td>FileTransfer interface [UltraLiteJ API]</td>
<td>161</td>
</tr>
<tr>
<td>getSize method</td>
<td>ResultSet interface [UltraLiteJ API]</td>
<td>204</td>
</tr>
<tr>
<td>getSQLCode method</td>
<td>SQLInfo interface [Android] [UltraLiteJ API]</td>
<td>218</td>
</tr>
<tr>
<td>getSQLCount method</td>
<td>SQLInfo interface [Android] [UltraLiteJ API]</td>
<td>218</td>
</tr>
<tr>
<td>getSqlOffset method</td>
<td>ULjException class [UltraLiteJ API]</td>
<td>279</td>
</tr>
<tr>
<td>getState method</td>
<td>Connection interface [UltraLiteJ API]</td>
<td>114</td>
</tr>
<tr>
<td>getStatusId method</td>
<td>ValidateDatabaseProgressData interface [Android]</td>
<td>285</td>
</tr>
<tr>
<td>getStreamErrorCode method</td>
<td>FileTransfer interface [UltraLiteJ API]</td>
<td>161</td>
</tr>
<tr>
<td>getStreamError_message method</td>
<td>FileTransfer interface [UltraLiteJ API]</td>
<td>161</td>
</tr>
<tr>
<td>getStreamParms method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>268</td>
</tr>
<tr>
<td>getStreamParms method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>268</td>
</tr>
<tr>
<td>getString method</td>
<td>DatabaseInfo interface [UltraLiteJ API]</td>
<td>134</td>
</tr>
<tr>
<td>getSyncedTableCount method</td>
<td>SyncResult class [UltraLiteJ API]</td>
<td>268</td>
</tr>
<tr>
<td>getSyncObserver method</td>
<td>Connection interface [UltraLiteJ API]</td>
<td>114</td>
</tr>
<tr>
<td>getSyncResult method</td>
<td>ConfigPersistent interface [UltraLiteJ API]</td>
<td>248</td>
</tr>
<tr>
<td>getSyncResult method</td>
<td>Connection interface [UltraLiteJ API]</td>
<td>114</td>
</tr>
<tr>
<td>getTableName method</td>
<td>ResultSetMetadata interface [UltraLiteJ API]</td>
<td>214</td>
</tr>
<tr>
<td>getTableColumnName method</td>
<td>ResultSetMetadata interface [UltraLiteJ API]</td>
<td>214</td>
</tr>
<tr>
<td>getTableOrder method</td>
<td>SyncParms class [UltraLiteJ API]</td>
<td>249</td>
</tr>
</tbody>
</table>
getTotalDownloadRowCount method
SyncResult class [UltraLiteJ API], 269
getTotalTableCount method
SyncResult class [UltraLiteJ API], 269
getTrustedCertificates method
StreamHTTPParms interface [UltraLiteJ API], 234
getType method
ULjEvent interface [Android] [UltraLiteJ API], 277
getUpdateCount method
PreparedStatement interface [UltraLiteJ API], 183
getURLSuffix method
StreamHTTPParms interface [UltraLiteJ API], 224
getUserName method
FileTransfer interface [UltraLiteJ API], 162
SyncParms class [UltraLiteJ API], 250
getUserName method [Android]
ConfigPersistent interface [UltraLiteJ API], 95
getUUIDValue method
ResultSet interface [UltraLiteJ API], 206
getVersion method
FileTransfer interface [UltraLiteJ API], 162
SyncParms class [UltraLiteJ API], 250
getWrittenName method
ResultSetMetadata interface [UltraLiteJ API], 214

H
hasResultSet method
PreparedStatement interface [UltraLiteJ API], 183
hasShadowPaging method [BlackBerry]
ConfigPersistent interface [UltraLiteJ API], 96
HTTP_STREAM variable
SyncParms class [UltraLiteJ API], 260
HTTPS_STREAM variable
SyncParms class [UltraLiteJ API], 260

I
IN_USE variable
SyncResult.AuthStatusCode interface [UltraLiteJ API], 270
indexes
UltraLiteJ API schema information in, 19
IndexSchema interface [UltraLiteJ API]
ASCENDING variable, 174
DESCENDING variable, 174
description, 173
PERSISTENT variable, 174
PRIMARY_INDEX variable, 174
UNIQUE_INDEX variable, 175
UNIQUE_KEY variable, 175
IndexSchema object
UltraLiteJ development, 19
inner joins
element code, 33
INTEGER variable
Domain interface [UltraLiteJ API], 150
INVALID variable
SyncResult.AuthStatusCode interface [UltraLiteJ API], 271
isDownloadOnly method
SyncParms class [UltraLiteJ API], 250
isNull method
DecimalNumber interface [UltraLiteJ API], 144
ResultSet interface [UltraLiteJ API], 207
UUIDValue interface [UltraLiteJ API], 284
isPingOnly method
SyncParms class [UltraLiteJ API], 250
isRestartable method
StreamHTTPParms interface [UltraLiteJ API], 224
isResumePartialTransfer method
FileTransfer interface [UltraLiteJ API], 163
isSynchronizationDeleteDisabled method
[BlackBerry]
Connection interface [UltraLiteJ API], 115
isTransferredFile method
FileTransfer interface [UltraLiteJ API], 163
isUploadOK method
SyncResult class [UltraLiteJ API], 269
isUploadOnly method
SyncParms class [UltraLiteJ API], 251

L
Last method
UltraLiteJ example, 17
last method [Android]
ResultSet interface [UltraLiteJ API], 208
LONGBINARY variable
Domain interface [UltraLiteJ API], 151
LONGBINARYFILE variable
Domain interface [UltraLiteJ API], 151
LONGVARCHAR variable
Domain interface [UltraLiteJ API], 151

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 301
M
managing
UltraLiteJ transactions, 14
multiply method
DecimalNumber interface [UltraLiteJ API], 144
Unsigned64 class [UltraLiteJ API], 281

N
navigating SQL result sets
UltraLiteJ, 17
next method
ResultSet interface [UltraLiteJ API], 208
UltraLiteJ example, 17
NOT_CONNECTED variable
Connection interface [UltraLiteJ API], 121
NUMERIC variable
Domain interface [UltraLiteJ API], 151

O
onError method
SISRequestHandler interface [BlackBerry]
[UltraLiteJ API], 216
onRequest method
SISRequestHandler interface [BlackBerry]
[UltraLiteJ API], 216
OPTION_BLOB_FILE_BASE_DIR variable
[BlackBerry]
Connection interface [UltraLiteJ API], 121
OPTION_DATABASE_ID variable [BlackBerry]
Connection interface [UltraLiteJ API], 122
OPTION_DATE_FORMAT variable
Connection interface [UltraLiteJ API], 122
OPTION_DATE_ORDER variable
Connection interface [UltraLiteJ API], 122
OPTION_MAX_HASH_SIZE variable
Connection interface [UltraLiteJ API], 123
OPTION_ML_REMOTE_ID variable [BlackBerry]
Connection interface [UltraLiteJ API], 123
OPTION_ML_SERVER_VERSION variable [BlackBerry]
Connection interface [UltraLiteJ API], 123
OPTION_NEAREST_CENTURY variable
Connection interface [UltraLiteJ API], 124
OPTION_TIME_FORMAT variable
Connection interface [UltraLiteJ API], 125
OPTION_TIMESTAMP_FORMAT variable
Connection interface [UltraLiteJ API], 126
OPTION_TIMESTAMP_INCREMENT variable
Connection interface [UltraLiteJ API], 126
OPTION_TIMESTAMP_WITH_TIME_ZONE_FOR
MAT variable
Connection interface [UltraLiteJ API], 127

P
PERSISTENT variable
IndexSchema interface [UltraLiteJ API], 174
platforms
supported in UltraLiteJ, 1
prepared statements
UltraLiteJ, 14
preparedStatement interface
UltraLiteJ, 14
PreparedStatement interface [UltraLiteJ API]
close method, 177
description, 175
execute method, 177
executeQuery method, 177
getBlobOutputStream method, 178
getClobWriter method, 179
getOrdinal method, 180
getParameterCount method [Android], 180
getParameterType method [Android], 180
getPlan method, 181
getPlanTree method, 181
getResultSet method, 182
hasResultSet method, 183
set method, 183
setNull method, 191
prepareStatement method
Connection interface [UltraLiteJ API], 115
previous method
ResultSet interface [UltraLiteJ API], 208
UltraLiteJ example, 17
PRIMARY_INDEX variable
IndexSchema interface [UltraLiteJ API], 174
PROPERTY_DATABASE_NAME variable
Connection interface [UltraLiteJ API], 127
PROPERTY_PAGE_SIZE variable
Connection interface [UltraLiteJ API], 128
REAL variable
  Domain interface [UltraLiteJ API], 152
RECEIVING_DATA variable
  SyncObserver.States interface [UltraLiteJ API], 240
RECEIVING_TABLE variable
  SyncObserver.States interface [UltraLiteJ API], 240
RECEIVING_UPLOAD_ACK variable
  SyncObserver.States interface [UltraLiteJ API], 241
registerForEvent method [Android]
  Connection interface [UltraLiteJ API], 116
Relative method
  UltraLiteJ example, 17
relative method [Android]
  ResultSet interface [UltraLiteJ API], 209
release method
  Connection interface [UltraLiteJ API], 116
DatabaseManager class [UltraLiteJ API], 141
remainder method
  Unsigned64 class [UltraLiteJ API], 282
resetLastDownloadTime method
  Connection interface [UltraLiteJ API], 116
result set schemas
  UltraLiteJ, 17
result sets
  UltraLiteJ navigation, 17
ResultSet interface [UltraLiteJ API]
  afterLast method [Android], 194
  beforeFirst method [Android], 194
close method, 194
description, 191
first method [Android], 194
getBlobInputStream method, 194
getBoolean method, 195
getBytes method, 196
getBlobReader method, 197
getDate method, 198
getDecimalNumber method, 199
getDouble method, 200
getFloat method, 201
getInt method, 202
getLong method, 203
getOrdinal method, 203
getResultSetMetadata method, 204
getRowCount method [Android], 204
getSize method, 204
getString method, 205
getUUIDValue method, 206
isNull method, 207
last method [Android], 208
next method, 208
previous method, 208
relative method [Android], 209
ResultSet object
  UltraLiteJ data retrieval example, 17
ResultSetMetadata interface [UltraLiteJ API]
description, 209
getAliasName method, 210
columnCount method, 211
correlationName method, 211
domainName method, 211
domainPrecision method, 212
domainScale method, 212
domainSize method, 212
domainType method, 213
qualifiedName method, 213
getColumnCount method, 211
correlationName method, 211
domainName method, 211
domainPrecision method, 212
domainScale method, 212
domainSize method, 212
domainType method, 213
getName method, 213
getWrittenName method, 214
rollback method
  Connection interface [UltraLiteJ API], 117
UltraLiteJ transactions, 14
rollbackPartialDownload method [Android]
  Connection interface [UltraLiteJ API], 117
rollbacks
  UltraLiteJ transactions, 14
ROLLING_BACK_DOWNLOAD variable
  SyncObserver.States interface [UltraLiteJ API], 241
samples
  CreateDb.java, 31
  CreateSales.java, 34
  Demo.java, 30
  DumpSchema , 36
  Encrypted.java, 36
  LoadDb.java, 31
  ReadInnerJoin.java, 33
  ReadSeq.java, 32
  Reorg.java, 35
SalesReport.java, 34
SortTransactions.java, 35
Sync, 39
synchronizing with UltraLiteJ, 23
UltraLiteJ, 30
schemas
UltraLiteJ API access, 19
SELECT statement
UltraLiteJ data retrieval example, 17
selecting data from database tables
UltraLiteJ, 17
SENDING_DATA variable
SyncObserver.States interface [UltraLiteJ API], 241
SENDING_DOWNLOAD_ACK variable
SyncObserver.States interface [UltraLiteJ API], 241
SENDING_HEADER variable
SyncObserver.States interface [UltraLiteJ API], 241
SENDING_TABLE variable
SyncObserver.States interface [UltraLiteJ API], 242
set method
DecimalNumber interface [UltraLiteJ API], 144
PreparedStatement interface [UltraLiteJ API], 183
UUIDValue interface [UltraLiteJ API], 284
setAcknowledgeDownload method
SyncParms class [UltraLiteJ API], 251
setAdditionalParms method [Android]
SyncParms class [UltraLiteJ API], 251
setAuthenticationParms method
FileTransfer interface [UltraLiteJ API], 163
SyncParms class [UltraLiteJ API], 252
setCacheSize method
ConfigPersistent interface [UltraLiteJ API], 96
setCertificateCompany method
StreamHTTPParms interface [UltraLiteJ API], 235
setCertificateName method
StreamHTTPParms interface [UltraLiteJ API], 235
setCertificateUnit method
StreamHTTPParms interface [UltraLiteJ API], 235
setConnectionString method [Android]
ConfigPersistent interface [UltraLiteJ API], 96
setDatabaseId method
Connection interface [UltraLiteJ API], 117
setDatabaseName method
Configuration interface [UltraLiteJ API], 101
setDownloadOnly method
SyncParms class [UltraLiteJ API], 253
setE2eePublicKey method [BlackBerry]
StreamHTTPParms interface [UltraLiteJ API], 224
setE2eeType method [BlackBerry]
StreamHTTPParms interface [UltraLiteJ API], 225
setEncryptionKey method
ConfigPersistent interface [UltraLiteJ API], 97
setErrorLanguage method
DatabaseManager class [UltraLiteJ API], 142
setExtraParameters method [Android]
StreamHTTPParms interface [UltraLiteJ API], 225
setHost method
StreamHTTPParms interface [UltraLiteJ API], 226
setHTTPUserIdAndPassword method [BlackBerry]
StreamHTTPParms interface [UltraLiteJ API], 226
setKeepPartialDownload method [Android]
SyncParms class [UltraLiteJ API], 253
setLazyLoadIndexes method [BlackBerry]
ConfigPersistent interface [UltraLiteJ API], 97
setLivenessTimeout method
FileTransfer interface [UltraLiteJ API], 164
SyncParms class [UltraLiteJ API], 254
setLocalFileName method
FileTransfer interface [UltraLiteJ API], 164
setLocalPath method
FileTransfer interface [UltraLiteJ API], 165
setNewPassword method
SyncParms class [UltraLiteJ API], 255
setNull method
DecimalNumber interface [UltraLiteJ API], 145
PreparedStatement interface [UltraLiteJ API], 191
UUIDValue interface [UltraLiteJ API], 284
setOption method
Connection interface [UltraLiteJ API], 118
setOutputBufferSize method
StreamHTTPParms interface [UltraLiteJ API], 227
setPageSize method
Configuration interface [UltraLiteJ API], 101
setPassword method
Configuration interface [UltraLiteJ API], 102
FileTransfer interface [UltraLiteJ API], 102
SyncParms class [UltraLiteJ API], 255
setPingOnly method
  SyncParms class [UltraLiteJ API], 256
setPort method
  StreamHTTPParms interface [UltraLiteJ API], 227
setPublications method
  SyncParms class [UltraLiteJ API], 256
setRemoteKey method
  FileTransfer interface [UltraLiteJ API], 166
setRestartable method
  StreamHTTPParms interface [UltraLiteJ API], 228
setResumePartialDownload method [Android]
  SyncParms class [UltraLiteJ API], 257
setResumePartialTransfer method
  FileTransfer interface [UltraLiteJ API], 167
setRowScoreFlushSize method [BlackBerry]
  ConfigPersistent interface [UltraLiteJ API], 98
setRowScoreMaximum method [BlackBerry]
  ConfigPersistent interface [UltraLiteJ API], 99
setServerFileName method
  FileTransfer interface [UltraLiteJ API], 167
setSyncObserver method
  Connection interface [UltraLiteJ API], 119
  SyncParms class [UltraLiteJ API], 257
setTableOrder method
  SyncParms class [UltraLiteJ API], 257
setTrustedCertificates method
  StreamHTTPParms interface [UltraLiteJ API], 236
setUploadOnly method
  SyncParms class [UltraLiteJ API], 258
setURLSuffix method
  StreamHTTPParms interface [UltraLiteJ API], 228
setUserName method
  FileTransfer interface [UltraLiteJ API], 168
  SyncParms class [UltraLiteJ API], 258
setUserName method [Android]
  ConfigPersistent interface [UltraLiteJ API], 99
setVersion method
  FileTransfer interface [UltraLiteJ API], 168
  SyncParms class [UltraLiteJ API], 259
setZlibCompression method
  StreamHTTPParms interface [UltraLiteJ API], 229
setZlibDownloadWindowSize method
  StreamHTTPParms interface [UltraLiteJ API], 229
setZlibUploadWindowSize method
  StreamHTTPParms interface [UltraLiteJ API], 230
SHORT variable
  Domain interface [UltraLiteJ API], 152
SISListener interface [BlackBerry] [UltraLiteJ API]
  description, 215
  startListening method, 215
  stopListening method, 215
SISRequestHandler interface [BlackBerry] [UltraLiteJ API]
  description, 216
  onError method, 216
  onRequest method, 216
SQLCODE
  UltraLiteJ error handling, 19
SQLInfo interface [Android] [UltraLiteJ API]
  description, 217
  getMessage method, 217
  getParameter method, 217
  getParameterCount method, 218
  getSQLCode method, 218
  getSQLCount method, 218
ST_GEOMETRY variable
  Domain interface [UltraLiteJ API], 152
STARTING variable
  SyncObserver.States interface [UltraLiteJ API], 242
startListening method
  SISListener interface [BlackBerry] [UltraLiteJ API], 215
stopListening method
  SISListener interface [BlackBerry] [UltraLiteJ API], 215
StreamHTTPParms interface [UltraLiteJ API]
  addCustomHTTPHeader method [BlackBerry], 220
  description, 218
  E2EE_RSA variable [BlackBerry], 230
  getCustomHTTPHeaders method [BlackBerry], 221
  getE2eePublicKey method [BlackBerry], 222
  getE2eeType method [BlackBerry], 222
  getExtraParameters method [Android], 222
  getHost method, 222
  getHTTPPassword method [BlackBerry], 223
  getHTTPUserId method [BlackBerry], 223
  getOutputBufferSize method, 223
  getPort method, 224
  getURLSuffix method, 224
  isRestartable method, 224
  setE2eePublicKey method [BlackBerry], 224
  setE2eeType method [BlackBerry], 225
setExtraParameters method [Android], 225
setHost method, 226
setHTTPUserAndPassword method [BlackBerry], 226
setOutputBufferSize method, 227
setPort method, 227
setRestartable method, 228
setURLSuffix method, 228
setZlibCompression method, 229
setZlibDownloadWindowSize method, 229
setZlibUploadWindowSize method, 230
zlibCompressionEnabled method, 230
StreamHTTPSParms interface [UltraLiteJ API]
description, 231
getCertificateCompany method, 234
getCertificateName method, 234
getCertificateUnit method, 234
getTrustedCertificates method, 234
setCertificateCompany method, 235
setCertificateName method, 235
setCertificateUnit method, 235
setTrustedCertificates method, 236
subtract method
DecimalNumber interface [UltraLiteJ API], 145
Unsigned64 class [UltraLiteJ API], 283
supported platforms
UltraLiteJ, 1
SYNC_ALL variable
Connection interface [UltraLiteJ API], 128
SYNC_ALL_DB_PUB_NAME variable
Connection interface [UltraLiteJ API], 128
SYNC_ALL_PUBS variable
Connection interface [UltraLiteJ API], 128
SYNC_COMPLETE_EVENT variable
ULjEvent interface [Android] [UltraLiteJ API], 277
synchronization
adding to Android application, 41
adding to BlackBerry application, 64
synchronize method
Connection interface [UltraLiteJ API], 119
synchronizing
UltraLiteJ, 21
SyncObserver interface [UltraLiteJ API]
description, 236
syncProgress method, 237
SyncObserver.States interface [UltraLiteJ API]
COMMITTING_DOWNLOAD variable, 239
CONNECTING variable, 239
description, 238
DISCONNECTING variable, 239
DONE variable, 240
ERROR variable, 240
FINISHING_UPLOAD variable, 240
RECEIVING_DATA variable, 240
RECEIVING_TABLE variable, 240
RECEIVING_UPLOAD_ACK variable, 241
ROLLING_BACK_DOWNLOAD variable, 241
SENDING_DATA variable, 241
SENDING_DOWNLOAD_ACK variable, 241
SENDING_HEADER variable, 241
SENDING_TABLE variable, 242
STARTING variable, 242
SyncParms class [UltraLiteJ API]
description, 242
getAcknowledgeDownload method, 245
getAdditionalParms method [Android], 245
getAuthenticationParms method, 246
getKeepPartialDownload method [Android], 246
getLivenessTimeout method, 246
getNewPassword method, 247
getPassword method, 247
getPublications method, 247
getResumePartialDownload method [Android], 247
getStreamParms method, 248
getSyncObserver method, 248
getSyncResult method, 249
getTableOrder method, 249
getUserName method, 250
getVersion method, 250
HTTP_STREAM variable, 260
HTTPS_STREAM variable, 260
isDownloadOnly method, 250
isPingOnly method, 250
isUploadOnly method, 251
setAcknowledgeDownload method, 251
setAdditionalParms method [Android], 251
setAuthenticationParms method, 252
setDownloadOnly method, 253
setKeepPartialDownload method [Android], 253
setLivenessTimeout method, 254
setNewPassword method, 255
setPassword method, 255
setPingOnly method, 256
setPublications method, 256
setResumePartialDownload method [Android], 257
setSyncObserver method, 257
setTableOrder method, 257
setUploadOnly method, 258
setUserName method, 258
setVersion method, 259
syncProgress method
SyncObserver interface [UltraLiteJ API], 237
SyncResult class [UltraLiteJ API]
description, 260
getAuthMessage method, 262
getAuthStatus method, 262
getAuthValue method, 262
getCurrentTableName method, 262
getIgnoredRows method, 263
gerenameDownloadRetained method [Android], 263
gerceivedByteCount method, 263
gerceivedDeletes method, 263
gerceivedIgnoredDeletes method, 264
gerceivedIgnoredUpdates method, 264
gerceivedInserts method, 265
gerceivedRowCount method, 265
gerceivedTruncateDeletes method, 265
gerceivedUpdates method, 266
generatedByteCount method, 266
generatedDeleles method, 267
generatedInserts method, 267
generatedUpdates method, 267
getStreamErrorCode method, 268
getStreamErrorMessage method, 268
generatedTableCount method, 268
gerceivedDownloadRowCount method, 269
gerceivedTableCount method, 269
isUploadOk method, 269
SyncResult.AuthStatusAuthStatus interface [UltraLiteJ API]
description, 269
EXPIRED variable, 270
IN_USE variable, 270
INVALID variable, 271
UNKNOWN variable, 271
VALID variable, 271
VALID_BUT_EXPIRES_SOON variable, 271
SYS_ARTICLES variable
TableSchema interface [UltraLiteJ API], 273
SYS_COLUMNS variable
TableSchema interface [UltraLiteJ API], 273
SYS_FKEY_COLUMNS variable
TableSchema interface [UltraLiteJ API], 273
SYS_FOREIGN_KEYS variable
TableSchema interface [UltraLiteJ API], 273
SYS_INDEX_COLUMNS variable
TableSchema interface [UltraLiteJ API], 273
SYS_INDEXES variable
TableSchema interface [UltraLiteJ API], 274
SYS_INTERNAL variable
TableSchema interface [UltraLiteJ API], 274
SYS_PRIMARY_INDEX variable
TableSchema interface [UltraLiteJ API], 274
SYS_PUBLICATIONS variable
TableSchema interface [UltraLiteJ API], 274
SYS_TABLES variable
TableSchema interface [UltraLiteJ API], 274
SYS_ULDATA variable
TableSchema interface [UltraLiteJ API], 274
SYS_ULDATA_INTERNAL variable
TableSchema interface [UltraLiteJ API], 275
SYS_ULDATA_OPTION variable
TableSchema interface [UltraLiteJ API], 275
SYS_ULDATA_PROPERTY variable
TableSchema interface [UltraLiteJ API], 275
system tables
UltraLiteJ, 36

TABLE_IS_DOWNLOAD_ONLY variable
TableSchema interface [UltraLiteJ API], 275
TABLE_IS_NOSYNC variable
TableSchema interface [UltraLiteJ API], 275
TABLE_IS_SYSTEM variable
TableSchema interface [UltraLiteJ API], 276
TABLE_MODIFIED_EVENT variable
ULjEvent interface [Android] [UltraLiteJ API], 277
tables
UltraLiteJ API schema information in, 19
TableSchema interface [UltraLiteJ API]
description, 271
SYS_ARTICLES variable, 273
SYS_COLUMNS variable, 273
SYS_FKEY_COLUMNS variable, 273
SYS_FOREIGN_KEYS variable, 273
SYS_INDEX_COLUMNS variable, 273
SYS_INDEXES variable, 274

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 307
SYS_INTERNAL variable, 274
SYS_PRIMARY_INDEX variable, 274
SYS_PUBLICATIONS variable, 274
SYS_TABLES variable, 274
SYS_ULDATA variable, 274
SYS_ULDATA_INTERNAL variable, 275
SYS_ULDATA_OPTION variable, 275
SYS_ULDATA_PROPERTY variable, 275
TABLE_IS_DOWNLOAD_ONLY variable, 275
TABLE_IS_NOSYNC variable, 275
TABLE_IS_SYSTEM variable, 276
TableSchema object
  UltraLiteJ development, 19
target platforms
  UltraLiteJ, 1
TIME variable
  Domain interface [UltraLiteJ API], 152
TIMESTAMP variable
  Domain interface [UltraLiteJ API], 153
TIMESTAMP_ZONE variable
  Domain interface [UltraLiteJ API], 153
TINY variable
  Domain interface [UltraLiteJ API], 153
transaction processing
  UltraLiteJ management, 14
transactions
  UltraLiteJ management, 14
troubleshooting
  UltraLiteJ handling errors, 19
tutorials
  building a BlackBerry application, 49
  building an Android application, 41
U
UL_VALID_BAD_ROWID variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 287
UL_VALID_CHECKING_INDEX variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 287
UL_VALID_CHECKING_PAGE variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 287
UL_VALID_CHECKING_TABLE variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 287
UL_VALID_CONNECT_ERROR variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 288
UL_VALID_CORRUPT_PAGE variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 288
UL_VALID_CORRUPT_PAGE_TABLE variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 288
UL_VALID_DATABASE_ERROR variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 289
UL_VALID_END variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 289
UL_VALID_FAILED_CHECKSUM variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 289
UL_VALID_INTERRUPTED variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 290
UL_VALID_NO_ERROR variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 290
UL_VALID_ROWCOUNT_MISMATCH variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 290
UL_VALID_START variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 290
UL_VALID_STARTUP_ERROR variable
  ValidateDatabaseProgressData.StatusId interface [Android] [UltraLiteJ API], 290
ULDatabaseSchema object
  UltraLiteJ development, 19
ULjEvent interface [Android] [UltraLiteJ API]
  COMMIT_EVENT variable, 277
description, 276
getParameter method, 276
type method, 277
SYNC_COMPLETE_EVENT variable, 277
TABLE_MODIFED_EVENT variable, 277
ULjException class [UltraLiteJ API]
description, 277
tCausingException method, 278
getErrorCode method, 278
getParameter method [Android], 278
getParameterCount method [Android], 279
getSqlOffset method, 279
UltraLite databases
    connecting in UltraLiteJ, 5
    UltraLiteJ API information access, 19
UltraLite Java edition databases
    connecting in UltraLiteJ, 5
    UltraLiteJ information access, 19
UltraLiteJ
    about, 3
    accessing schema information, 19
    Android application tutorial, 41
    Android CustDB sample, 30
    architecture, 79
    BlackBerry application tutorial, 49
    BlackBerry CustDB sample, 23
    creating a database, 49
    data modification, 14
    data modification with SQL, 8
    data retrieval, 17
    database stores, 5
    deployment, 27, 28
    example code, 30
    JAR resource files, 4
    Java edition databases, 4
    Network protocol options, 23
    persistence, 5
    quick start, 3
    supported platforms, 1
    synchronizing with MobiLink, 21
    system table schemas, 36
    transaction processing, 14
UltraLiteJ API
    ColumnSchema interface, 79
    ConfigFile interface, 85
    ConfigFileAndroid interface [Android], 86
    ConfigNonPersistent interface [BlackBerry], 88
    ConfigObjectStore interface [BlackBerry], 89
    ConfigPersistent interface, 91
    Configuration interface, 100
    Connection interface, 102
    DatabaseInfo interface, 130
    DatabaseManager class, 134
    DecimalNumber interface, 142
    Domain interface, 145
    FileTransfer interface, 155
    FileTransferProgressData interface, 170
    FileTransferProgressListener interface, 172
    IndexSchema interface, 173
    PreparedStatement interface, 175
    ResultSet interface, 191
    ResultSetMetadata interface, 209
    SISListener interface [BlackBerry], 215
    SISRequestHandler interface [BlackBerry], 216
    SQLInfo interface [Android], 217
    StreamHTTPParms interface, 218
    StreamHTTPSParms interface, 231
    SyncObserver interface, 236
    SyncObserver.States interface, 238
    SyncParms class, 242
    SyncResult class, 260
    SyncResult.AuthStatusCode interface, 269
    TableSchema interface, 271
    ULjEvent interface [Android], 276
    ULjException class, 277
    Unsigned64 class, 279
    UUIDValue interface, 283
    ValidateDatabaseProgressData interface [Android], 285
    ValidateDatabaseProgressData.StatusId interface [Android], 286
    ValidateDatabaseProgressListener interface [Android], 291
    UltraLiteJ API reference
    package names, 79
    ULVF_DATABASE variable [Android]
        Connection interface [UltraLiteJ API], 129
    ULVF_EXPRESS variable [Android]
        Connection interface [UltraLiteJ API], 129
    ULVF_FULL_VALIDATE variable [Android]
        Connection interface [UltraLiteJ API], 129
    ULVF_INDEX variable [Android]
        Connection interface [UltraLiteJ API], 129
    ULVF_TABLE variable [Android]
        Connection interface [UltraLiteJ API], 130
    UNIQUE_INDEX variable
        IndexSchema interface [UltraLiteJ API], 175
    UNIQUE_KEY variable
        IndexSchema interface [UltraLiteJ API], 175
    UNKNOWN variable
        SyncResult.AuthStatusCode interface [UltraLiteJ API], 271
    unregisterForEvent method [Android]
        Connection interface [UltraLiteJ API], 119
    Unsigned64 class [UltraLiteJ API]
        add method, 280
        compare method, 280
        description, 279
divide method, 281
multiply method, 281
remainder method, 282
subtract method, 283
UNSIGNED_BIG variable
   Domain interface [UltraLiteJ API], 153
UNSIGNED_INTEGER variable
   Domain interface [UltraLiteJ API], 154
UNSIGNED_SHORT variable
   Domain interface [UltraLiteJ API], 154
uploadFile method
   FileTransfer interface [UltraLiteJ API], 169
UUID variable
   Domain interface [UltraLiteJ API], 154
UUIDValue interface [UltraLiteJ API]
   description, 283
g(getString method, 284
isNull method, 284
set method, 284
setNull method, 284
VALID variable
   SyncResult.AuthStatusCode interface [UltraLiteJ API], 271
VALID_BUT_EXPIRES_SOON variable
   SyncResult.AuthStatusCode interface [UltraLiteJ API], 271
validateDatabase method [Android]
   Connection interface [UltraLiteJ API], 120
ValidateDatabaseProgressData interface [Android]
   description, 285
getParms method, 285
getStatusId method, 285
ValidateDatabaseProgressData.StatusId interface
   [Android] [UltraLiteJ API]
   description, 286
g(UL_VALID_BAD_ROWID variable, 287
UL_VALID_CHECKING_INDEX variable, 287
UL_VALID_CHECKING_PAGE variable, 287
UL_VALID_CHECKING_TABLE variable, 288
UL_VALID_CONNECT_ERROR variable, 288
UL_VALID_CORRUPT_PAGE variable, 288
UL_VALID_CORRUPT_PAGE_TABLE variable, 288
UL_VALID_DATABASE_ERROR variable, 289
UL_VALID_END variable, 289
UL_VALID_FAILED_CHECKSUM variable, 289
UL_VALID_INTERRUPTED variable, 290
UL_VALID_NO_ERROR variable, 290
UL_VALID_ROWCOUNT_MISMATCH variable, 290
UL_VALID_START variable, 290
UL_VALID_STARTUP_ERROR variable, 290
ValidateDatabaseProgressListener interface [Android]
   description, 291
validateProgressed method, 291
validateProgressed method
   ValidateDatabaseProgressListener interface
      [Android] [UltraLiteJ API], 291
VARCHAR variable
   Domain interface [UltraLiteJ API], 154
W
waitForEvent method [Android]
   Connection interface [UltraLiteJ API], 121
Z
zlibCompressionEnabled method
   StreamHTTPParms interface [UltraLiteJ API], 230