Oracle® Database Mobile Server

Mobile Client Guide

Release 12.1.0

E58642-01

January 2015

ORACLE®

Oracle Database Mobile Server Mobile Client Guide Release 12.1.0

E58642-01

Contents

Preface

This preface introduces you to the *Oracle Database Mobile Server Developer's Guide*, discussing the intended audience, documentation accessibility, and structure of this document.

Audience

This manual is intended for application developers as the primary audience and for database administrators who are interested in application development as the secondary audience.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

Use the following manuals and Websites as reference when installing and configuring Berkeley DB, SQLite or Java DB:

- *Getting Started with Java DB*, refer to: `http://docs.oracle.com/javadb/`, `http://db.apache.org/derby/` and `http://wiki.apache.org/db-derby/`
- *Berkeley DB Installation and Build Guide*
- *Getting Started with the Oracle Berkeley DB SQL APIs*
- *http://www.sqlite.org/*

Conventions

The following conventions are also used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.
`italic monospace`	Italic monospace type indicates a variable in a code example that you must replace. For example: `Driver=`*`install_dir`*`/lib/libtten.sl` Replace *`install_dir`* with the path of your TimesTen installation directory.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

1

Mobile Client Overview

Oracle Database Mobile Server delivers critical bi-directional data synchronization capability to mobile or fixed location distribution devices, while providing a centralized backend interface for managing mobile deployments. On the client device, the mobile client facilitates the transfer of data to and from the client database, which can be Berkeley DB, SQLite or Java DB. Install the desired database and the mobile client for Berkeley DB, SQLite or Java DB on your client device.

The following sections describe both databases and the mobile client for these databases:

- Section 1.1, "Mobile Client Architecture"
- Section 1.2, "Mobile Client for the Berkeley DB Database"
- Section 1.3, "Mobile Client for SQLite Database"
- Section 1.4, "Mobile Client for Java DB Database"
- Section 1.5, "Selecting Mobile Client"

1.1 Mobile Client Architecture

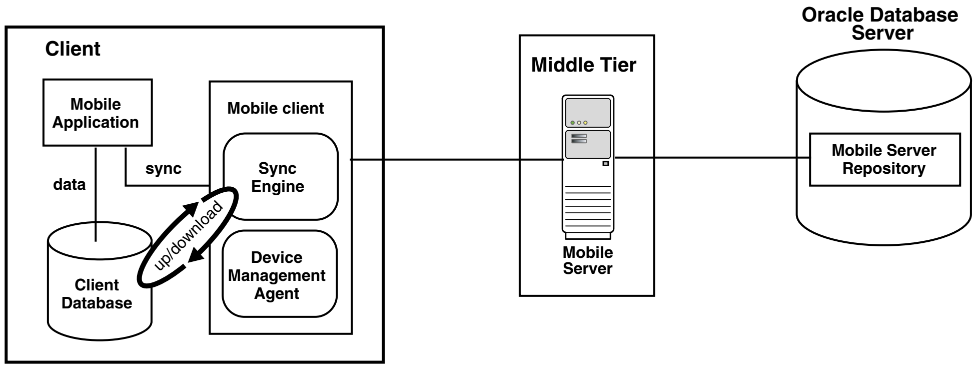
As shown in Figure 1–1, when both the client database and the mobile client are installed, the mobile device has the following components:

- Client database—The client database can be Berkeley DB with the SQL interface (commonly referred to as just Berkeley DB in this document), SQLite or Java DB.
- Mobile client—When you install the mobile client, the following components are provided:
  - Sync Engine—Automatic synchronization can be enabled on the Android, iOS, Win32, Windows Mobile, and Linux platforms. However, you can initiate manual synchronization within a mobile application on all platforms.

    The Sync Engine interacts with Berkeley DB, SQLite or Java DB databases to upload and download data in conjunction with the mobile server to synchronize the data with the Oracle database.
  - Device Manager Agent (DM Agent)—The mobile server uses the DM Agent to send commands to the mobile device for remote management. The DM Agent is installed on Android, iOS, Win32, Windows Mobile, and Linux platforms. The Blackberry, OJEC mobile client and pure Java client installed on standard Java SE platforms cannot be remotely managed.
- Mobile application—Interacts with the client database to manage the data and with the Sync Engine to initiate a manual synchronization.

The mobile client synchronizes the data in client database with the mobile server. The mobile server applies the client uploaded changes to back-end Oracle Database, and compose the server side changes and notify the client to download. This book describes how to configure, manage and implement synchronization using the mobile client. It does not discuss how to build, install, configure, manage or use the client databases.

Figure 1–1 Architecture for Device with a Mobile Client and Client Database



The following sections describe each mobile client:

- Section 1.2, "Mobile Client for the Berkeley DB Database"
- Section 1.3, "Mobile Client for SQLite Database"
- Section 1.4, "Mobile Client for Java DB Database"

1.2 Mobile Client for the Berkeley DB Database

Berkeley DB is a general-purpose, high-performance, embedded database that is designed for high-throughput applications. The primary goal of Berkeley DB is to deliver fast, scalable and flexible data management services to your application while remaining transparent to the end-user. Berkeley DB executes in the same process as your application.

Berkeley DB provides the following features that are expected of client/server enterprise-scale SQL databases: high throughput, high availability, high concurrency, replication, low-latency reads, non-blocking writes, failure recovery, data scalability, in-memory caching, ACID transactions, automatic and catastrophic recovery. Berkeley DB offers advanced features in a self-contained, small footprint software library.

The mobile client was built to use the Berkeley DB, which adds a SQL API to the Berkeley DB storage engine. The mobile client uses this interface to facilitate synchronization between the client and the back-end database.

1.3 Mobile Client for SQLite Database

SQLite is a small, compact, and self-contained database available on multiple platforms and available to the public. It has a small footprint and is easy to install and administer. In addition, many devices have SQLite already installed, including Android and Blackberry devices.

You can synchronize the data in one or more SQLite databases to a back-end Oracle database with the mobile client. This mobile client provides the ability to synchronize the data in SQLite databases with the Sync Engine contained within the mobile client.

SQLite is installed independently from the mobile client. SQLite does not provide the same SQL functionality as an Oracle database. This book describes how to configure, manage and implement synchronization using the mobile client. It does not discuss how to configure, manage or use SQLite. For information on SQLite and a full list of what functionality is supported, see `http://www.sqlite.org/`.

The SQLite Mobile Client can be installed on the following platforms: Linux, Windows (Win32), Windows Mobile, Android, iOS, and Blackberry platforms. Device management is supported on Android, iOS, Win32, Windows Mobile and Linux platforms. The Sync Engine supports both automatic and manual synchronization for SQLite. However, without device management support, remote device management and automatic synchronization is not supported on the Blackberry platform.

1.4 Mobile Client for Java DB Database

Java DB is a full--featured, small footprint SQL database, which comes bundled with Oracle JDKs. Java DB is pure Java and it runs everywhere that Java does, including the resource-constrained Java 8 compact profile 2 platform. It is easy to install Java DB, start using it, and even plug in custom, application-specific code. Java DB provides many enterprise database features, although it does not provide exactly the same functionality as Oracle databases. Java DB runs in a classic client-server configuration. It also runs embedded on the same JVM as the application. Java DB is a good fit for compile-once-deploy-everywhere applications which need multiple concurrent readers and writers.

Using the mobile client, which is installed independently, you can synchronize multiple Java DB databases with a back-end Oracle database. This guide describes how to configure, manage, and implement synchronization using the mobile client. The Java DB Mobile Client can be installed on Linux and Windows, its pure Java and can run on any Java SE platform. For more information on configuring, managing, and using Java DB, see `http://db.apache.org/derby/`.

1.5 Selecting Mobile Client

We have native mobile client, Android mobile client, iOS mobile client, Blackberry mobile client, pure Java SE mobile client, and pure Java ME mobile client.

- The native mobile client runs on Windows, Linux and Windows Mobile platforms, and it includes both Sync Engine and DM Agent components. The native mobile client can run with Berkeley DB and SQLite databases.
- The Android mobile client runs on Android platforms, and it includes the Sync Engine, the DMAgent application and the Update application. The Android mobile client can run with Berkeley DB and SQLite databases.
- The iOS mobile client runs on iOS platforms, and it includes both Sync Engine and DM Agent components. The iOS mobile client can run with Berkeley DB and SQLite databases.
- The Blackberry mobile client runs on Blackberry platforms, and it includes only Sync Engine component. The Blackberry mobile client can run with only SQLite database.
- The pure Java SE mobile client runs on Java SE platforms. A Java SE platform is a platform where Oracle Java SE can run. The pure Java SE mobile client includes only Sync Engine component. This mobile client can run with Berkeley DB, SQLite and Java DB databases.

- The pure Java ME mobile client runs on Java ME platforms. A Java ME platform is a platform where Oracle Java ME can run. The pure Java ME mobile client includes only Sync Engine component. This mobile client can run with both Berkeley DB and SQLite. With the OJEC client, the pure Java ME client is used with Berkeley DB database only, because the OJEC client embeds only Berkeley DB database.

Different mobile clients have different capability. The client is selected based on platform and client database being used.

2 Installing the Mobile Client

One of the benefits of Oracle Database Mobile Server is that you can have an application downloaded onto a device, where data can be synchronized between the device and the back-end Oracle database. When you install the mobile client, Oracle Database Mobile Server installs the Sync Engine and Device Manager Agent. The Device Manager Agent is not available for Java SE or ME (OJEC) clients.

Note: Any reference to the mobile client in this book apply to Java DB Mobile Client or SQLite Mobile Client.

All references of Berkeley DB refers to the Berkeley DB SQL Interface.

The following sections detail how to install the mobile client software on your client device:

- Section 2.1, "Supported Platforms and Requirements for the Mobile Client"
- Section 2.2, "Preparing the Device for a Mobile Application"
- Section 2.3, "Installing the Mobile Client"
- Section 2.4, "Configuring the Location of Mobile Client and Database Files"
- Section 2.5, "Configuring for Automatic Synchronization When Installing the Client"
- Section 2.6, "Uninstalling the Mobile Client"

See Chapter 1, "Oracle Database Mobile Server Management" in the *Oracle Database Mobile Server Administration and Deployment Guide* for information on how to manage functionality from the mobile server.

2.1 Supported Platforms and Requirements for the Mobile Client

The Berkeley DB and SQLite Mobile Clients are certified on the following platforms:

Table 2–1 Supported Platforms for Berkeley DB , SQLite and Java DB Mobile Clients

Platform	Berkeley DB Mobile Client	SQLite Mobile Client	Java DB Mobile Client
Microsoft Windows 7 (32-bit and 64-bit)	Yes	Yes	No
Microsoft Windows 2003 (64-bit)	Yes	Yes	No
Microsoft Windows 2008 R2 (64-bit)	Yes	Yes	No
Microsoft Windows 8 (32-bit and 64-bit)	Yes	Yes	No
Oracle Enterprise Linux 5.0, or 6.0 containing Unbreakable Enterprise Kernel (32-bit and 64-bit	Yes	Yes	No
OpenSUSE 13.1 (32-bit and 64-bit)	Yes	Yes	No
Ubuntu 14.04 (32-bit and 64-bit)	Yes	Yes	No
Fedora 20 (32-bit and 64-bit)	Yes	Yes	No
Windows Mobile 5.0, 6.0 and 6.5	Yes	Yes	No
Android 4.2, 4.3 and 4.4	Yes	Yes	No
Blackberry RIM 5.0 and 6.0	Yes	Yes	No
Java SE 1.7 and 1.8	Yes	Yes	Yes
Java ME (OJEC 1.1)	Yes	Yes	No
iOS 6.0, 7 and 8	Yes	Yes	No

Note: On 64-bit versions of Ubuntu, OpenSUSE and Fedora platforms, we need to install 32-bit supporting libraries to install and run Mobile Client. For more information, refer to Section 2.3.3.

Automatic synchronization and device management are available on most mobile client platforms. Table 2–2 displays what features are available on which platforms.

Table 2–2 Feature Support for Client Platforms

Platform	Automatic Synchronization	Device management through the DM Agent
Windows Mobile	Yes	Yes
Win32	Yes	Yes
Linux	Yes	Yes
Android	Yes	Yes
Blackberry	No	No

Table 2–2 (Cont.) Feature Support for Client Platforms

Platform	Automatic Synchronization	Device management through the DM Agent
Java SE 1.7 and 1.8	Yes	No
Java ME (OJEC 1.1)	Yes	No
iOS	Yes	No

Note: iOS mobile device management is supported on iOS 7 and 8. For more information on iOS mobile device management, refer to Section 7.11, *Using Mobile Manager to Manage iOS Devices*, and Chapter 11, *iOS Device Management* in *Administration and Deployment Guide*.

2.1.1 Certified Operating Systems and Other Software Requirements

The following tables detail the requirements for the client platforms on which you may install the mobile client. The requirements do not include requirements for either client database, but are only the requirements for the mobile client including the Sync Engine and Device Manager.

- Table 2–3, " BlackBerry and Android Platform Requirements"
- Table 2–4, " Software Requirements for Mobile Clients"
- Table 2–5, " Supported and Certified Technologies for Native Mobile Clients"
- Table 2–6, " Pocket PC and Windows Mobile Supported Platforms"
- Table 2–7, " Supported and Certified Technologies for Java Clients"

Table 2–3 BlackBerry and Android Platform Requirements

Platform	Minimum Storage for Mobile Client
BlackBerry	100 KB
Android	100 KB

Table 2–4 Software Requirements for Mobile Clients

Device Platform	Certified Operating System	Other Software Requirements
Win32	Windows 2003, Windows 2008, Windows 7, Windows 8	If using Java APIs for synchronization, use Oracle JDK 1.7 or 1.8 If implementing any .NET applications, use Microsoft .NET Framework 1.1 or 2.0
Windows Mobile	Windows Mobile 5 Windows Mobile 6 Windows Mobile 6.5 Windows Mobile Device Center 6.1	If using Java APIs for synchronization, use Oracle JDK 1.7 or 1.8. Windows Mobile Device Center 6.1. Microsoft.NET Compact Framework 3.5

Table 2–4 (Cont.) Software Requirements for Mobile Clients

Device Platform	Certified Operating System	Other Software Requirements
Linux	Oracle Enterprise Linux 5.0 or 6.0 containing Unbreakable Enterprise Kernel OpenSUSE 13.1 Ubuntu 14.04 Fedora 20	Oracle JDK 1.7or 1.8 Qt3

You should install all of the patches required for the JDK for the operating system. This is constantly under review and published on the JDK download page on the Oracle Java Web site.

2.1.2 Supported and Certified Technologies for Native Mobile Clients

The following are the supported and certified technologies for native mobile clients:

Note: Ensure that after you install the required software, the appropriate directories are included in the `PATH`. For example, after you install the JDK, ensure that the `JAVA_HOME/bin` is included in the `PATH`.

ADO.Net is supported for both the Berkeley DB and SQLite Mobile Clients. For more information on ADO.Net support, see the following URL:

`http://system.data.sqlite.org/index.html/doc/trunk/www/features.wiki`

Table 2–5 Supported and Certified Technologies for Native Mobile Clients

Device Platform	Supported Technologies	Certified Technologies
Win32	▪ ADO.Net ▪ JDBC ▪ ODBC	Oracle JDK 1.7 or 1.8
Windows Mobile	▪ Windows Mobile Device Center 6.1. ▪ ADO.Net ▪ JDBC ▪ ODBC	Oracle JDK 1.7 or 1.8
Linux	▪ JDBC ▪ ODBC	Oracle JDK 1.7 or 1.8

Table 2–6 Pocket PC and Windows Mobile Supported Platforms

Product Name	Windows Mobile Version	Chipsets	Mobile Client CAB File
`Windows Mobile 6`	`5.2.1236`	ARMV4I	PPC60 ARMV4I, <mobile_client>.cab
`Windows Mobile 6.5`	`5.2.23090`	ARMV4I	PPC60 ARMV4I, <mobile_client>.cab
`Windows Mobile 5`	`5.1.1700`	ARMV4I	PPC50 ARMV4I, <mobile_client>.cab

Table 2–7 Supported and Certified Technologies for Java Clients

Pure Java Clients Device Platform	Supported Technologies	Certified Technologies
Java SE 1.6, 1.7 or 1.8	JDBC	Berkeley DB 12.1.6.1.20, Ch-WernerJDBCdriver
		SQLite 3.8.5, Zentus JDBC driver
		Java DB embedded in JDK 1.8, Apache Derby Embedded JDBC driver
Java ME (CDC 1.1)	JSR169	OJEC 1.1

2.2 Preparing the Device for a Mobile Application

To execute mobile applications on a device, do the following:

1. Install the mobile client software that is appropriate for the client platform on your client machine. For example, install the SQLite WIN32 on a Windows 32 client machine.

   See Section 2.3, "Installing the Mobile Client" for a full description.

2. Download the user applications and its associated data.

   Synchronize the mobile client for the first time. Sign in with the user name/password of the mobile user who owns the mobile applications. The data for each application is retrieved.

   Notes: For the restrictions on creating the user name and password, see Section 4.3.1.2.1, "Define User Name and Password" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

   For more information about synchronization, see Chapter 5, "Managing Synchronization" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

3. You can now launch your applications from your client machine or from your mobile device.

2.3 Installing the Mobile Client

The following sections provide directions for the mobile client install:

- Section 2.3.1, "Installing the Mobile Client on Blackberry Devices"
- Section 2.3.2, "Installing the Mobile Client on Android Devices"
- Section 2.3.3, "Installing the Mobile Client for Win32, Windows Mobile or Linux"
- Section 2.3.4, "Installing Mobile Client with Multiple Languages"
- Section 2.3.5, "Installing the Mobile Client for Java SE"
- Section 2.3.6, "Installing the Mobile Client for Java ME"
- Section 2.3.7, "Installing iOS Mobile Client"

We do not support the following configuration scenarios:

- A mobile client and the Mobile Development Kit (MDK) cannot be installed on a single device.
- A client user cannot have more than one device.
- A mobile client cannot synchronize different types of client databases on the same mobile device.

2.3.1 Installing the Mobile Client on Blackberry Devices

To install the mobile client on Blackberry devices, perform the following:

Note: Applications cannot be downloaded to your Blackberry device from the mobile server, since device management is not supported for this device. You must download all applications to your Blackberry device as documented on the Blackberry Web site at `http://www.blackberry.com`.

1. Open a browser (on the Blackberry device), to point to the mobile server setup page using the following URL:

   ```
   http://<mobile_server>:<port>/mobile/setup
   ```

   Note: Substitute `https` if using HTTP over SSL.

   Figure 2–1 displays the mobile client setup page, which contains links to install mobile client software for multiple languages. You can select another language than English on the Language pulldown.
2. Click the mobile client for your language and the Blackberry client platform. This downloads and installs the mobile client.
3. Perform a manual synchronization for the mobile client.
4. Synchronization requires you to enter the user name and password for the mobile user. During the first synchronization, all data for this user is brought down and installed on your mobile device.

   Note: For information on the restrictions on creating the user name and password, see Section 4.3.1.2.1, "Define User Name and Password" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

2.3.2 Installing the Mobile Client on Android Devices

See the following subsections:

- Section 2.3.2.1, "Mobile Client Applications"
- Section 2.3.2.2, "Deploying Mobile Client on Android Device"
- Chapter 2.3.2.3, "The Mobile Server Sync Application."

2.3.2.1 Mobile Client Applications

Oracle Mobile Client on Android consists of three applications: Oracle Setup, Oracle DMAgent and Oracle Update. This is similar to how native clients function. Native clients have a setup program, a dmagent program and an update program.

- The Oracle Setup application is used to install the Android Mobile Client on the device. A user downloads the Oracle Setup apk from the Mobile Manager setup page, and installs it on the device. The execution of the Oracle Setup application will download the other two applications (the Oracle DMAgent and the Oracle Update).
- The Oracle DMAgent application is responsible for receiving remote commands from the Mobile Server, performing commands on the local device and sending back command execution results to the Mobile Server. The remote commands Android DMAgent accepts include: "retrieving software information", "retrieving device specific information", "trigger remote uninstall of the mobile client", "modify the configuration file on the device" and "Oracle Mobile client Update".
- The Oracle Update application is responsible for checking upgradable software on the Mobile Server. Each execution of this program will check if there is a new Mobile Client software, or a user application update. If updated software is found, a list will be presented to the user and the user can select which ones to install. Selected software will be downloaded to the mobile device. The user can decide not to install the new software right away, and check back later.

Oracle does not provide APIs for the DMAgent and Update functions. If the user wants to use that functionality, a Mobile Client has to be deployed on the device and those functions will be available. A user can also use the Oracle Mobile Sync functions without deploying the Mobile Client and deploy and manage their own application.

2.3.2.2 Deploying Mobile Client on Android Device

To install the mobile client on Android devices, perform the following:

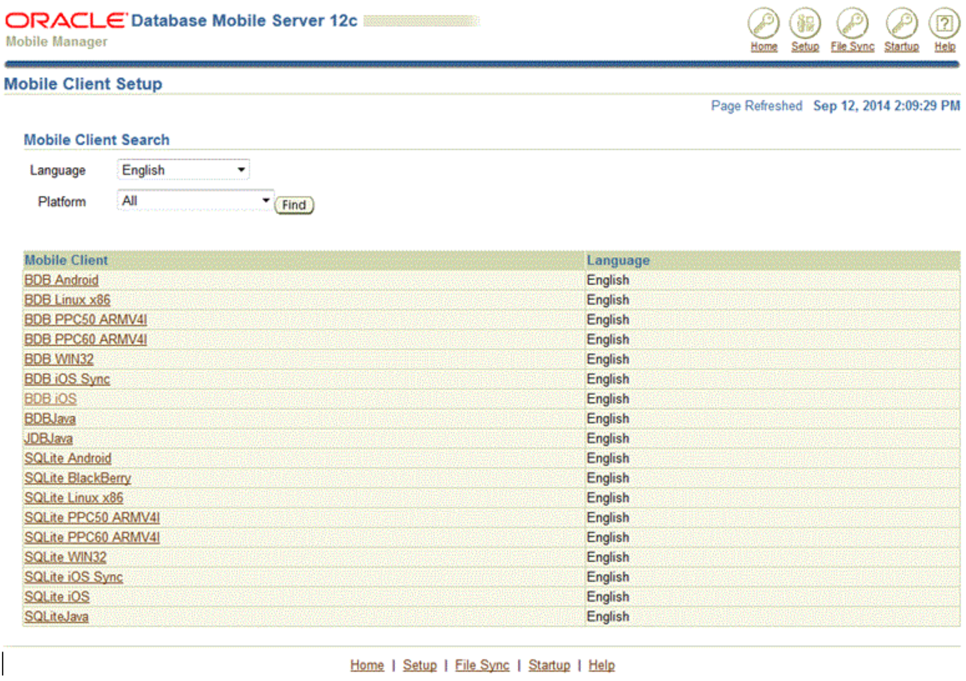
1. Open a browser (on the Android device), to point to the mobile server setup page using the following URL:

   ```
   http://<mobile_server>:<port>/mobile/setup
   ```

   Note: Substitute `https` if using HTTP over SSL.

   Figure 2–1 displays the mobile client setup page, which contains links to install mobile client software for multiple languages. You can select another language than English on the Language pulldown.

Figure 2–1 Mobile Client Setup Page



Mobile Client Setup

Page Refreshed Sep 12, 2014 2:09:29 PM

Mobile Client Search

Language: English

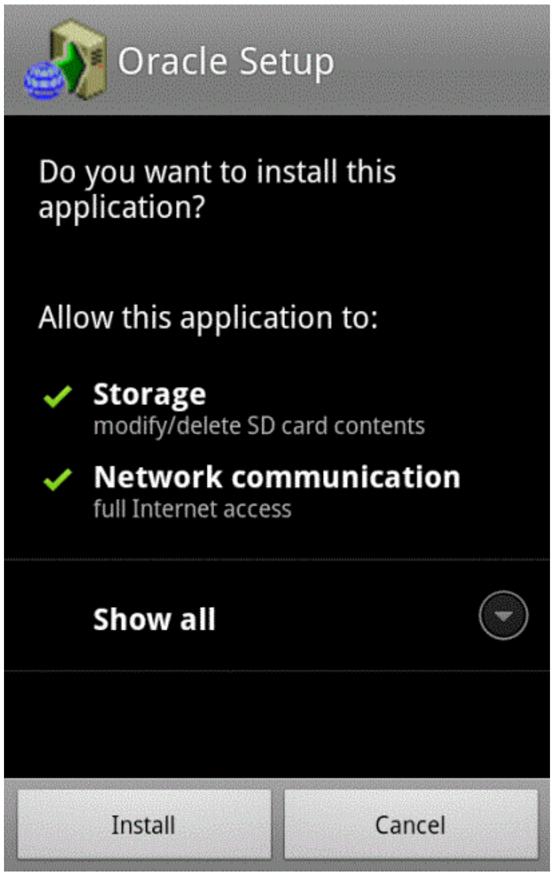
Platform: All

Mobile Client	Language
BDB Android	English
BDB Linux x86	English
BDB PPC50 ARMV4I	English
BDB PPC60 ARMV4I	English
BDB WIN32	English
BDB iOS Sync	English
BDB iOS	English
BDBJava	English
JDBJava	English
SQLite Android	English
SQLite BlackBerry	English
SQLite Linux x86	English
SQLite PPC50 ARMV4I	English
SQLite PPC60 ARMV4I	English
SQLite WIN32	English
SQLite iOS Sync	English
SQLite iOS	English
SQLiteJava	English

Home | Setup | File Sync | Startup | Help

2. Click the mobile client for your language and the Android client platform. This downloads the setup.apk for SQLite Android client and setup_bdb.apk for Berkeley DB Android client.
3. Bootstrap the setup_bdb.apk or setup.apk to install Berkeley DB Android or SQLite Android on the Android device. If you want to install the Oracle setup application, click the "Install" button when the system prompts you (see Figure 2–2).

Figure 2–2 Android Mobile Client Setup Page



Do you want to install this application?

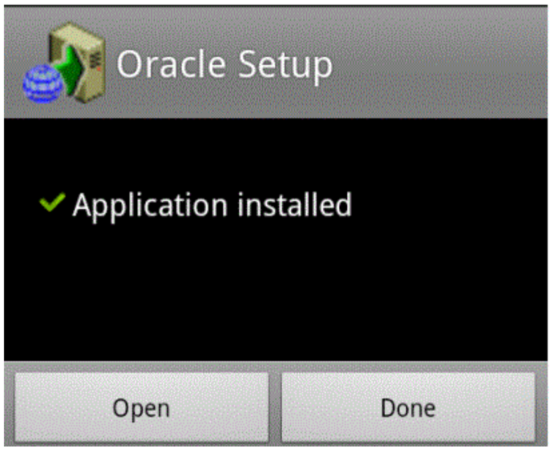
Allow this application to:

✓ **Storage**
modify/delete SD card contents

✓ **Network communication**
full Internet access

4. Figure 2–3 shows that the Oracle Setup application is installed successfully. Click the "Open" button to run the Oracle Setup application.

Figure 2–3 Android Setup Aplication Installed Page

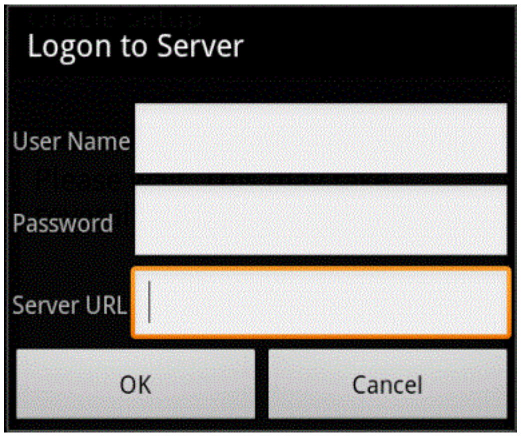


✓ Application installed

5. Figure 2–4 shows that the invocation of Oracle Setup application prompts you to input mobile user name, mobile user password and mobile server URL for registration purpose.

   Input the necessary information and click "OK".

Figure 2–4 Android Device Registration Page


Logon to Server

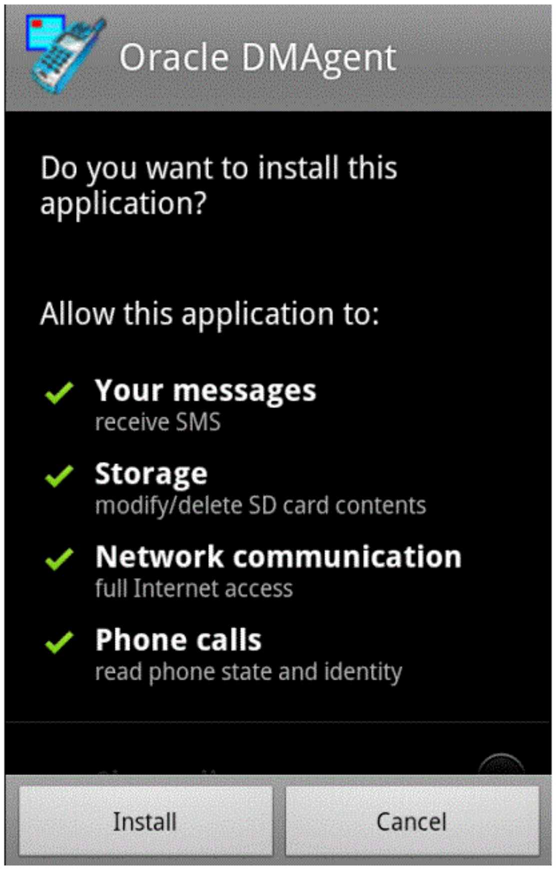
User Name

Password

Server URL

6. The Oracle Setup application will download two other applications on the Android device - Oracle DMAgent and Oracle Update, and will install them one by one.
7. Oracle Setup downloads Oracle DMAgent application to the Android device and installs it. Click the "Install" button in Figure 2–5 to install Oracle DMAgent application.

Figure 2–5 Install Oracle Android DMAgent Application Page


Oracle DMAgent

Do you want to install this application?

Allow this application to:

Your messages
receive SMS

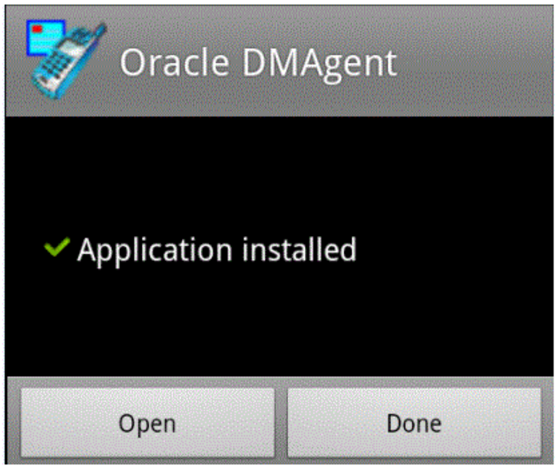
Storage
modify/delete SD card contents

Network communication
full Internet access

Phone calls
read phone state and identity

8. Figure 2–6 shows that the Oracle DMAgent application is installed successfully, Click the "Open" button to run the Oracle DMAgent application.

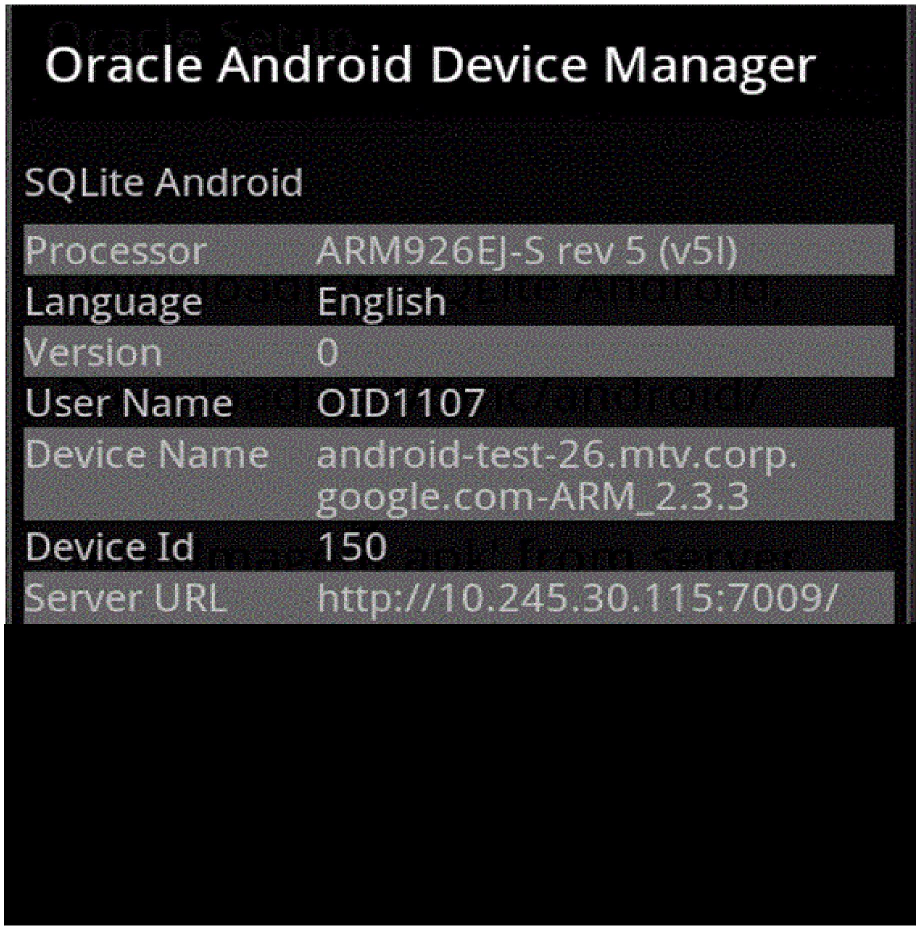
Figure 2–6 Oracle DMAgent Application Installed Page



9. When Oracle DMAgent starts, it displays information like Mobile Device Type (Berkeley DB Android or SQLite Android), Processor, Language, the software version of the Oracle Android Device Manager, the User Name, the Device Name, the Mobile Device Id and the Mobile Server URL, as Figure 2–7 Oracle Android Device Manager page demonstrates.

   Click "OK " to confirm this information.

Figure 2–7 Oracle Android Device Manager Page



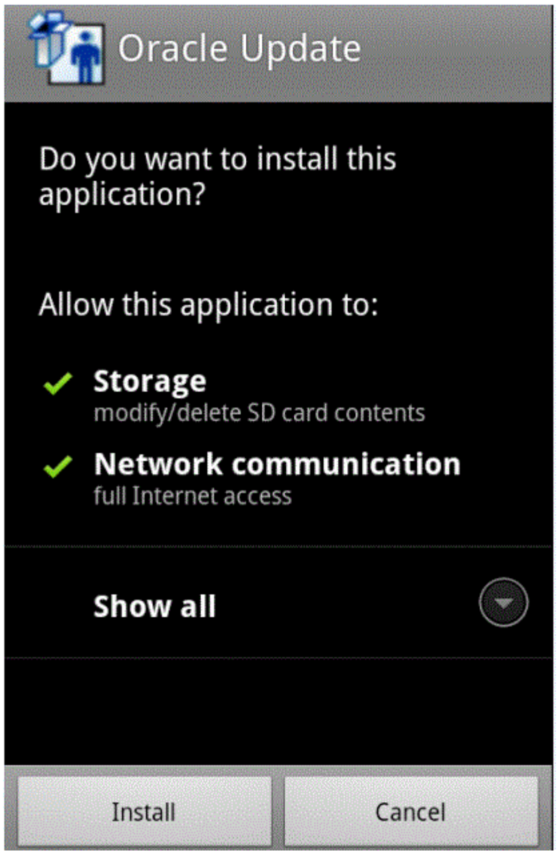
Oracle Android Device Manager

SQLite Android

Processor	ARM926EJ-S rev 5 (v5l)
Language	English
Version	0
User Name	OID1107
Device Name	android-test-26.mtv.corp.google.com-ARM_2.3.3
Device Id	150
Server URL	http://10.245.30.115:7009/

10. Oracle Setup also downloads Oracle Update application to the Android device and installs it. Click the "Install" button in Figure 2–8 to install Oracle Update application.

Figure 2–8 Install Oracle Update Application Page


Oracle Update

Do you want to install this application?

Allow this application to:

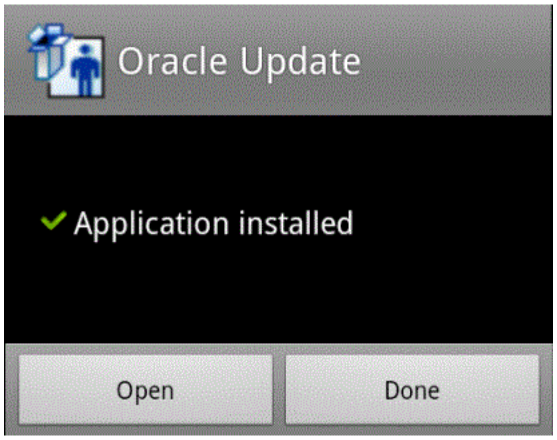
✓ **Storage**
modify/delete SD card contents

✓ **Network communication**
full Internet access

Show all

11. Figure 2–9 shows Oracle Update application installed successfully. Click "Open" button to invoke the Oracle Update application.

Figure 2–9 Oracle Update Application Installed Page


Oracle Update

✓ Application installed

12. The Oracle Update application will prompt you of any available software or application update. You can select to install or skip to install.

13. The Oracle Setup execution completes after downloading and installing Oracle DMAgent and Oracle Update applications.

2.3.2.3 The Mobile Server Sync Application

Android platforms require that any software downloaded to the device is digitally signed with a certificate whose private key is held by the application's developer. You can save files directly on the Android device's internal storage. By default, files saved to the internal storage are private to your application and other applications signed with different certificates cannot access them. The Oracle Mobile Sync application cannot be deployed by default on Android devices. Users integrating the Mobile Sync functions into their application must sign the application and download the application to the Android device.

For instructions on installing the Oracle Mobile Sync application on Android devices with an example of creating and downloading the Oracle Mobile Sync application, see Chapter 4, "Creating Sync Application for Android.".

2.3.3 Installing the Mobile Client for Win32, Windows Mobile or Linux

The mobile client libraries are 32-bit. To use the mobile client on a 64-bit Linux platform, 32-bit supporting libraries must be present. Some 64-bit Linux platforms have 32-bit libraries on install, but some 64-bit Linux platforms do not have 32-bit libraries. In such cases, the missing 32-bit supporting libraries must be manually installed.

Before you install the mobile client on your device, make sure that there is 1 MB of space available to download the `setup.exe`.

Note: Before installing the mobile client on a Linux platform, set MOBILE_CLIENT_HOME environment variable to the desired mobile client installation directory (default is: ~/mobileclient). Also, set environment variable PATH to include $MOBILE_CLIENT_HOME/bin and set environment variable LD_LIBRARY_PATH to include $MOBILE_CLIENT_HOME/bin, $JDK_HOME/jre/lib/i386/client and $JDK_HOME/jre/lib/i386.

To install the mobile client software, perform the following tasks.

Note: Any developer can modify how the client is installed before the installation with the INF file. For details on how to customize your Win32, Windows Mobile or Linux client, see Section 7.1, "Customize the Mobile Client Software Installation for Your Mobile Device" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

1. Open a browser (on the mobile client), to point to the mobile server using the following URL:

   ```
   http://<mobile_server>:<port>/mobile/setup
   ```

   Note: Substitute `https` if using HTTP over SSL.

Figure 2–1 displays the mobile client setup page, which contains links to install mobile client software for multiple platforms and languages.

- Language: Select a language other than English on the Language pulldown. English is the default.
- Platform: Choose to see all available platforms for the indicated language.

Client platforms are provided in the mobile client setup page. These client CAB files are optimized for size to minimize the footprint on your device.

Note: Available clients may differ from what is shown above.

2. Click the mobile client for your language and client platform.

Note:

- On Microsoft Windows and Windows Mobile platforms, all the messages on the client platform are in the same language as you specified during download.
- On Linux platform, all the messages on the client are in the same language as specified by the system's environment variable 'LANG' and not in the language that you have specified during download. For example, if your system's 'LANG' variable is set to English (for example, 'en_US'), even though you downloaded the German mobile client setup, the messages displayed will be in English. To avoid this discrepancy in the message language, ensure that the LANG and the downloaded mobile client setup language are same.

3. The "Save As" dialog box appears. The file name field displays the setup executable file for the selected platform as a .exe file type on Windows and Windows Mobile platform and a batch file type on Linux platform. Save the executable file to a directory on the client machine.

Note: For Windows Mobile, install any of the Oracle Database Mobile Server Windows Mobile platforms to ActiveSync or Windows Mobile Device Center (WMDC). Then, when the device is put into the cradle, ActiveSync or WMDC installs the Oracle Database Mobile Server on the device when it synchronizes.

4. Install the mobile client. For all platforms, except installing Windows Mobile on ActiveSync or Windows Mobile Device Center (WMDC), go to the directory where you saved the setup executable file.

Note: On Linux, some library dependencies must be solved first before installing the mobile client. For example, libjpeg.so.62 and libmng.so.1 may be missing on Ubuntu 14.04. This could be solved by: apt-get install libjpeg62 liblcms1 libmng1

Double-click the file to execute it.

5. Enter the user name and password for the mobile user.

Note: For information on the restrictions on creating the user name and password, see Section 4.3.1.2.1, "Define User Name and Password" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

6. You may be required to select the type of privilege under which to install the mobile client. This may already be designated by the administrator in the INF file before installation or the current user may have a privilege that defaults to a certain privilege for the installation.

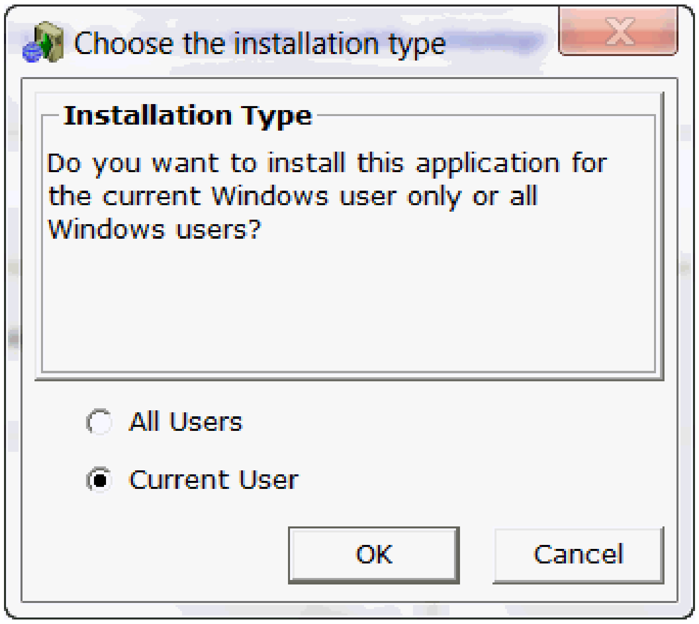
   - All Users—The user installing this mobile client has administrator privileges and can install the mobile client.

     Note: To avoid providing administrator privileges, create a directory named "Oracle Database Mobile Client" under the "<Windows Start Menu>\Programs" directory. The exact location of <Windows Start Menu> is dependent on what Windows version you are using.

   - Current User—Selecting this option designates that the user does not have administrator privileges, but can install and use the mobile client as a single user.

     Note: For details on how to designate the user privilege and for more information on user installation types, see Section 7.1, "Customize the Mobile Client Software Installation for Your Mobile Device" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

Figure 2–10 Select Installation Privileges


Choose the installation type

Installation Type

Do you want to install this application for the current Windows user only or all Windows users?

7. Provide the client directory name where to install the mobile client.
8. Once installed, synchronize the mobile client for the first time. During the first synchronization, all applications and data for this user is brought down and installed on your mobile client.
9. Each platform has further steps. See Table 2–8 for a description of the steps for each platform.

> **Note:** See Section 2.5, "Configuring for Automatic Synchronization When Installing the Client" for directions on how to enable a default synchronization after any client installation on your device.

Table 2–8 Initializing the First Synchronization for Each Mobile Client Platform

Mobile Client	Initial Synchronization Details
Windows Mobile	Perform the following steps. 1. If you install the Windows Mobile platform through ActiveSync or WMDC, insert the Windows Mobile device in the cradle. ActiveSync or WMDC performs a synchronization to install Oracle Database Mobile Client on the device. 2. After the mobile client is installed on the device, start the Device Manager Agent on the device either by selecting Device Manager in the programs group or by executing `dmagent.exe`, which is in the `orace` directory. 3. Enter the user name and password. If the mobile server URL field is empty, provide the URL as well. You can either enter the complete URL of the mobile server, the IP address and port number of the mobile server, or hostname and port number of the mobile server. If left off, the prefix "`http://`" is added automatically. Only use the hostname if the device is properly configured to use DNS name resolution. Otherwise, enter the IP address. The device is now registered with the mobile server and ready to be used.
All other platforms	Perform the following steps. 1. Locate the directories where you installed the runtime libraries, and launch the Mobile Sync application. 2. The `mSync` dialog appears. Enter the user name and password of the mobile user. If you do not know your user name and password, ask your system administrator, who creates users and assigns passwords to each user. In the "Server" field, enter the URL for your mobile server. Click "Apply" and click "Sync".

For information on installing mobile client with command line, see section:

- Section 2.3.3.1, "Installing the Mobile Client with Command Line"

2.3.3.1 Installing the Mobile Client with Command Line

Alternatively, you can install the Mobile Client for Win32, Windows Mobile or Linux with command line. The usage is:

```
setup [/q] [/nogui /v] <client platform name> <Mobile Server URL> <user
name> <password> <installation location> <client platform type> <install
type>
```

- The /q option skips the installation location prompt and uses the one provided on the command line.
- The /nogui option enables mobile client be installed silently without any popup dialogs. This option is only valid for Win32 and Linux clients.
- The /v option is valid when /nogui is specified, it prints out the installation progress messages on standard output.
- The client platform name/client platform type option checks the available platform names and types, go to *Mobile Manager -> Mobile Devices -> Platforms*, click on the platform name to see the platform info. The client platform name provided on the command line must be in the format: <platform name;language code>. For example, to install a Berkeley DB WIN32 client on a Windows system with English as the system language, provide "BDB WIN32;US" as the client platform name.
- The install type option is only required for Win32 platform. Specify install=PM to install the client for All Users, or specify install=PU to install the client for Current User.

The examples below show how to install a Berkeley DB client on Win32, Windows Mobile PPC60 and Linux platform:

-
  ```
  setup.exe /q "BDB WIN32;US" http://mobile_server_host:7001/mobile JOHN
  welcome1 C:\mobileclient WIN32_x86_US_BDB install=PU
  ```

-
  ```
  setup.exe /q "BDB PPC60 ARMV4I;US" http://mobile_server_
  host:7001/mobile JOHN welcome1 \OraCE WINCE_ARMV4I_US_BDB_60
  ```

-
  ```
  ./setup /q "BDB Linux x86;US" http://mobile_server_host:7001/mobile
  JOHN welcome1 /home/user/mobileclient LINUX_x86_US_BDB
  ```

The examples below show how to install the mobile client silently on Win32 and Linux platform:

-
  ```
  cmd /c setup.exe /nogui /v "BDB WIN32;US" http://mobile_server_
  host:7001/mobile JOHN welcome1 C:\mobileclient WIN32_x86_US_BDB
  install=PU
  ```

-
  ```
  ./setup /nogui /v "BDB Linux x86;US" http://mobile_server_
  host:7001/mobile JOHN welcome1 /home/user/mobileclient LINUX_x86_US_BDB
  ```

2.3.4 Installing Mobile Client with Multiple Languages

When you download the *setup* program to install mobile client on target platform, you can specify a Language. This determines the language that the mobile client software (msync, dmagent, syncagent and update) will use.

2.3.5 Installing the Mobile Client for Java SE

To install the mobile client for Java SE, perform the following:

Note: Mobile client for Java SE does not include device management component. Therefore, applications cannot be deployed by the mobile client for Java SE.

1. Open a browser to point to the mobile server setup page using the following URL:

   ```
   http://<mobile_server>:<port>/mobile/setup
   ```

   Figure 2–1 displays the mobile client setup page, which contains links to install Java SE mobile client software for multiple languages.

   > **Note:** The Java SE mobile client includes NLS resources for all supported locales in one jar archive.

2. Click BDBJava, JDBJava or SQLiteJava and download the arcivhe for the database you want to use.

   > **Note:** The jdbc drivers are not included with the client. They need to be installed and configured separately. Download the appropriate drivers supported on your particular platform.

   The synchronization APIs (refer to Section 3.1.1. "OSE Synchronization API for Applications on Mobile Clients" in the *Developer's Guide*), can be invoked from your own Java application, or you can use the included sync class: `oracle.opensync.tools.OSync`

   For usage details, refer to Section 3.2.1, "Synchronization for Pure Java Client".

2.3.6 Installing the Mobile Client for Java ME

The Java ME sync client is included with the MDK installation. It is located in:

```
<MOBILE_HOME>\Mobile\Sdk\j2me\ojec\osync_me.jar
```

Install this archive in your J2ME environment.

> **Note:** The sync client also needs JSR169 implementation on top of SQLite engine.

If you are using Oracle Java Embeded Client (OJEC 1.1), JSR169 drivers for Berkeley DB are bundled with it; therefore in this case, no external JDBC drivers needed.

> **Note:** J2ME CDC 1.1 spec (based on JDK 1.4 JCE) does not include RSA Ciphers which are needed for the encryption support on the mobile client.

You need to install a JCE provider with support for RSA cyphers. For example, you can use BouncyCastle JCE provider for JDK 1.4. In the case of OJEC 1.1, the JCE povider archive needs to be installed at the cvm lib\ext directory. Also, lib/security/java.security file needs to be modified to include the following line:

```
security.provider.[n]=org.bouncycastle.jce.provider.BouncyCastleProvider
```

Where **n** is a sequential number following the last number in the list of the existing providers already included in the file.

Once all the necessary archives are installed, the sync client can be invoked either from your application by using Java sync APIs (for information, see Section 3.1.1.1 "OSE

Synchronization Java API" in the *Developer's Guide*) or by invoking a command line tool:

```
./bin/cvm
-Xbootclasspath/a:./lib/sqlite.jar:./lib/jdbc.jar:./lib/jsr280.jar
-Dsun.boot.library.path=./lib -classpath ./lib/osync_me.jar
oracle.opensync.tools.OSync <USER_NAME> <PASSWORD> <SERVER_IP:SERVER_PORT
e.g. 127.0.0.1:7001>  -param OSE.FILES=YES
```

2.3.7 Installing iOS Mobile Client

iOS Mobile Client gets installed and runs entirely within the user's application, so no components need to be additionally installed on the device.

Note: Device management is not currently supported for iOS Mobile clients, so Section 2.5, "Configuring for Automatic Synchronization When Installing the Client" and Section 2.6, "Uninstalling the Mobile Client" of this guide do not apply for iOS clients.

However, to develop mobile application that uses mobile sync you need components such as header files, libraries and resources that you use to compile and link your application. These components are located in the mini-mdk package that you can download from the Mobile Server:

1. You need Mac computer to develop mobile sync application for iOS. Your Mac has to be running OS X v10.7 Lion or above depending on which iOS version you are targeting.

   You need to download and install Xcode IDE to develop the mobile sync application.

2. On your Mac, open a browser to point to the mobile server setup page using the following URL:

   ```
   http://<mobile_server>:<port>/mobile/setup
   ```

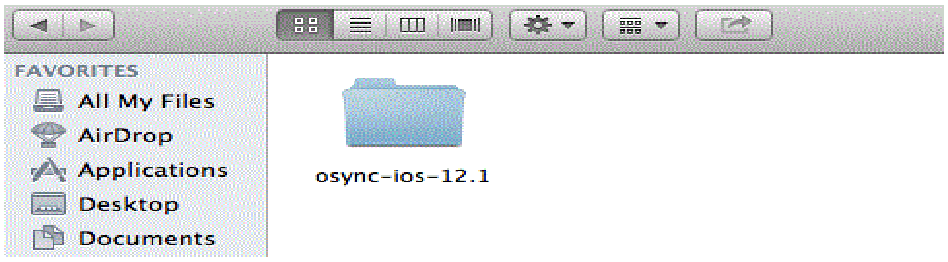
   Figure 2–1 displays the mobile client setup page.

   You can download either Sqlite iOS sync package or Berkeley DB iOS sync package (as highlighted above). The package file names are:

   - For Sqlite Sync Package: osync_ios.dmg
   - For Berkeley DB Sync Package: osync_bdb_ios.dmg

3. By default the package file will be downloaded into your "Downloads" directory. Open it. It will mount as a disk image on your desktop and you will also see the folder with its contents:

Figure 2–11 Installing iOS Mobile Sync Package



- For Sqlite Sync Package: osync-ios-12.1
- For Berkeley DB Sync Package: osync-bdb-ios-12.1

4. Open a "Finder" window and locate the directory where you want to place the package. Then drag the package folder from disk image contents folder to that location. Now the package is installed and you can move the disk image icon to the "Trash" to unmount it.

5. The package folder (osync-ios-12.1 or osync-bdb-ios-12.1) has the components you need to develop mobile sync application. For information on the description of the components and how to use them in your application, see Chapter 6, "Creating Sync Application for iOS".

2.4 Configuring the Location of Mobile Client and Database Files

The location of the client database is determined by the `DATA_DIRECTORY` parameter in the `OSE.INI` file. Refer to Appendix A.1.2.1, "DATA_DIRECTORY," for default values for this parameter.

- All client databases are stored in the `DATA_DIRECTORY/`*`<user>`* directory, where *`<user>`* is the synchronization user id.

  For Berkeley DB and SQLite, client databases are named with the .db extension, such as TERRY/myapp.db, where TERRY is the synchronization user id, and myapp is the client database name, which is specified during the creation of publication. For Java DB, all client databases are stored in files that live in a directory of the same name as the client database. For above example, there will be a directory named TERRY/myapp, and under that directory, there will be files storing client databases. If a synchronization user has two publications A and B, then for Berkeley DB and SQLite, there will be two client database files, A.db and B.db, under the <user> folder. For Java DB, there will be two sub directories under the <user> folder, each of <user>/A and <user>/B directories represents a single database.

- Internal configuration files for the mobile client are stored in the `MOBILE_CLIENT_HOME/bin/oseconf` directory.

The following shows an example of configuring the client database directory on a Win32 platform for Berkeley DB and SQLite:

```
SQLITE.DATA_DIRECTORY=C:\mobileclient\sqlite\sqlite_db
```

The following shows an example of configuring the client database directory on Java SE platform for Java DB:

```
JAVADB.DATA_DIRECTORY=c:\mobileclient\javadb_home
```

For more details on this parameter, see Appendix A.1.2.1, "DATA_DIRECTORY".

Note: Differences for iOS clients:

For iOS clients, all client databases and other synchronization-related data has to be stored within the iOS application sandbox. The directory for mobile client in iOS application sandbox is "Library/Application Support/oracle" relative to the sandbox root. All databases and synchronization-related files are stored in that directory and its subdirectories.

The default location of database files is "Library/Application Support/oracle/sqlite_db/<user>" and the location of internal configuration files is "Library/Application Support/oracle/oseconf".

Note that the value of DATA_DIRECTORY parameter in ose.ini must be specified relative to the application sandbox root.

2.5 Configuring for Automatic Synchronization When Installing the Client

In the default configuration, mobile clients do not automatically synchronize after you install the client. However, for Win32, Windows Mobile or Linux platforms, you can modify your configuration to automatically synchronize each client after it is installed, as follows:

1. Logon to the mobile server as an administrator and launch the Mobile Manager tool.
2. Click on "Mobile Devices", followed by "Administration".
3. Click on "Command Management".
4. Edit the "Command Device Info" (Retrieve device information).
5. Insert "Synchronize" as a Selected Command and click "Apply" to accept the changes.

See Section 7.5, "Sending Commands to Your Mobile Devices" in the *Oracle Database Mobile Server Administration and Deployment Guide* for more details on sending commands to your mobile device.

2.6 Uninstalling the Mobile Client

The following sections provide directions for the mobile client uninstall:

- Section 2.6.1, "Uninstalling the Native Mobile Client"
- Section 2.6.2, "Uninstalling the Android Client"

2.6.1 Uninstalling the Native Mobile Client

When you want to uninstall the mobile client, execute the `uninst executable` that is located in the install directory for the mobile client.

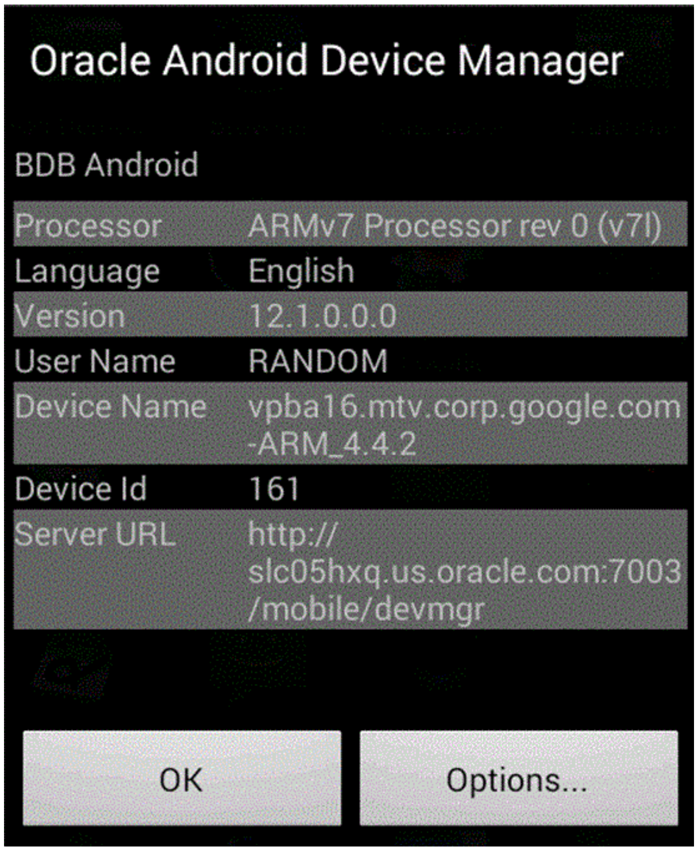
Alternatively, you can issue "uninst /q" command to uninstall the Mobile Client. The "/q" option skips the "Delete Mobile Client" prompt and "Delete Database" prompt during uninstallation.

2.6.2 Uninstalling the Android Client

Follow below steps to uninstall mobile client on Android device:.

1. Select Oracle DMAgent, and click "Options..." button in Figure 2–12.

Figure 2–12 Oracle Android Device Manager Screen



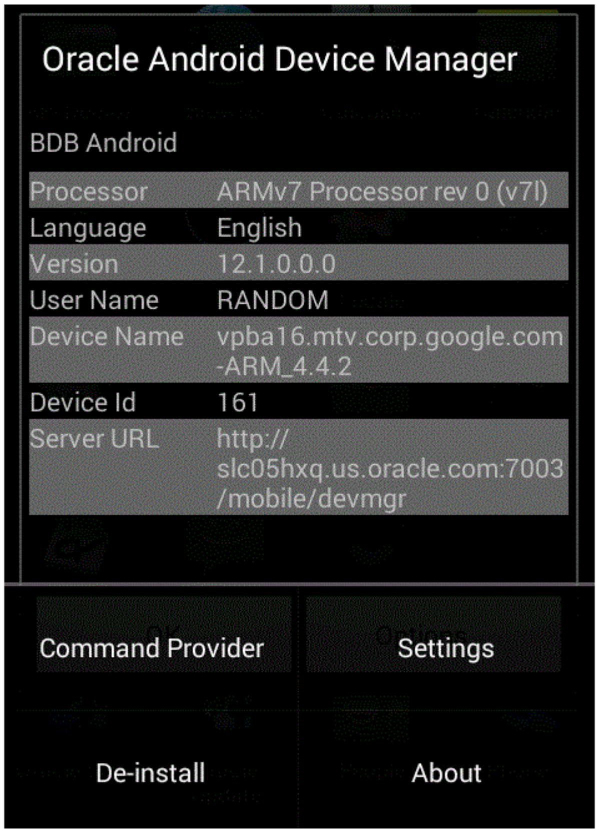
Oracle Android Device Manager

BDB Android

Processor	ARMv7 Processor rev 0 (v7l)
Language	English
Version	12.1.0.0.0
User Name	RANDOM
Device Name	vpba16.mtv.corp.google.com -ARM_4.4.2
Device Id	161
Server URL	http:// slc05hxq.us.oracle.com:7003 /mobile/devmgr

2. Click "De-install" button in Figure 2–13.

Figure 2–13 De-Install Oracle Android Device Manager



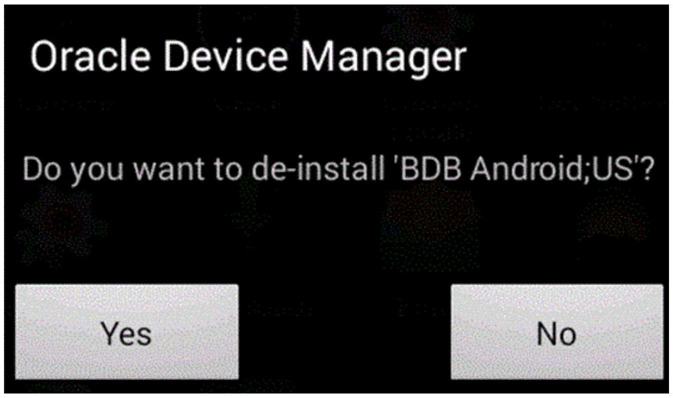
Oracle Android Device Manager

BDB Android

Processor	ARMv7 Processor rev 0 (v7l)
Language	English
Version	12.1.0.0.0
User Name	RANDOM
Device Name	vpba16.mtv.corp.google.com -ARM_4.4.2
Device Id	161
Server URL	http:// slc05hxq.us.oracle.com:7003 /mobile/devmgr

3. Select Yes" to question in Figure 2–14 and all following questions.

Figure 2–14 De-installation Process



Oracle Device Manager

Do you want to de-install 'BDB Android;US'?

3 Managing Your Mobile Client

The following sections describe how to manage the Oracle Database Mobile Server functionality on the mobile client:

3.1 Starting the Mobile Client

When you installed the mobile client on Linux or Windows, it is configured so that the mobile client always starts automatically when the device is initiated.

3.2 Synchronize Data for Applications on the Mobile Client

You can have an application downloaded onto a device, where data can be synchronized between the mobile client and the back-end Oracle database.

The following describes how to initiate synchronization from each type of mobile client:

- Java SE and Java ME clients: Refer to Chapter 5, "Synchronization Utilities for Java Sync Client" for information on how to invoke synchronization for pure java clients.
- Blackberry and Android clients: The application built for these clients initiate synchronization by executing the Mobile Client's Java APIs. For details on synchronization APIs, see Chapter 2, "Synchronization" and Chapter 3, "Managing Synchronization on the Mobile Client" in the *Oracle Database Mobile Server*

Developer's Guide for more information. For full details on the Java APIs, see the Javadoc

- Win32, Linux, Windows Mobile and iOS clients: The application built for these clients can use the Mobile Client's Java APIs or C/C++ APIs. Thus, start the application as you would start any application on these platforms.

Note: When you initiate a synchronization from the client, either manually or by scheduling a job, the synchronization cannot occur if there is an active connection with an uncommitted transaction opened from another source. This could be from scheduling two jobs to synchronize at the same time, from mSync, or the client synchronization APIs.

Note: iOS clients: Applications using these clients can use C/C++ APIs to manage manual and automatic synchronization, as described in Chapter 3, "Managing Synchronization on the Mobile Client" in the *Oracle Database Mobile Server Developer's Guide*.

Initiate synchronization through one of the following methods:

- Execute the `msync` executable, described in Section 3.3, "Use the mSync GUI to Initiate Synchronization".

Note: msync or any other UI tools are not currently supported for iOS clients.

- Implement synchronization within your application using the synchronization APIs, as described in Chapter 2, "Synchronization" and Chapter 3, "Managing Synchronization on the Mobile Client" in the *Oracle Database Mobile Server Developer's Guide*.

Note: The mobile client device clock must be accurate for the time zone set on the device before attempting to synchronize. An inaccurate time may result in the following exception during synchronization: `CNS: 9026 "Wrong user name or password. Please enter correct value and reSync."`

For more information, see Section 3.2.1, "Synchronization for Pure Java Client."

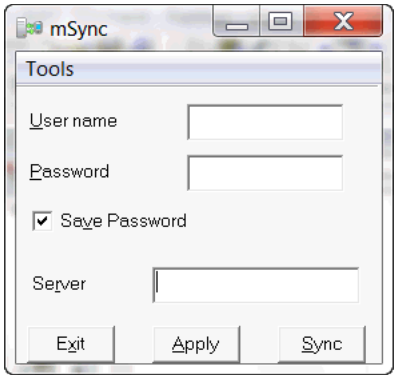
3.2.1 Synchronization for Pure Java Client

Refer to Chapter 5, "Synchronization Utilities for Java Sync Client" for information on how to invoke synchronization for pure java clients.

3.3 Use the mSync GUI to Initiate Synchronization

You can initiate synchronization for the native mobile client using the mSync GUI, as shown in Figure 3–1.

Figure 3–1 Using the mSync GUI to Initiate Synchronization



To bring up the mSync GUI, execute `msync.exe` on Win32 and Windows Mobile or `msync` on Linux, which is located in the `/bin` subdirectory under the directory where you installed the mobile client. For Blackberry and Android platforms, start mSync by clicking the mSync application icon.

Modify the following supplied values, if incorrect:

- User name and password for the user that is starting the synchronization.

> **Note:** See Section 4.3.1.2.1, "Define User Name and Password" in the *Oracle Database Mobile Server Administration and Deployment Guide* for conventions for creating the user name or password.

- Check if you want the password saved for future requests.
- Mobile Server URL, which is http://<mobile_server_hostname>:<port_number>, replace 'http' with 'https' in case SSL is used. If SSL is not used, 'http://' can be skipped.

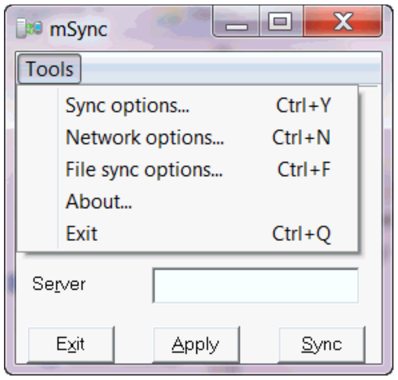
Click "Sync" to start the Synchronization. Click "Apply" to save any modifications you made to the entries. Click "Exit" to leave the tool.

If there are software updates that are waiting to be downloaded to the client, then the update tool is automatically executed after the end of the synchronization process. See Section 3.10, "Initiate Updates for the Mobile Client" for more information.

> **Note:** The only time that the client does not check for software updates is if you are using the Synchronization APIs. If you want to launch the update UI, then enter `update` on the command line.

You can also modify the tool options by selecting the "Tools" menu, as shown in Figure 3–2.

Figure 3–2 The mSync Tools Selection



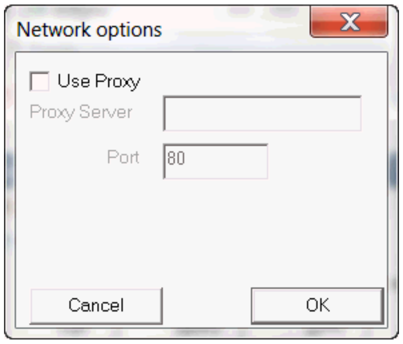
The following sections describe the Tools options:

- Section 3.3.1, "Network Options for MSync Tool"
- Section 3.3.2, "Sync Options for MSync Tool"
- Section 3.3.3, "Sync to a File Using File-Based Sync"
- Section 3.3.4, "Use Mobile Client Tools on Linux"

3.3.1 Network Options for MSync Tool

Figure 3–3 displays the Network options screen where you can specify a proxy if your network provider requires that you use a proxy server to access the internet. Click "Use Proxy" to use a proxy and then enter the proxy server and port number.

Figure 3–3 The mSync Network Options Selection



Port: 80

3.3.2 Sync Options for MSync Tool

Figure 3–4 displays the Sync Options screen where you can specify the following:

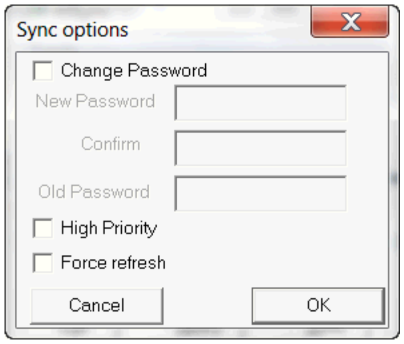
- Mobile User Password—Modify the existing password. The mobile user password is stored on both the client and the mobile server. To ensure that both are modified, only change the password when connected to the mobile server. See Section 3.4, "Synchronization Mechanisms on Mobile Client" for details.
- High Priority—Select this checkbox to specify synchronizing only High Priority data. This specifies under what conditions the different priority records are synchronized. By default, the value is LOW, which is synchronized last. If you have a very low network bandwidth and a high ping delay, you may only want to synchronize your HIGH priority data.

When you select this checkbox, you are enabling pre-defined high priority records to be synchronized first. This only applies for those publication items that have specified a restricting predicate. See Section 1.2.10, "Priority-Based Replication" in the *Oracle Database Mobile Server Troubleshooting and Tuning Guide* for more information.

- Force Refresh—The force refresh option is an emergency only synchronization option. Check this option when a client is corrupt or malfunctioning, so that you decide to replace the mobile client data with a fresh copy of data from the enterprise data store with the forced refresh. When this option is selected, any data transactions that have been made on the client are lost.

  When a force refresh is initiated all data on the client is removed. The client then brings down an accurate copy of the client data from the enterprise database to start fresh with exactly what is currently stored in the enterprise data store.

Figure 3–4 The mSync Options Selection



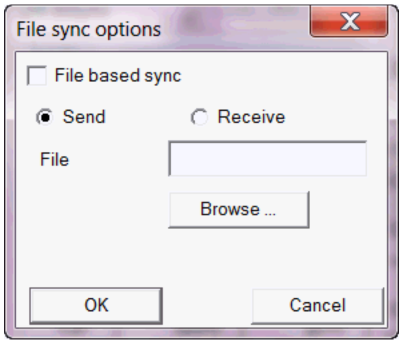
3.3.3 Sync to a File Using File-Based Sync

Once you select "File Based Sync" off the Tools menu, the screen shown in Figure 3–5 is displayed. To synchronize to a file, click on the "File based sync" checkbox and perform the following:

- If you select the send radio button, then browse for a directory where you want the client to save the upload data file from the mobile client for the mobile server.
- If you select the receive radio button, then provide the location for the download data file from the mobile server.

For full details on File-Based Sync, see Section 5.10, "Synchronizing to a File with File-Based Sync" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

Figure 3–5 File Sync Options



3.3.4 Use Mobile Client Tools on Linux

The mobile client for Linux supports the `msync`, `dmagent`, `update` and `autosync` tools. To use the UI-based tools, use the following executables: `msync`, `dmagent`, `update`, or `autosync`.

To synchronize on a Linux client with the command line tool, use the `msync` executable for synchronization, as follows:

```
./msync username/password@http://server[:port][@proxy:port]
```

Note: Substitute https if using HTTP over SSL.

For example,

```
./msync john/john@testserver:8000
```

The other `msync` options, such as `-save`, `-a`, `-password` and `-force` currently will not result in a successful sync. This is a limitation only for the `msync` executable in the MDK installation on Linux.

3.4 Synchronization Mechanisms on Mobile Client

SQLite, Berkeley DB and Java DB clients support two types of synchronization: state-based (no queues) and queue-based. Both modes implement change capture but use different mechanisms to upload client changes to the server and download server changes to the client. In state-based mode, client changes are taken directly from snapshots and server changes are applied directly to the snapshots. This mechanism is efficient but the drawback is that all snapshots are locked during sync session so that users's applications cannot read or modify snapshots during the sync session.

In queue-based mode, synchronization is split into 3 tasks:

- Compose task: client changes are taken from snapshots and put into the outqueue.
- Sync task: client changes in the outqueue are uploaded to the server, and the server changes are dowloaded into the inqueue.
- Apply task: server changes in the inqueue are applied to the snapshots.

The advantage of this mode is that Sync task (the most lengthy of the 3, since it usually involves network trasfers) does not lock the snapshots. During execution of sync task, user's application is free to read or modify the snapshots.

Note: Compose and apply tasks still require the snapshots to be locked.

Also, queue-based mode is transaction-based: client and server changes are split into atomic transactions, each identified by a unique transaction id. This allows compose, sync and apply tasks to run asynchronously. In addition, compose and apply tasks are performed on per-database basis. The main application of this structure is automatic rule-based synchronization that is performed by sync agent. The disadvangage compared to the state-based mode is that more time is needed because of additional compose and apply tasks.

Note: In queue-based mode, foreground sync (the sync done through msync tool or sync APIs) invokes compose, sync and apply tasks. These tasks are run in sequence: first compose is performed for each database, then sync task and then apply is performed for each database. The snapshots are locked during the whole foreground sync session.

The mode is controlled by ose.ini parameter SQLITE.QUEUES for Berkeley DB and SQLite databases, or JAVADB.QUEUES for Java DB database, which can be set to YES or NO. Note that the queue-based mode is the default. To enable state-based mode, set this parameter in ose.ini to NO.

Note: Currently this setting is permanent. It needs to be set before the first sync and it cannot be changed after that unless the client is removed and reinstalled or all client database are removed (changing SQLITE.QUEUES or JAVADB.QUEUES parameter after the client databases have been used may result in unpredictable behavior).

For more details on this parameter, see Appendix A.1.2.2, "QUEUES".

The following section describe metadata structure that is used to implement state-based and queue-based mechanisms:

- Section 3.4.1, "State-based Mode"
- Section 3.4.2, "Queue-based mode"

3.4.1 State-based Mode

For each snapshot a state table is created that references each record in the snapshot through primary key columns of that snapshot. For example, for a snapshot TABLE1, a state table OSE_ST$TABLE1 is created with primary key columns identical to that of TABLE1. Additional columns are created for OSE_ST$TABLE1 and are used for change capture, server and client versioning, and data priority.

Note: The above applies only to updatable snapshots. For read-only snapshots, the state table is not created. For more information about read-only and updatable snapshots, see Section 2.3.1.1.1, "Read-only Snapshots" and Section 2.3.1.1.2, "Updatable Snapshots" of the *Mobile Server Developer's Guide*.

For versioning, the state table contains the details for all the records in the snapshot and not just records modified by the client. These records are populated and updated with the server's data and versions. The records that are not updated by the client are marked as "clean" (see Section 3.4.1.1).

Another tables used by sync are OSE$TABLES and OSE$TRSEQ. OSE$TABLES meta-table exists in each database where snapshots reside and describes meta-information for every snapshot in the given database. OSE$TRSEQ also exists in each database and contains acknowledgement sequences needed by the server. These tables are only used by sync internally and should not be modified by the user.

The following sections describe state table columns and how data priority should be set by user's application:

- Section 3.4.1.1, "State Table Columns (except for the primary key columns)"
- Section 3.4.1.2, "Data Priority Handling"

3.4.1.1 State Table Columns (except for the primary key columns)

- OSE$STATE - column to indicate DML type of the record, INSERT ("I"), UPDATE ("U"), DELETE ("D") or CLEAN ("C") ("Clean" meaning, that the record was not updated by client since last sync)
- OSE$PRIO - priority of the record, set to 0 to indicate high priority, 1 to indicate normal priority
- OSE$SVER - record's server version
- OSE$CVER - record's client version

3.4.1.2 Data Priority Handling

The snapshot table itself does not contain extra column(s) to indicate record priority. It is up to the applications to choose what the priority of a client record should be. Currently two priorities are supported - high (value 0) and normal (value 1). The state table for each updatable snapshot includes priority column as described in the previous section. The default priority is normal (value 1). If an application wishes to make a given record high priority, it can find the corresponding state table record by using snapshot record's primary key and set the value of OSE$PRIO column to 0. It can make the record normal priority by setting this value to 1.

Note: This only applies to the records that were modified on the client, because only those record's changes will be uploaded to the server during sync.

Usually, application will have its own criteria on which records need to be of high priority, decided by the application data in the snapshot's record. For example, it could decide that records with certain department id in the employees snapshot table must be of high priority. In this kind of cases, the application to create additional triggers on the snapshot table that will appropriately set OSE$PRIO column in the state table when a new record is inserted or a record is updated in the snapshot table.

It is also common to setup automatic sync rules such that high priority records will be uploaded to the server more promptly than normal priority records. See Section 4.5, "Define the Rules Under Which the Automatic Synchronization Starts" of the *Mobile Server Developer's Guide* for information on automatic synchronization rules.

3.4.2 Queue-based mode

Queue-based mode is a superset of state-based mode: change capture on the client is done the same way as above using state table and triggers, but instead of being synced directly, client changes are put into the outqueue during compose task. The apply task will propagate server changes from the inqueue to the snapshots and will populate their state tables as well.

For Berkeley DB and SQLite, the queue tables for each database are stored in a separate database, "queue database". Each database with snapshot data will have it's own corresponding queue database with the same name prefixed by "OSE$". For example, database TERRY/TESTDB.db, where TERRY is the synchronization id, and TESTDB is the client database name, will have its corresponding queue database TERRY/OSE$TESTDB.db. It is easy to refer to those as data db and queue db.

For Java DB, the queue tables for each database are stored in a separate schema, "queue schema", in the user database. Each database with snapshot data will have its own corresponding queue schema with the same name prefixed by "OSE$" in the user database. For example, user database TERRY/TESTDB, where TERRY is the synchronization id, and TESTDB is the client database name, will have two schema, TERRY and OSE$TERRY.

The queue database/queue schema contains the following:

- OSE$TABLES - for queue-based mode is in queue db instead of data db.
- OSE$TRSEQ - both data db and queue db have it for using in the recovery mechanism.
- OSE$TRANS - table describing client and server transactions.
- OSE$DATAQ - table containing both outqueue and inqueue records for each transaction. Inqueue records are differentiated from outqueue records by having negative values in TRID column.
- OSE$BLOBQ - table containing BLOBs for both outqueue and inqueue records.

3.5 Manage Snapshots on the Mobile Client

The following are the types of snapshots you can enable for tracking the changes on the client database:

- *State-based*. State-based snapshots decipher the difference in the state of the data between subsequent synchronization events. This snapshot type is more resource efficient than queue-based snapshots. To enable state-based snapshots, set the `QUEUES` parameter in the `OSE.INI` file to `NO`.

  Snapshot state tables, `OSE_ST$<snapshot>`, are created in the client database and are populated by SQL triggers with primary keys of the modified rows.

- *Queue-based*: Both client and server changes are stored in a single queue. Whenever the snapshot is not locked by an application, the synchronization retrieves data from the In Queue and applies it to the base snapshot. At this point, the synchronization propagates data from the Out Queue to the server.

  Although both snapshot types rely on triggers, queue-based snapshots allow concurrent operations on the client database while any synchronization is in progress. The Sync Agent compose operation places modified data into the Out Queue. Later, the sync session uploads multiple client transactions delineated by a unique transaction id to the server.

To enable queue-based snapshots, set the `QUEUES` parameter in the `OSE.INI` file to `YES`. This is the default.

When you use queue-based snapshots, a queue database file is created, for Berkeley DB and SQLite, it is a standalone database named OSE$<client_database_name>.db, for Java DB, there is no separate database.

- Data queue for both In Queue and Out Queue records named `OSE$DATAQ`.
- BLOB queue named `OSE$BLOBQ`.
- Snapshot registry named `OSE$TABLES`.
- Transactions registry named `OSE$TRANS`.
- Transaction sequences per publication named `OSE$TRSEQ`.

The `OSE$DATAQ` queue is used for all snapshots and contains both In and Out Queue records. The TRID column is positive when the record is an Out Queue record. When you synchronize with queue-based snapshots enabled, new data from the client is uploaded from the `OSE$DATAQ` queue table and new data from the Oracle database is downloaded into this queue.

For more details on this parameter, see Appendix A.1.2.2, "QUEUES".

3.6 Control Automatic Synchronization for a Specific Mobile Client

As described in Section 5.5, "Using Automatic Synchronization" in the *Oracle Database Mobile Server Administration and Deployment Guide*, you can enable automatic synchronization for mobile clients either in the publication item or for the entire platform.

However, you can disable automatic synchronization for a single client by configuring the `DISABLE` parameter to `YES` in the `OSE.INI` file on the mobile client. This disables the Sync Agent and the only method for synchronization is a manual synchronization.

For more details on this parameter, see Appendix A.1.5, "Background Sync Parameter—BGSYNC".

3.7 Providing Security for the Mobile Client

The introduction of handheld devices within the corporate environment can pose a security threat to an organization. Devices are now used to store not only company contacts; but, with external cards, may store up to 60 gigabytes of information or more. Devices also provide a mobile point of entry into the organizational network that is located outside the network security perimeter. It is essential to secure this data if a device is lost or compromised.

Securing a device involves a layered approach. You must secure not only access to the device, but data stored on the device and communications across the network. Most aspects of security for a mobile device must be incorporated before Oracle Database Mobile Server is included within the security infrastructure.

1. Security starts with the device itself. Authentication on the device must be implemented through pin or password authentication, biometric readers, secure digital media for storage, and even how the device is stored, transported, and accounted for.
2. Once access is gained to the device, further security must be implemented within the mobile application to prevent the application from being able to retrieve invalid data. Technologies, such as the Microsoft.Net Compact Framework,

incorporate API calls that may be used to encrypt and decrypt any data that will be stored or retrieved from the device.

Oracle Database Mobile Server provides several security features that may be utilized to help in securing data. These features aid in protecting information during synchronization and once access to a device has been obtained. The two most important aspects of security for the mobile infrastructure are the following:

1. Use Secure Socket Layer (SSL) to protect the transmission of data during the synchronization process. For more information, see Section 9.4, "Configuring for Secure Socket Layer (SSL) Communication" in the *Oracle Database Mobile Server Administration and Deployment Guide*.
2. Encrypt the mobile client database, whether for the Berkeley DB or the SQLite database. For more information, see Section 3.7.1, "Encryption for the Client Databases".

3.7.1 Encryption for the Client Databases

For Berkeley DB, SQLite and Java DB, you can encrypt the data by using the encryption methods provided by the database respectively. To encrypt database created during synchronization, set OSE.ENCRYPTDB to YES in OSE.INI file. Refer to the *Development Guide* for more information. If OSE.ENCRYPTDB is specified to be YES, and no key is specified, then sync engine uses the password for synchronization as the key to encrypt the client database. This key must be added to the connection URL when accessing the client database via JDBC driver.

For details on encryption for these databases, see the following:

- SQLite provides a proprietary extension for encryption called SQLite Encryption Extension (SEE). For more information, see the following link:

  `http://www.hwaci.com/sw/sqlite/see.html`

- The Berkeley DB also supports the SQLite Encryption Extension (SEE) with some limitations. Berkeley DB encryption is discussed in the following documentation:

  `http://docs.oracle.com/cd/E17076_04/html/bdb-sql/sql_encryption.html`
- The Java DB encryption is discussed in the following documentation:

  `http://docs.oracle.com/javadb/10.10.1.2/devguide/cdevcsecure24366.html`

For Berkeley DB and SQLite databases, specify `password=<encryption key>` property in the JDBC connection URL to access an encrypted database. For Java DB database, specify `dataEncryption=true key>` and `dbootPassword=<encryption key>` properties in the JDBC connection URL to access an encrypted database.

3.8 Improve Performance by Disabling the Resume Feature

The resume feature manages intermittent network failures. If resume is enabled on both the server and the client, synchronization will resume automatically within the specified resume timeout period. Also, if sync session was interrupted during a network operation, the next synchronization will try to resume the operation, as long as resume is enabled and the resume timeout has not expired.

The resume transport adds overhead with additional network round trips and additional data to be saved on the client and on the server. Any device with reliable networks may disable the resume feature to improve performance of the synchronization system for this device and improve scalability on the server.

You can disable the resume feature for the mobile client by setting the `RESUME` parameter in the `OSE.INI` file to NO. For more details on the resume feature and disabling it for your mobile client, see Section A.1.1, "OSE Parameters - OSE" and Section 5.7, "Resuming an Interrupted Synchronization" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

3.9 Use the Device Manager Client GUI to Manage the Client-Side Device

On Win32, Windows Mobile, or Linux client platforms, you can manage the client software using the Device Manager. See Section 7.7, "Using the Device Manager Agent (dmagent) on the Client" in the *Oracle Database Mobile Server Administration and Deployment Guide* for more information.

3.10 Initiate Updates for the Mobile Client

You can initiate a request for software updates from the mobile client by executing the Oracle Database Mobile Server Update tool. For more information, see Section 7.6.3, "Initiate Updates of Oracle Database Mobile Server Software for Mobile Clients" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

Note: This function is not available to Blackberry, Java SE and Java ME clients since they do not have any Update tool.

3.11 Communicate Between the Internet and Intranet Through a Reverse Proxy

If a Win32, Windows Mobile or Linux mobile client is on either side of the firewall, set up a proxy or reverse proxy to facilitate communication between the mobile client and mobile server. For more information, see Section 9.6, "Using a Firewall Proxy or Reverse Proxy" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

4 Creating Sync Application for Android

The following sections use the `simple_sync_android` sample project to describe the steps to create an Android application that invokes Oracle Database Mobile Server sync APIs.

Note: This chapter assumes that you know how to use Eclipse to build an Android project and how to appropriately develop and sign an Android application.

- Section 4.1, "Prerequisites"
- Section 4.2, "Import the Oracle Database Mobile Server Android Project into Eclipse"
- Section 4.3, "Build Oracle Database Mobile Server Android Project"

4.1 Prerequisites

The following are the prerequisites for enabling synchronization for a SQLite or Berkeley DB Android application:

1. Install Eclipse IDE with the ADT plug-in, as detailed at the following site:

   `http://developer.android.com/sdk/eclipse-adt.html#installing`

2. Install the latest Android SDK, as detailed at the following site:

   `http://developer.android.com/sdk/index.html`

3. Install the Mobile Server Development Kit.

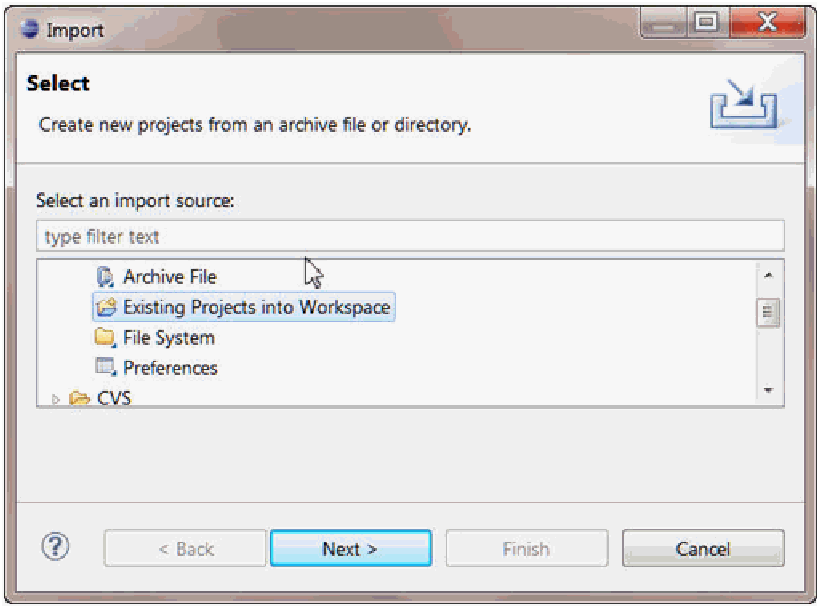
4.2 Import the Oracle Database Mobile Server Android Project into Eclipse

Import the Oracle Database Mobile Server `simple_sync_android` sample Android project into your Eclipse Workspace.

The following steps show how to import the mobile server sample Android project.

1. In Eclipse, with your Workspace open, select File->Import and choose "Existing Projects into Workspace". Click "Next".

Figure 4–1 Import Existing Projects into Eclipse Workspace

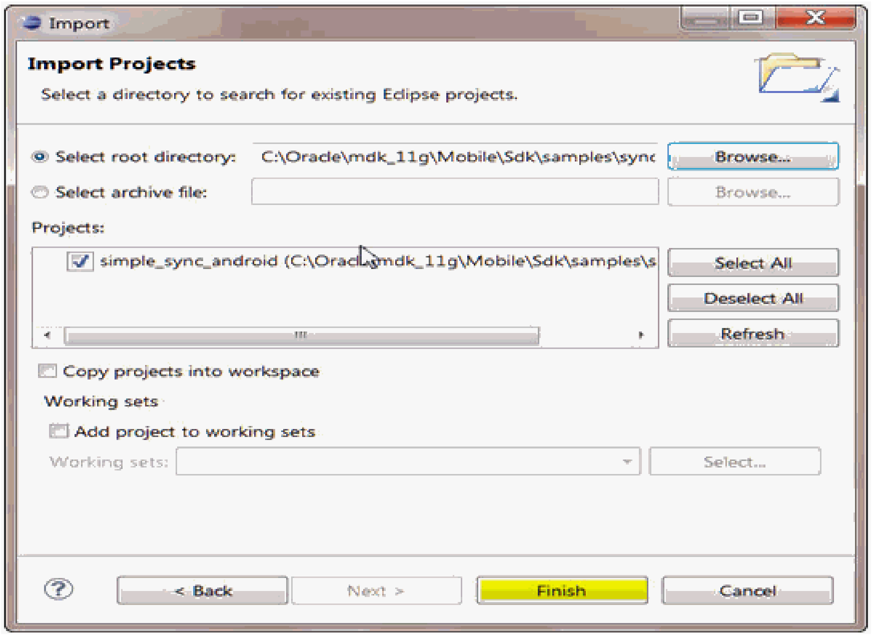


2. In the project location point to

   <MDK_ROOT>\Mobile\Sdk\samples\Sync\android\simple_sync_android

   Replace <MDK_ROOT> with the full path where Oracle Database Mobile Server Development Kit was installed. In Figure 4–2, for example, it is C:\oracle\mdk_11g

Figure 4–2 Select Root Directory for Eclipse Project



(●) Select root directory: C:\Oracle\mdk_11g\Mobile\Sdk\samples\sync
( ) Select archive file:

[✓] simple_sync_android (C:\Oracle\mdk_11g\Mobile\Sdk\samples\s

[ ] Copy projects into workspace

[ ] Add project to working sets

Working sets:

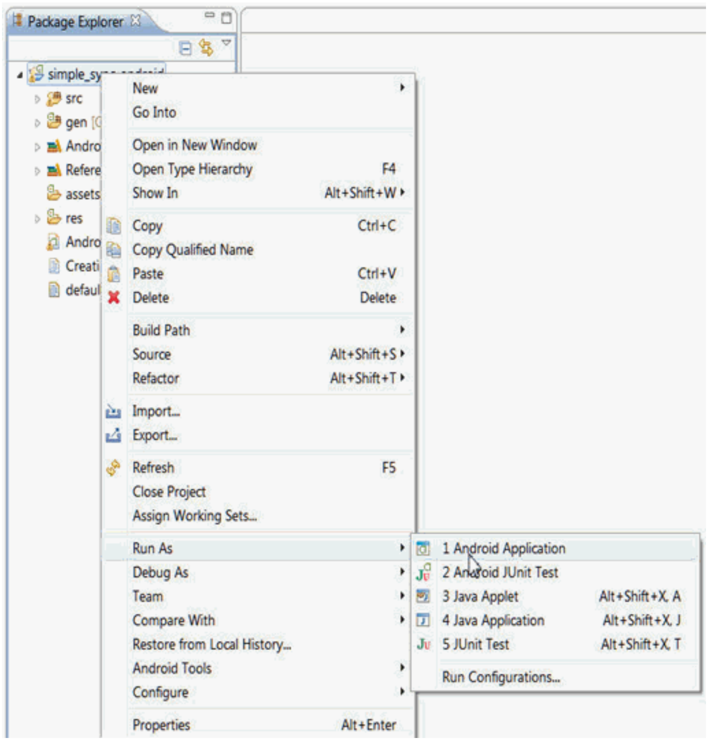
3. Click on "Browse…" button which should find the project files. Select the `simple_sync_android` project and click "Finish". The `simple_sync_android` project is now imported into your Eclipse Workspace.

4.3 Build Oracle Database Mobile Server Android Project

The following steps show how to build Oracle Database Mobile Server Android project:

1. Since the project references sync classes, in order to build it, you must copy the sync library file (s) to the libs subdirectory of the project.
2. For Berkeley DB client, copy the contents of <MDK_ROOT>\Mobile\Sdk\android\lib\bdb to <MDK_ROOT>\Mobile\Sdk\samples\Sync\android\simple_sync_android\libs. The libs directory should contain:
   - <MDK_ROOT>\Mobile\Sdk\samples\Sync\android\simple_sync_android\libs\osync_bdb_android.jar
   - <MDK_ROOT>\Mobile\Sdk\samples\Sync\android\simple_sync_android\libs\sqlite.jar
   - <MDK_ROOT>\Mobile\Sdk\samples\Sync\android\simple_sync_android\libs\armeabi\liboracle-jdbc.so
3. For SQLite client, copy osync_android.jar from <MDK_ROOT>\Mobile\Sdk\android\lib\ to <MDK_ROOT>\Mobile\Sdk\samples\Sync\android\simple_sync_android\libs\. The libs directory should contain: <MDK_ROOT>\Mobile\Sdk\samples\Sync\android\simple_sync_android\libs\osync_android.jar.
4. Build the project.
5. Run/Debug the simple_sync_android on a device emulator. After a successful build, to run the application, right click on the “simple_sync_android” project. Go to “Run As” item, and select “Android Application.” This will bring up the Android Emulator, where you should find “Mobile Sync” application.

Figure 4–3 "Run As" Eclipse Project Menu



6. Once the Android emulator is loaded, the mSync application will be started. To sync with a Mobile Server, provide a sync client's authentication information, the Mobile Server's http url (for example, myhost: 8090) and select the "Sync" button.

   Several menu options are available when you select the "Device Menu" button. For more information, see Table 4–1.

Figure 4–4 Synchronization UI

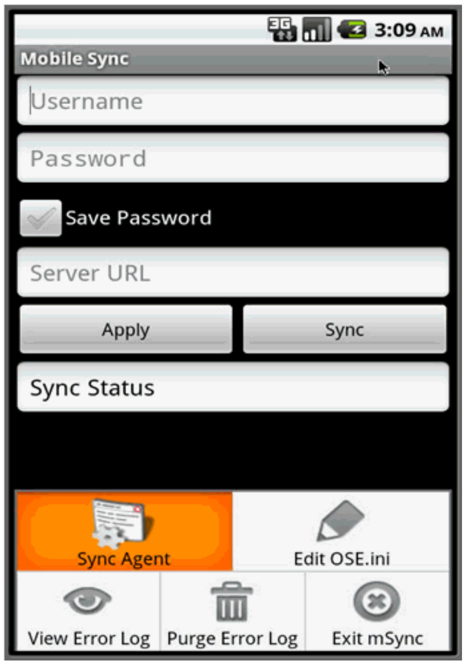


Table 4–1 Menu Options for "Device Menu" Button

Button	Functions
Sync Agent	Brings up the automatic Sync Agent screen
Edit OSE.ini	Allows users to modify contents of parameter file, OSE.ini
View Error Log	View the contents of the sync error log
Purge Error Log	Remove all contents from the sync error log
Exit mSync	Exit the application and terminate the mSync process. Selecting the Android device's "Back" button puts the sync process in the background.

7. Examine the data after sync.

   a. Use the Android SDK tool **adb shell** (located at <Android SDK root>\android-sdk\platform-tools), to connect to the running instance of a device or an emulator.

   b. For SQLite client, use the **sqlite3** tool, included with Android SDK, to connect to the synchronized database(s). For more information on sqlite3 utility, see Android documentation. The simple sync client, by default, creates databases under:

      /data/data/tests.sync/app_oracle.sync/sqlite_db/<SYNC USERNAME>

      For example, if the user name is S11U1 and the user is subscribed to "SAMPLE11" publication, to connect to the database sample11.db and list all of its tables, the following commands are required: adb shell, cd /data/data/tests.sync/app_oracle.sync/sqlite_db/S11U1, sqlite3 sample11.db, and .tables

   c. For Berkeley DB client, you can use **dbsql** utility for Android platform to examine the synchronization data. The dbsql utility is located at <MDK_ROOT>\Mobile\Sdk\android\tools\bdb. You can use the Android SDK tool adb push (located at <Android SDK root>\android-sdk\platform-tools), to copy it into the running instance of a device or an emulator. For more information on dbsql utility, see Berkeley DB documentation. The simple sync client, by default, creates databases under:

      /data/data/tests.sync/app_oracle.sync/bdb/data/<SYNC USERNAME>

   d. Examine MainAct.java

      The sample consists of the UI layout code (found in the res\layout subdirectory) and the main application Java class file: MainAct.java. All the logic of initializing sync structures and calling the sync functions is located here. Examine this file for detailed explanation of the required sequence of calls to the sync API.

5 Synchronization Utilities for Java Sync Client

The following sections provide information on synchronization utilities for java sync client:

- Section 5.1, "OSync Utility"
- Section 5.2, "SetParam Utility"
- Section 5.3, "SyncConsole Utility"

5.1 OSync Utility

The OSync utility is used to do synchronization for Java Sync client. The usage is given as follows:

Table 5–1 Usage of OSync Utility

Parameter	Description
user, pwd, url	Username, password, server URL
-f	Use files to store temporary sync payload
-ns	Do not save sync metadata files before sync
-sp	Save sync password in encrypted form in sync metadata files
-r	Enable resume
-ssl	Use SSL over HTTP
-sel pub	Sync with selective publication
-nnp	Do not download new publications
-fr	Force complete refresh
-np password	New password
-hp	High priority
-so	Send only one-way sync
-cancel stage percent	Sync and cancel at stage and percent, specify stage and percent with numbers, valid numbers for stage are 1 - 4, valid numbers for percent are 1 - 99.
-param par1=val1 par2=val2	Set parameters in ose.ini before sync, refer to oracle.opensync.tools.SetParam for valid parameters in ose.ini.

The OSync utility is supported for both the Java SE client and the Java ME (OJEC) client. For Java SE Berkeley DB and SQLite clients, OSync will do synchronization with the Ch-Werner JDBC Driver by default. However, you can specify which JDBC driver to use during synchronization by using the OSync command below. For Java SE Java DB client the Apache Derby Embedded JDBC driver is used and you cannot specify an alternate. For Java ME (OJEC) client the Ch-Werner JDBC driver is used and you cannot specify an alternate.

The following command is used to sync with the Berkeley DB database and Ch-Werner JDBC driver:

```
java -Djava.library.path=<path to BDB native and jdbc jni libraries> -cp .:osync_
se_bdb.jar:jdbc.jar oracle.opensync.tools.OSync <USER_NAME> <PASSWORD> <SERVER_
URL: MobileServerIp:MobileServerPort> -param OSE.FILES=YES
```

The following command is used to sync with the SQLite database and Zentus JDBC driver:

```
java -cp .:osync_se_sqlite.jar:sqlitejdbc-v053.jar oracle.opensync.tools.OSync
<USER_NAME> <PASSWORD> <SERVER_URL: MobileServerIp:MobileServerPort> -param
OSE.FILES=YES -param JDBC.DRIVER=org.sqlite.JDBC -param JDBC.URL_PFX=jdbc:sqlite:
```

The following command is used to sync with the Apache Derby Embedded JDBC Driver and Java DB database:

```
java -cp .:osync_se_javadb.jar:derby.jar oracle.opensync.tools.OSync <USER_NAME>
<PASSWORD> <SERVER_URL: e.g. MobileServerIp:MobileServerPort> -param OSE.FILES=YES
```

Another method to specify which JDBC driver to use during synchronization is to create an ose.ini file manually in the same directory as Java sync engine archive file and add the parameters below to it before you invoke OSync utility.

To sync with Ch-Werner JDBC driver, the corresponding parameters should be:

- JDBC.DRIVER=SQLite.JDBCDriver
- JDBC.URL_PFX=jdbc:sqlite:/

To sync with Zentus JDBC driver, the corresponding parameters should be:

- JDBC.DRIVER=org.sqlite.JDBC
- JDBC.URL_PFX=jdbc:sqlite:

Note: Parameters JDBC.DRIVER and JDBC.URL_PFX are only supported by Java SE Berkeley DB and Java SE SQLite clients.

The difference between the ME client and SE client is that you must specify parameter OSE.FILES=YES for the ME client to do the synchronization, either in the OSync command line, or in the ose.ini configuration file.

If your Mobile Server URL starts with "https", you must specify parameter NETWORK.DISABLE_SSL_CHECK = YES, either in the OSync command line, or in the ose.ini configuration file.

For more information, refer to javadoc of class oracle.opensync.tools.OSync.

5.2 SetParam Utility

SetParam utility is used to create and edit ose.ini, which is in the same directory as Java sync engine archive file. The supported parameters in ose.ini are given in the following table:

Table 5–2 Usage of SetParam Utility

Parameter	Description
BDB.LARGE_RECORD_OPT=<value>	Any record larger than <value> will be stored in a new format that improves read and write performance for large records.
BDB.STREAMING=YES|NO	Set this parameter to YES to use the Ch-Werner Java wrapper with the Pure Java SE clients, OJEC client, or Berkeley DB Android client. Set this parameter to NO to use the Ch-Werner JDBC driver with the Pure Java SE clients, OJEC client, or Berkeley DB Android client.
BGSYNC.DISABLE=YES|NO	Enable/Disable Sync Agent
OSE.RESUME=YES|NO	Use resume or not
OSE.FILES=YES|NO	Use files or not to store temporary sync payload
OSE.ENCRYPTDB=YES|NO	Encrypt the client database (YES), otherwise do not encrypt it (NO). The default is NO. This applies to native and Pure Java clients using Berkeley DB.
SQLITE.LIMIT_CONNECTIONS=YES|NO	Limit number of database connections during sync to 2 (queue-based mode) or 1 (state-based mode). Currently required for Blackberry client where OS limits number of database connections.
SQLITE.QUEUES=YES|NO	Use queue-based mode (YES) or state-based mode (NO) for Berkeley DB and SQLite.
SQLITE.DATA_DIRECTORY=<path name>	Root of the client database directory for Berkeley DB and SQLite (for example, <mobileclient>/bdb/data for Berkeley DB, or <mobileclient>/sqlite_db for SQLite)
JDBC.DRIVER=<JDBC driver>	JDBC driver class name for Berkeley DB and SQLite (for example, org.sqlite.JDBC for Zentus JDBC driver, and SQLite.JDBCDriver for Ch-Werner JDBC driver)
JDBC.URL_PFX=<JDBC URL prefix>	JDBC URL prefix for Berkeley DB and SQLite (for example, jdbc:sqlite: for Zentus JDBC driver, and jdbc:sqlite:/ for Ch-Werner JDBC driver)
JAVADB.DATA_DIRECTORY=<path name>	Root of the Java DB client database directory. Default is <mobileclient>/javadb_home.
JAVADB.QUEUES=YES|NO	Use queue-based mode (YES) or state-based mode (NO) for Java DB.
JAVADB.SHUTDOWN_ON_CLOSE=YES|NO	Whether or not to shut down the Java DB engine when sync engine is shut down. Default value is NO.
BGSYNC.NET_WAIT_TIMEOUT=<integer value>	Time interval in ms to wait to check network status in syncagent in absence of network notifications. Default is 600000 ms or 10 minutes.
NETWORK.DISABLE_SSL_CHECK=YES|NO	Disable SSL Certificate check on the client. Needed if server uses self-signed certificates.

Usage:

```
java -cp .:<Java SE archive> oracle.opensync.tools.SetParam parameter1=<value1>
parameter2=<value2>
```

Example:

```
java -cp .:osync_se_bdb.jar oracle.opensync.tools.SetParam OSE.FILES=NO
java -cp .:osync_se_sqlite.jar oracle.opensync.tools.SetParam OSE.FILES=NO
java -cp .:osync_se_javadb.jar oracle.opensync.tools.SetParam OSE.FILES=NO
```

You can invoke the utility in command line or call the included class methods in oracle.opensync.tools.SetParam from your own classes to programmatically set the parameters in ose.ini before the call to sync.

For more information, refer to javadoc of class oracle.opensync.tools.SetParam.

5.3 SyncConsole Utility

You can use SyncConsole utility to invoke a synchronization, set parameters in ose.ini and control syncagent for Java Sync client.

Usage:

- To start it with Berkeley DB database and Ch-Werner JDBC driver:

```
java -Djava.library.path=<path to BDB native and jdbc jni libraries> -cp
.:osync_se_bdb.jar:jdbc.jar oracle.opensync.tools.SyncConsole
```

- To start it with SQLite database and Zentus JDBC driver:

```
java -cp .:osync_se_sqlite.jar:sqlitejdbc-v053.jar
oracle.opensync.tools.SyncConsole
```

- To start it with Java DB database and Apache Derby Embedded JDBC driver:

```
java -cp .:osync_se_javadb.jar:derby.jar oracle.opensync.tools.SyncConsole
```

You can input commands once you start SyncConsole. The supported commands are as follows:

Table 5–3 Usage of SyncConsole Utility

Command	Description
sync	Invokes oracle.opensync.tools.OSync to do a synchronization. You can refer to Section 5.1, "OSync Utility" for valid parameters for OSync.
status	Reports sync agent status.
start	Start Syncagent. The command will wait for syncagent to be started unless -nw or -nowait option is specified.
stop	Stop Syncagent. The command will wait for syncagent to be stopped unless -nw or -nowait option is specified.
pause	Pause Syncagent. The command will wait for syncagent to be paused unless -nw or -nowait option is specified.
resume	Resume Syncagent. The command will wait for syncagent to be resumed unless -nw or -nowait option is specified.
enable	Enables Sync Agent
disable	Disables Sync Agent. It would stop Sync Agent first and then Disable it. If you specify -nw or -nowait, it will disable immediately.
message	Invoking this command first time will add message handler to syncagent to dump syncagent messages into file named "SyncConsoleMsg.txt" in the working directory. Invoking this command second time will remove this message handler so that writing of syncagent messages will be stopped.
msg	The same as command "message".
setparam	Invokes oracle.opensync.tools.SetParam to set parameters in ose.ini. You can refer to Section 5.2, "SetParam Utility" for valid parameters for SetParam.

Table 5–3 (Cont.) Usage of SyncConsole Utility

Command	Description
q	Exit SyncConsole process.
exit	The same as command "q".
quit	The same as command "q".

For example:

```
Sync Console>sync john john server_url
```

6

Creating Sync Application for iOS

The following sections describe how to use mobile client sync package for iOS to develop mobile sync application for Sqlite or Berkeley DB:

- Section 6.1, "Contents of the iOS Mobile Client Package"
- Section 6.2, "Building Sync Application"

You need to download and install the mobile sync package on your Mac from the mobile server in order to develop the application.

Refer to Section 2.3.7, "Installing iOS Mobile Client" for requirements and instructions.

6.1 Contents of the iOS Mobile Client Package

Here is the directory tree for Berkeley DB sync package:

osync-bdb-ios-12.1

- **doc**
  - *mcguide.pdf*
- **include**
  - *bgmsg.h*
  - *bgsync.h*
  - *ose.h*
  - *oseerr.h*
  - *sqlitePluginErr.h*
- **lib**
  - **ios6.0**
    - **armv7**
      - *libdb_sql-6.1.a*
      - *libosync_bdb.a*
  - **ios6.0simulator**
    - **i386**
      - *libdb_sql-6.1.a*
      - *libosync_bdb.a*
- **res**

osync_res

del.proj

osync.strings

en.lproj

osync.strings

es.lproj

osync.strings

fr.lproj

osync.strings

it.lproj

osync.strings

ja.lproj

osync.strings

ko.lproj

osync.strings

pt-BR.lproj

osync.strings

zh-Hans.lproj

osync.strings

The structure of Sqlite sync package is almost identical, except that it does not contain Berkeley DB library libdb_sql-6.1.a (instead the Sqlite library provided by the system is used):

osync-ios-12.1

doc

mcguide.pdf

include

bgmsg.h

bgsync.h

ose.h

oseerr.h

sqlitePluginErr.h

lib

ios6.0

armv7

libosync.a

ios6.0simulator

i386

libosync.a

res

- **osync_res**
  - **de.lproj**
    - *osync.strings*
  - **en.lproj**
    - *osync.strings*
  - **es.lproj**
    - *osync.strings*
  - **fr.lproj**
    - *osync.strings*
  - **it.lproj**
    - *osync.strings*
  - **ja.lproj**
    - *osync.strings*
  - **ko.lproj**
    - *osync.strings*
  - **pt-BR.lproj**
    - *osync.strings*
  - **zh-Hans.lproj**
    - *osync.strings*

Note: The contents are subject to change when new versions/platforms are supported.

The following sections provide information on the top-level directories in the package:

- Section 6.1.1, "doc Directory"
- Section 6.1.2, "include Directory"
- Section 6.1.3, "lib Directory"
- Section 6.1.4, "res Directory"

6.1.1 doc Directory

Contains this document and the related files.

6.1.2 include Directory

Contains sync header files. Your application needs to include these files for using the sync APIs:

- **ose.h** contains foreground sync APIs and related constants and structures.
- **bgsync.h** contains background sync APIs and related constants and structures.
- **oseerr.h** contains foreground sync error code constants.

- **sqlitePluginErr.h** contains SQLite and Berkeley DB plugin error code constants that provide more information about database-related sync errors.
- **bgmsg.h** contains background sync error code constants and background sync message and other related constants.

6.1.3 lib Directory

Contains sync and Berkeley DB static libraries (for Berkeley DB client). Your application needs to link with libosync.a and libosync_bdb.a to use the sync APIs for Sqlite and Berkeley DB clients respectively. For Berkeley DB client, your application also needs to link with Berkeley DB library libdb_sql-6.1.a to access Berkeley DB client databases.

The contents are arranged by platform and CPU type. Both device and simulator builds are provided.

Currently iOS 6.0 and above targets are supported built for armv7 architecture.

6.1.4 res Directory

Contains a bundle directory osync_res that contains resources used by sync client. Currently osync_res contains only string files used for sync client error messages and background sync messages.

However, more resources may be added in future to support UI components. Internationalization is currently provided for 9 languages: English, Spanish, French, German, Italian, Korean, Japanese, Portuguese(Brazilian) and Chinese Simplified.

6.2 Building Sync Application

The following sections provide information on how to build sync application:

- Section 6.2.1, "Prerequisites"
- Section 6.2.2, "Build Settings"

6.2.1 Prerequisites

This section assumes that you are already familiar with how to create iOS applications and hence describes how to include sync functionality into your iOS application project.

Note: For information on Mobile Server and Sync concepts, refer to Chapter 2, "*Synchronization*" of the "*Mobile Server Developer's Guide*".

Note: iOS client supports only native C APIs. Refer to Section 3.1.1.2, "*OSE Synchronization APIs For Native Applications*" of the *"Mobile Server Developer's Guide* " for OSE Synchronization APIs (foreground sync) and Section 3.2.1.2, "*Native APIs for the Sync Agent and Automatic Synchronization*" of the *"Mobile Server Developer's Guide"* for Automatic Synchronization APIs.

You need the following installed on your Mac:

- OS X v10.7 Lion or later version, currently OS X v.10.8 Mountain Lion.

- Xcode 4.5 or later version, currently 4.6. Xcode already includes iOS SDKs. To develop for iOS 6.1 and later Xcode 4.6 is required (it includes iOS 6.1 SDK).
- Mobile Sync package described in Section 6.1, "Contents of the iOS Mobile Client Package".

6.2.2 Build Settings

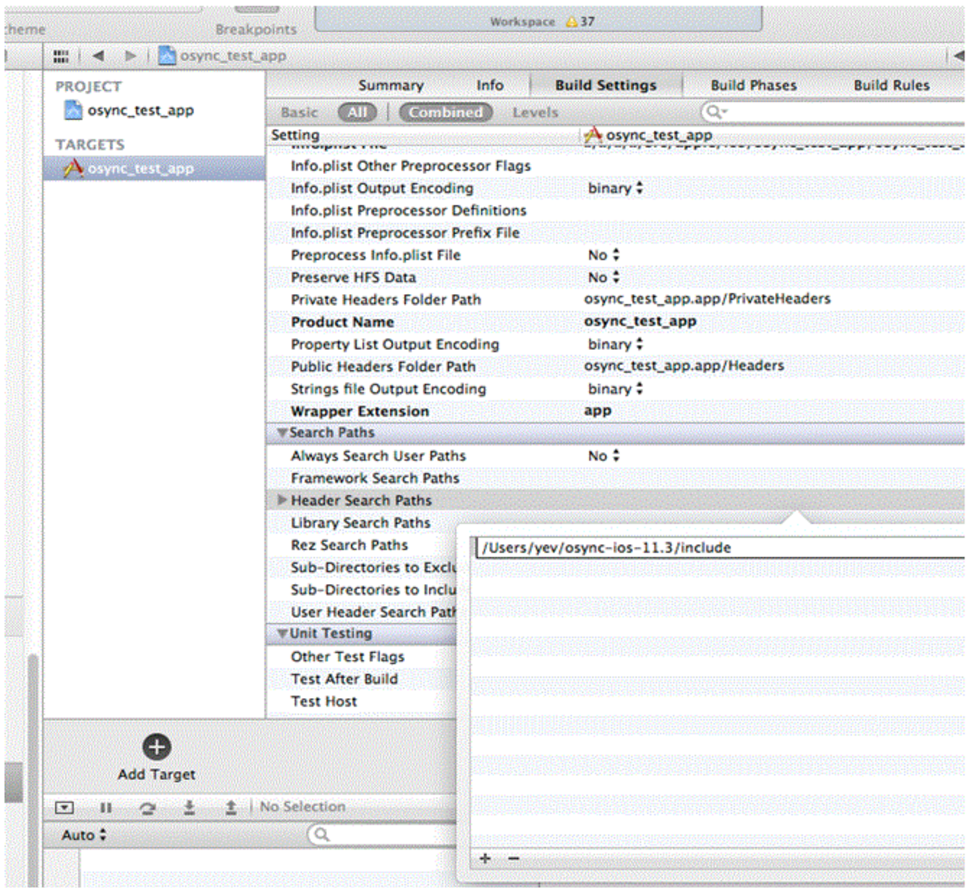
This section assumes that you are already familiar with how to create iOS applications and hence describes how to include sync functionality into your iOS application project. For more information see the following section:

- Section 6.2.2.1, "Header Search Path"

6.2.2.1 Header Search Path

- Click on your project, then on your application target, then on "Build Setting" tab.
- Scroll down until you see "Search Paths" section and double-click on a line of "Header Search Paths".
- Click "+" in the bottom left corner of the window pop up.
- Add "include" directory from the sync package to the header search path (either via absolute path or relative path from your project directory).

Figure 6–1 Add "include" Directory to Header Search Path



```
Setting                                 osync_test_app
Info.plist Other Preprocessor Flags
Info.plist Output Encoding              binary
Info.plist Preprocessor Definitions
Info.plist Preprocessor Prefix File
Preprocess Info.plist File              No
Preserve HFS Data                       No
Private Headers Folder Path             osync_test_app.app/PrivateHeaders
Product Name                            osync_test_app
Property List Output Encoding           binary
Public Headers Folder Path              osync_test_app.app/Headers
Strings file Output Encoding            binary
Wrapper Extension                       app
▼Search Paths
Always Search User Paths                No
Framework Search Paths
▶Header Search Paths
Library Search Paths
Rez Search Paths                        /Users/yev/osync-ios-11.3/include
Sub-Directories to Exclu
Sub-Directories to Inclu
User Header Search Path
▼Unit Testing
Other Test Flags
Test After Build
Test Host
```

Note: To use Sqlite C APIs to access your sync application databases, you also need to include "sqlite3.h" header file into your source files.

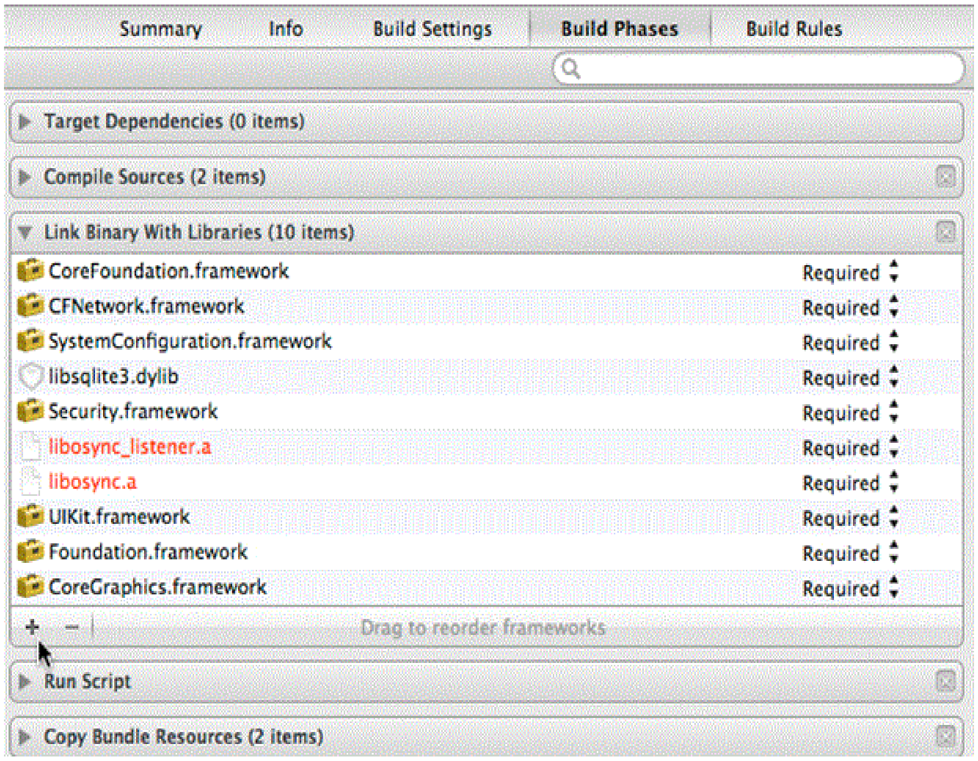
This file is already provided by iOS SDK so no additional include path is necessary. Berkeley DB fully supports Sqlite C APIs, so you can use "sqlite3.h" for Berkeley DB clients as well.

Refer to Berkeley DB documentation for more details.

6.2.3 Link With Libraries

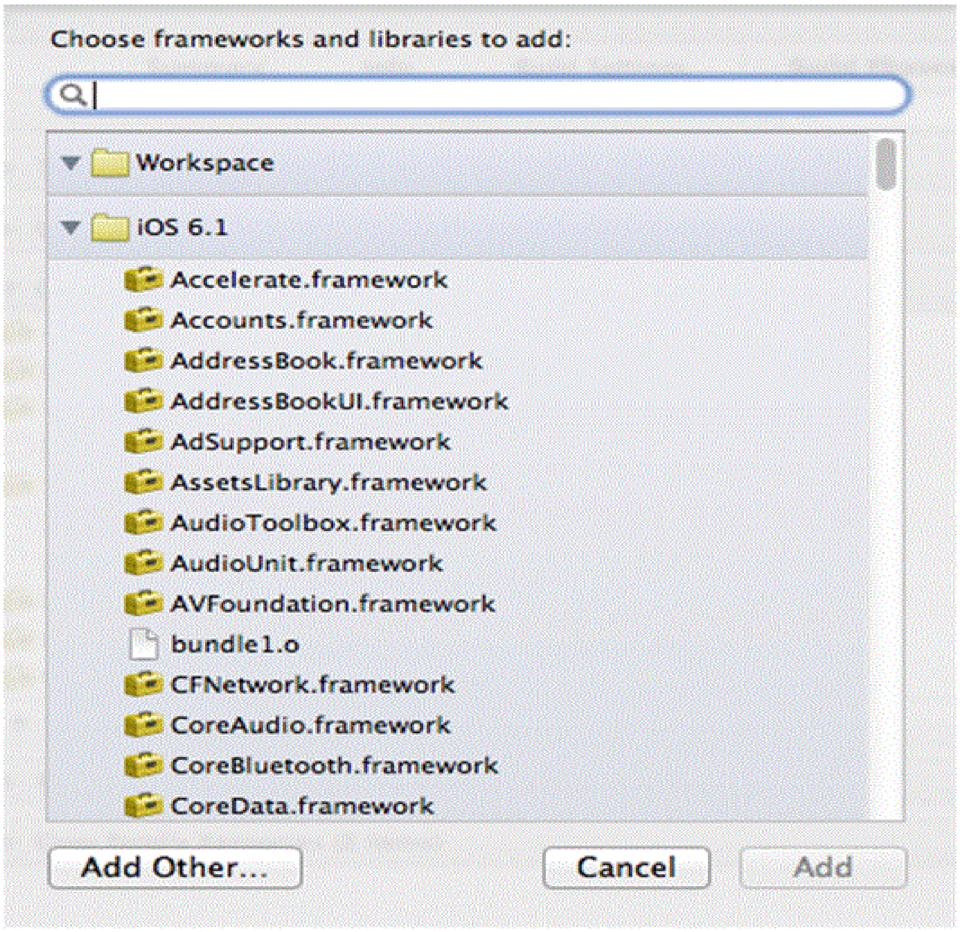
Click "Build Phases" tab (on the same screen), and toggle "Link Binary With Libraries". Click on "+".

Figure 6–2 Link With Libraries


Summary
Info
Build Settings
Build Phases
Build Rules
Target Dependencies (0 items)
Compile Sources (2 items)
Link Binary With Libraries (10 items)
CoreFoundation.framework Required
CFNetwork.framework Required
SystemConfiguration.framework Required
libsqlite3.dylib Required
Security.framework Required
libosync_listener.a Required
libosync.a Required
UIKit.framework Required
Foundation.framework Required
CoreGraphics.framework Required
Drag to reorder frameworks
Run Script
Copy Bundle Resources (2 items)

You will see the following window:

Figure 6–3 Add Frameworks and Libraries



See the sections below for more information:

- Section 6.2.3.1, "Frameworks"
- Section 6.2.3.2, "Libraries From The Sync Package"

6.2.3.1 Frameworks

The following frameworks are needed for mobile sync:

- CoreFoundation.Framework
- CFNetwork.Framework
- SystemConfiguration.Framework
- UIKit.Framework
- Foundation.Framework
- Libsqlite3.dylib - only for Sqlite clients

Note: Berkeley DB clients will use Berkeley DB libraries included in the sync package.

Add these frameworks using the procedure outlined above. You may later move the frameworks under "Frameworks" group in your project

6.2.3.2 Libraries From The Sync Package

If you are only building for one architecture (either device only or simulator only), it is simple to add the sync package libraries.

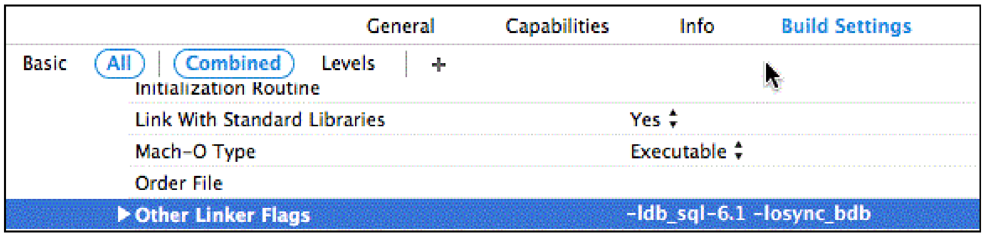
On the window from Figure 6–3, click on "Add Other". Navigate to the sync package directory, lib subdirectory and then appropriate subdirectories for your platform and CPU type (currently this will only depend on whether you are performing device or simulator build).

You need to add libosync.a and libosync_bdb.a for Sqlite and Berkeley DB clients respectively. For Berkeley DB clients only, you also need to add libdb_sql-6.1.a.

On the other hand, if you want to have both simulator and device architectures you may have to follow different steps:

1. Navigate to the "Linking" section, within the "Build Settings" tab (Figure 6–1) and find "Other Linker Flags" row.
2. Click to the right of this row and add linking to these libraries explicitely:
   - -losync for SQLite clients
   - -losync_bdb -ldb_sql-6.1 for Berkeley DB clients

Figure 6–4 Explicit Linking



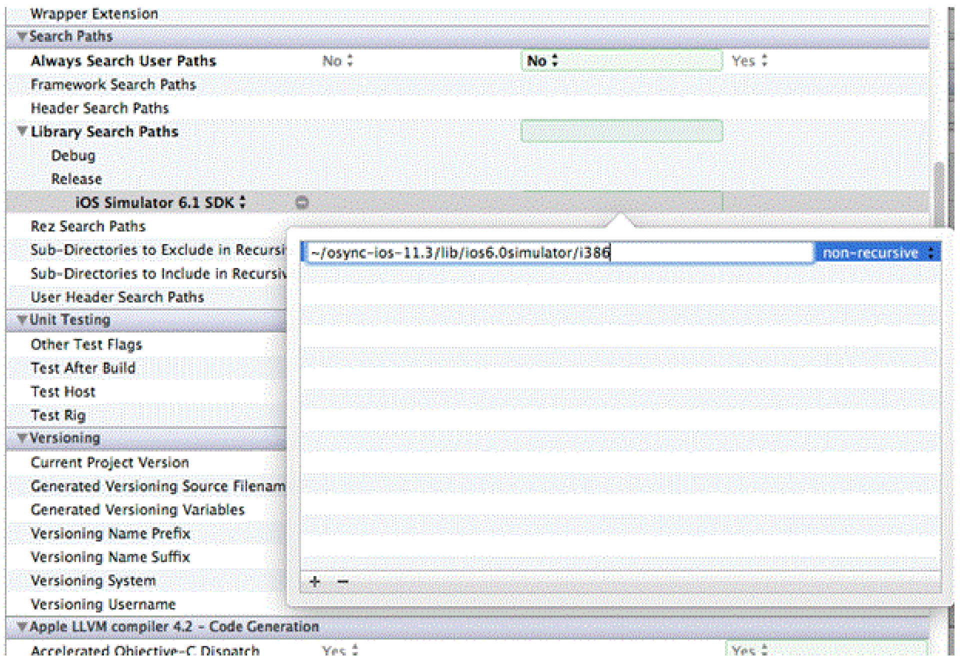
Link With Standard Libraries: Yes
Mach-O Type: Executable
Order File
Other Linker Flags: -ldb_sql-6.1 -losync_bdb

3. The previous step only specified the library names but not their locations. You can customize the search path for different architectures following the steps below:
   - In the same Build Settings" tab, find "Search Paths" section and "Library Search Paths" row. Toggle it.
   - For each build configuration, "Debug" and "Release", you can customize the library search path depending on SDK and architecture. Click "+" next to configuration. You will see "Any Architecture | Any SDK" click on it to pop up the menu.
   - To have both device and simulator builds, it would be enough to customize based on SDK ("iOS 6.1 SDK" or "iOS Simulator 6.1 SDK").
   - Double-click to the right of the selected item. A window will pop-up, click on "+" at the bottom left corner.
   - Enter appropriate path (for each SDK) within the lib directory of the sync package. See Figure 6–5 below.

   Note: You can separately customize Debug and Release configurations.

   Now your project will link with correct libraries from the sync package depending on the configuration, architecture and SDK.

Figure 6–5 Customizing Library Search Paths



```
Wrapper Extension
▼Search Paths
Always Search User Paths              No ⬍        No ⬍        Yes ⬍
Framework Search Paths
Header Search Paths
▼Library Search Paths
   Debug
   Release
      iOS Simulator 6.1 SDK ⬍
Rez Search Paths
Sub-Directories to Exclude in Recursi
Sub-Directories to Include in Recursiv
User Header Search Paths
▼Unit Testing
Other Test Flags
Test After Build
Test Host
Test Rig
▼Versioning
Current Project Version
Generated Versioning Source Filenam
Generated Versioning Variables
Versioning Name Prefix
Versioning Name Suffix
Versioning System
Versioning Username
▼Apple LLVM compiler 4.2 - Code Generation
Accelerated Objective-C Dispatch      Yes ⬍                   Yes ⬍

~/osync-ios-11.3/lib/ios6.0simulator/i386     non-recursive ⬍
+ −
```

6.2.4 Include Sync Package Resources

You must include resource from the sync package. Currently they only contain localized string files for error and other messages. More will be added in the future. Follow the steps below to include sync package resource:

1. Right-click on your project and select "Add Files To [Your Project Name]". Navigate to "res" directory in the sync package, selecr "osync_res" subdirectory and add it to the project.

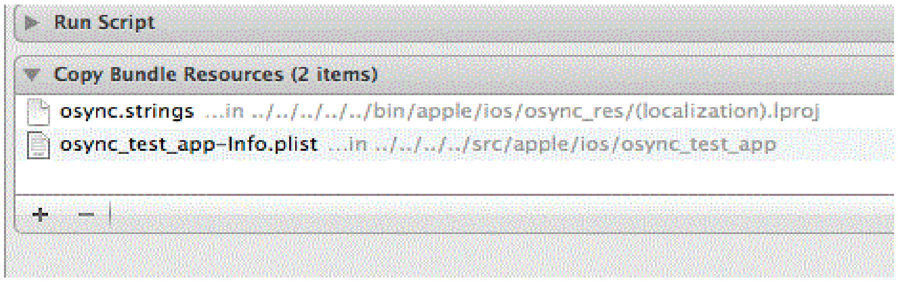
   Alternatively you can navigate to "res" directory in the Finder and drag "osync_res" directory to your project.

   Note: You can choose "Copy Files to Destination Group's Folder" option but it is not required.

   ---

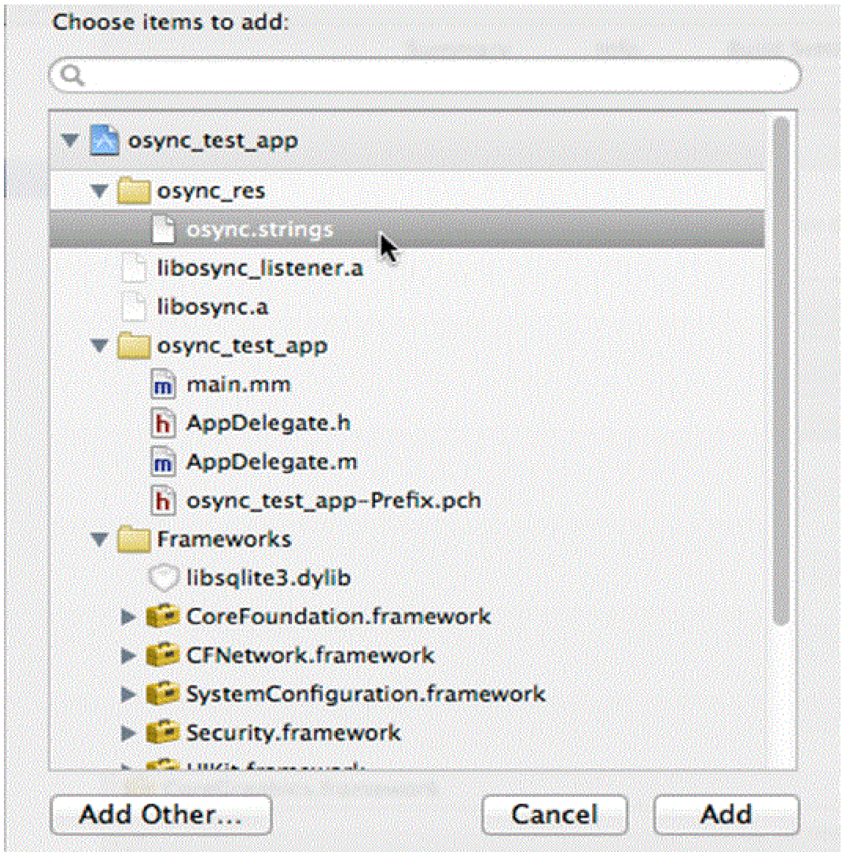
2. Xcode should automatically add the string files within "osync_res" to "Copy Bundle Resources" build phase. Verify this by going to the "Build Phases" tab and toggling "Copy Bundle Resources" section. You should see localized "osync.strings" file added.

Figure 6–6 osync.strings in "Copy Bundle Resources" Phase


Run Script
Copy Bundle Resources (2 items)
osync.strings ...in ../../../../../bin/apple/ios/osync_res/(localization).lproj
osync_test_app-Info.plist ...in ../../../../src/apple/ios/osync_test_app
+ −

3. In case you don't see "osync.strings" file added, you can add it manually by clicking "**+**" from Figure 6–6. On the screen that pops up, you should see "osync_res" folder under your project. Select "osync.strings" file in it. See Figure 6–7.

Figure 6–7 Adding "osync.strings" Manually



This concludes the steps needed to build Sqlite and Berkeley DB sync applications.

A Mobile Client Configuration Parameters

You can customize the mobile client by modifying the parameter values defined in the OSE.INI configuration file.

The installation automatically sets the parameters in the OSE.INI file, but you can modify them to customize the product behavior. To modify these files, use an ASCII text editor. You must have write permissions on the directory where either file is located to be able to modify them.

Note: On the Windows Mobile and Blackberry platforms, these files are named with the extension of .TXT, so that you can double-click on it to open the file.

The following sections detail the parameters within the OSE.INI and DEVMGR.INI configuration files:

- Section A.1, "OSE.INI File Overview"
- Section A.2, "DEVMGR.INI File"
- Section A.3, "Sample OSE.INI and DEVMGR.INI Files"

A.1 OSE.INI File Overview

The OSE.INI file stores properties used by the mobile clients. It contains parameters that define the location of the mobile client database and mobile client files, defines parameters for all databases on a system, and how to customize synchronization for the mobile client database. There is a single OSE.INI file for each mobile device for all users of that device. On Windows, Linux and Windows Mobile platforms, the latest modifications to parameters in the OSE.ini file take effect only after restarting the mSync, Sync Agent and dmagent programs. On Pure Java Client supported platforms, the latest modifications to parameters in the OSE.ini file take effect only after restarting the application embedding the Sync engine.

Note: The installation automatically sets the parameters in your OSE.INI file, but you can customize the mobile client by modifying the parameter values defined in your OSE.INI file. You must have write permissions on the directory where this file is located to be able to modify the OSE.INI file. To modify the OSE.INI file, use an ASCII text editor.

Depending on the platform, the OSE.INI file can be located in one of the following directories on the mobile device:

- On Win32, Windows Mobile and Linux platforms, the `OSE.INI` file is located in the `<`*`mobile_client_install_root`*`>\bin` directory.
- On Windows Mobile, the `OSE.TXT` file is located in the `<`*`mobile_client_install_root`*`>\bin` directory.
- On Blackberry, the `OSE.TXT` file is located in the `/store/home/user/oracle/sync` directory.
- On Android, the `OSE.INI` file is located in `/data/data/<`*`application_package`*`>/app_oracle.sync`. Applications import the `osync_android.jar` library; thus, the `<`*`application_package`*`>` should be replaced with the user's application that invokes the `OSESession` APIs.
- On Java SE and Java ME platforms, the `OSE.INI` file is located in `<mobileclient>`, where you install and place the Java SE archive file (e.g. osync_se_bdb.jar, osync_se_sqlite.jar, osync_se_javadb.jar andosync_me.jar).
- On iOS, ose.ini file is located within the application sandbox in the mobile client directory which is "Library/Application Support/oracle" (relative to the sandbox root).

Note: The user cannot modify files within application sandbox, so the only way to access and modify ose.ini file for iOS client is by using OSE APIs oseSetParam() and oseGetParamNC() within your application. For more information, see section 3.1.1.2.14 of the *Mobile Server Develoeper's Guide*.

The following are the parameter sections for the `OSE.INI` file:

- Section A.1.1, "OSE Parameters - OSE"
- Section A.1.2, "SQLite Mobile Client Parameters—SQLITE"
- Section A.1.3, "Java DB Mobile Client Parameters — JAVADB"
- Section A.1.4, "Berkeley DB Mobile Client Parameters—BDB"
- Section A.1.5, "Background Sync Parameter—BGSYNC"
- Section A.1.6, "Network Parameters - NETWORK"

A.1.1 OSE Parameters - OSE

The following are the OSE parameters:

- Section A.1.1.1, "Resume Parameter - RESUME"
- Section A.1.1.2, "Files Parameter - FILES"
- Section A.1.1.3, "Encryptdb Parameter - ENCRYPTDB"

A.1.1.1 Resume Parameter - RESUME

The RESUME parameter specifies whether the resume transport is enabled.

Values are YES, TRUE, NO or FALSE.

Syntax

```
OSE.RESUME=YES|NO|TRUE|FALSE
```

A.1.1.2 Files Parameter - FILES

The FILES parameter specifies whether files are used to temporarily store the data before it's sent/after it's received. Values are YES, TRUE, NO or

FALSE.

Syntax

```
OSE.FILES=YES|NO|TRUE|FALSE
```

A.1.1.3 Encryptdb Parameter - ENCRYPTDB

The ENCRYPTDB parameter specifies whether databases newly created during sync should be encrypted. Values are YES, TRUE, NO or FALSE.

Syntax

```
OSE.ENCRYPTDB=YES|NO|TRUE|FALSE
```

A.1.2 SQLite Mobile Client Parameters—SQLITE

The `SQLITE` section configures certain aspects of both the Berkeley DB and SQLite Mobile Clients. The following sections describe the mobile client parameters that you can modify:

- Section A.1.2.1, "DATA_DIRECTORY"
- Section A.1.2.2, "QUEUES"
- Section A.1.2.3, "LIMIT_CONNECTIONS"
- Section A.1.2.4, "JDBC.DRIVER and JDBC.URL_PFX"

A.1.2.1 DATA_DIRECTORY

The location of client database files is determined by the `DATA_DIRECTORY` parameter in the OSE.INI file.

- If the DATA_DIRECTORY parameter is not specified, the default value is as below
  - For SQLite on Win32, Linux and pure Java SE:<MOBILE_CLIENT_HOME>/sqlite/sqlite_db
  - For Berkeley DB on Win32, Linux and pure Java SE: <MOBILE_CLIENT_HOME>/bdb/data
  - For SQLite and Berkeley DB on Windows Mobile: \OraCE\sqlite_db
- Mobile Client database files, Oracle Database Mobile Server repository files, and temporary synchronization data are stored in the `DATA_DIRECTORY/<user>` directory, where *`<user>`* is the synchronization user id. The database repository files are named with the `.db` extension, such as `TERRY\mysqlite.db`. These files are used to manage the change control for transactions and synchronization for the user.
- Internal settings and parameters for the Mobile Client is stored in the `<mobileclient>/bin/oseconf` directory.

Example

Example of configuring the client database directory on a Win32 platform for Berkeley DB and SQLite:

```
SQLITE.DATA_DIRECTORY=C:\mobileclient\sqlite\sqlite_db
```

Example for setting the directory on a Linux platform:

```
SQLITE.DATA_DIRECTORY=/mobileclient/sqlite/sqlite_db
```

Example for setting the directory on a Blackberry:

```
SQLITE.DATA_DIRECTORY=file:///SDCard/databases/my_app
```

A.1.2.2 QUEUES

The `QUEUES` parameter specifies which type of snapshots the client will use in tracking the changes for Berkeley DB and SQLite databases. The following lists the two snapshot types:

- *Queue-based*: Both client and server changes are stored in a single queue. Whenever the snapshot is not locked by an application, the synchronization retrieves data from the In Queue and applies it to the base snapshot. At this point, the synchronization propagates data from the Out Queue to the server.

  Although both snapshot types rely on triggers, queue-based snapshots allow concurrent operations on Berkeley DB and SQLite databases while any synchronization is in progress. The Sync Agent's compose operation places modified data into the Out Queue. Later, the Sync Session uploads multiple client transactions delineated by a unique transaction id to the server.

  Set this type with `QUEUES`=YES.

- *State-based*. State-based snapshots decipher the difference in the state of the data between subsequent synchronization events. This snapshot type is more resource efficient than queue-based snapshots. To enable state-based snapshots, set the `QUEUES` parameter in the `OSE.INI` file to `NO`.

Syntax

```
SQLITE.QUEUES=YES|NO
```

A.1.2.3 LIMIT_CONNECTIONS

Set `LIMIT_CONNECTIONS` to `YES` when you want to limit the number of concurrent connections used by synchronization. Setting this parameter to `YES` keeps alive only the minimum required number of connections. If the `QUEUES` parameter is set to `YES`, the minimum number of connections necessary for synchronization is 2. If `QUEUES` is set to `NO`, only a single connection is required.

Setting the `LIMIT_CONNECTIONS` parameter is a trade-off between performance and memory limitations. This parameter is set to `YES` by default on all Blackberry devices for conserving memory.

Syntax

```
SQLITE.LIMIT_CONNECTIONS=YES|NO
```

A.1.2.4 JDBC.DRIVER and JDBC.URL_PFX

These parameters are used for Pure Java (PJ) SE and Pure Java ME clients to specify which JDBC driver is used by the client. These settings in ose.ini should be permanent and not be changed unless the client is removed and reinstalled. For more information on the values of these parameters for SQLite and Berkeley DB clients, see Section 5.1, "OSync Utility".

A.1.3 Java DB Mobile Client Parameters — JAVADB

The JAVADB section configures certain aspects of Java DB Mobile Clients. The following sections describe the mobile client parameters that you can modify:

- Section A.1.3.1, "DATA_DIRECTORY"
- Section A.1.3.2, "QUEUES"
- Section A.1.3.3, "SHUTDOWN_ON_CLOSE"

A.1.3.1 DATA_DIRECTORY

The location of Java DB database files is determined by the DATA_DIRECTORY parameter in the OSE.INI file. If the DATA_DIRECTORY parameter is not specified, the default value is <mobileclient>/javadb_home.

Additionally, the Mobile Client database files, the Oracle Database Mobile Server repository files, and the temporary synchronization data are stored in the DATA_DIRECTORY/<user> directory, where <user> is the synchronization user id.

Example:

Example for setting the directory on a Win32 platform:

```
JAVADB.DATA_DIRECTORY=C:\mobileclient\javadb_home
```

Example for setting the directory on a Linux platform:

```
JAVADB.DATA_DIRECTORY=/mobileclient/javadb_home
```

A.1.3.2 QUEUES

The QUEUES parameter specifies which type of snapshots the client will use in tracking the changes for Java DB databases. This parameter works the same way as the parameter SQLITE.QUEUES. Default value of this parameter is TRUE.

Syntax:

```
JAVADB.QUEUES=YES|NO
```

A.1.3.3 SHUTDOWN_ON_CLOSE

The SHUTDOWN_ON_CLOSE parameter controls if the Pure Java Sync engine should shut down the Java DB engine when it closes. Default value of this parameter is NO.

Syntax:

```
JAVADB.SHUTDOWN_ON_CLOSE=YES|NO
```

A.1.4 Berkeley DB Mobile Client Parameters—BDB

This section provides information on:

- Section A.1.4.1, "LARGE_RECORD_OPT"
- Section A.1.4.2, "Enable/Disable Ch-Werner JDBC Driver Parameter - STREAMING"

A.1.4.1 LARGE_RECORD_OPT

Setting LARGE_RECORD_OPT parameter will pass 'pragma large_record_opt' to the Berkeley DB database. This enables optimized storage of large records. Any record larger than the given number of bytes will be stored in a new format that improves read and write performance for large records.

For more information on 'large_record_opt' pragma, refer to *Oracle Berkeley DB Getting Started* with the SQL APIs document.

By default, this parameter is not set, and large record optimization in the Berkeley DB database is disabled.

Syntax

```
BDB.LARGE_RECORD_OPT=<value>
```

A.1.4.2 Enable/Disable Ch-Werner JDBC Driver Parameter - STREAMING

The STREAMING parameter is used by the Pure Java (PJ) SE client, the Android client and the OJEC client. This parameter specifies whether to use the Ch-Werner Java wrapper or the Ch-Werner JDBC driver to access the client database by choosing YES to use the Ch-Werner Java wrapper and NO to use the Ch-Werner JDBC driver. The default value of this parameter is NO.

The parameter is applicable to Berkeley DB Android client, Berkeley DB SE and SQLite SE clients and Berkeley DB OJEC client.

The Ch-Werner driver has the following two components:

- *Ch-Werner Java Wrapper:*

  A JDBC driver over the Java wrapper.

- *Ch-Werner JDBC Driver*:

  A Java wrapper with a Java Native Interface (JNI) layer over the SQL API.

The WERNER parameter determines the entry point in the stack. See below:

Table A–1 Entry Points of the WERNER Parameters

Parameter	Value
Ch-Werner JDBC Driver	BDB.STREAMING=FALSE
Ch-Werner Java Wrapper	BDB.STREAMING=TRUE
SQL API	Used for native clients

Syntax

```
BDB.STREAMING=YES|NO
```

Note: If you are using the Java wrapper, then empty blobs are supported.

If you are using the JDBC driver, then empty blobs are not supported.

When using the JDBC driver with an empty blob, an 'OPERATION_NOT_SUPPORTED' error is returned. As a workaround, store a null in the blob initially.

A.1.5 Background Sync Parameter—BGSYNC

The `DISABLE` parameter specifies whether the Sync Agent is disabled. Values are YES or NO. Disabling the Sync Agent prevents any automatic synchronization to be initiated for any user on this SQLite or BDB Mobile Client.

Syntax

```
BGSYNC.DISABLE=YES|NO
```

A.1.6 Network Parameters - NETWORK

This section provides information on:

- Section A.1.6.1, "DISABLE_SSL_CHECK"

A.1.6.1 DISABLE_SSL_CHECK

This ose.ini parameter is only for Pure Java (Android, Blackberry, SE, ME) and iOS clients. For other clients use the corresponding parameter in devmgr.ini (see section A.2.2.2).

Specifies whether the client can use certificates from mobile server that are not signed by trusted authority, such as self-signed certificate. The values are YES and NO, with NO being the default.

Set this parameter to YES if you want to use self-signed certificate for SSL encryption, but not to perform SSL authentication.

Syntax

```
NETWORK.DISABLE_SSL_CHECK=YES|NO
```

A.2 DEVMGR.INI File

The `DEVMGR.INI` file contains mobile client parameters for Device Management in the `DMC` section and the network parameters in the `NETWORK` section. For more information on device management parameters that can be modified before installing the client, see Section 7.2, "Configuring Mobile Clients Before Installation" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

The following sections describe the parameters for the `DMC` and `Network` sections:

- Section A.2.1, "Device Management Parameters—DMC Section"
- Section A.2.2, "Network Parameters—NETWORK Section"

A.2.1 Device Management Parameters—DMC Section

The Device Management parameters are as follows:

- Section A.2.1.1, "DISABLE_PROMPT"
- Section A.2.1.2, "PUSH_PORT"
- Section A.2.1.3, "UPDATE_DAY and UPDATE_TIME"
- Section A.2.1.4, "MAX_RETRY"
- Section A.2.1.5, "FREQUENCY"
- Section A.2.1.6, "DEBUG"

A.2.1.1 DISABLE_PROMPT

The `DISABLE_PROMPT` parameter accepts a `TRUE` or `FALSE` value, which causes the following action:

- `TRUE`: The device checks for software updates available on the server. If updates are available, these are brought down to the client and installed.
- `FALSE`: The device checks for software updates available on the server. If updates are available, the option to bring down the updates and install them is displayed to the user, who decides what action to take. If the client chooses to update, then these are brought down to the client and installed.

A.2.1.2 PUSH_PORT

The port number on the mobile device that accepts device management commands from the mobile server. By default, the port number is 8521. Do not modify on the client. Even though it is described here, you should only modify the `PUSH_PORT` variable in the INF file BEFORE the mobile client is installed. For more information, see Section 7.2, "Configuring Mobile Clients Before Installation" in the *Oracle Database Mobile Server Administration and Deployment Guide*.

A.2.1.3 UPDATE_DAY and UPDATE_TIME

The day and time to check for software updates for the client. You can modify day and time here or within the DMAgent UI. For more information on the DMAgent UI, see Section 7.7, "Using the Device Manager Agent (dmagent) on the Client" in the *Oracle Database Mobile Server Administration and Deployment Guide*. If you do want to modify them here, the values are as follows:

Day when the device checks for software updates. Used in combination with `UPDATE_TIME`.

UPDATE_DAY takes 0 - 8 which translates to the following days:

- Never = 0
- Daily = 1
- Sunday = 2
- Monday = 3
- Tuesday = 4
- Wednesday = 5
- Thursday = 6
- Friday = 7
- Saturday = 8

Time of day that the device checks for software updates from the mobile server. Used in combination with `UPDATE_DAY`. `UPDATE_TIME` can take values 0 - 23 which translates to the following time:

- 00:00 = 0
- 01:00 = 1
- 12:00 = 12
- 13:00 = 13
- 23:00 = 23

A.2.1.4 MAX_RETRY

Integer value that configures the maximum number of retry attempts before abandoning a server command.

A.2.1.5 FREQUENCY

The frequency of how many seconds between the client polls. The DMAGENT connects to the mobile server checking for new commands at the defined `FREQUENCY` interval.

A.2.1.6 DEBUG

If you turn on the `DEBUG` parameter in the `[DMC]` section, then this turns on the debugging for the device manager. All device manager debug messages are written to the `_dmdebug.txt` file.

To enable, set the `DEBUG` parameter in the `[DMC]` section to 1. Set to 0 to turn off debug feature, which is the default.

Default value: 0

A.2.2 Network Parameters—NETWORK Section

The following parameter configures how the client interacts over the network:

- Section A.2.2.1, "SERVER_URL"
- Section A.2.2.2, "DISABLE_SSL_CHECK"
- Section A.2.2.3, "HTTP_PROXY"

A.2.2.1 SERVER_URL

This parameter points to the mobile server. It communicates with the mobile server over HTTP or HTTPS. The expected syntax for the `SERVER_URL` parameter is as follows:

```
HTTP://<host>:<port>/mobile
```

For example:

```
[NETWORK]
SERVER_URL=HTTPs://myhost:8888/mobile
```

A.2.2.2 DISABLE_SSL_CHECK

For Pure Java and iOS clients use the corresponding parameter in ose.ini. For more information, see Section A.1.6.1, "DISABLE_SSL_CHECK".

You can use certificates that are not signed by a trusted authority, such as a self-signed certificate, on the mobile server. Set the following parameter in the `NETWORK` section on the client device:

```
[NETWORK]
DISABLE_SSL_CHECK=YES
```

This parameter enables the client to use the self-signed certificate for SSL encryption, but not to perform SSL authentication.

A.2.2.3 HTTP_PROXY

If user has a proxy between the mobile client and the mobile server, then in order for the Device Manager (dmagent) to access the mobile server to poll for command, then configure this parameter to the proxy server URL, including port number.

Format is `<hostname>:<port>`, as follows:

```
[NETWORK]
HTTP_PROXY=proxy.foo.com:8080
```

A.3 Sample OSE.INI and DEVMGR.INI Files

The following content is displayed from a sample `OSE.INI` file.

```
SQLITE.DATA_DIRECTORY=C:\mobileclient\sqlite
SQLITE.QUEUES=YES
OSE.RESUME=NO
BGSYSNC.DISABLE=NO
```

The following content is displayed from a sample `DEVMGR.INI` file.

```
[NETWORK]
DISABLE_SSL_CHECK=YES
HTTP_PROXY=proxy.foo.com:8080
```

Index