Oracle® Database Lite

Developer's Guide

10*g* (10.3.0)

B28923-01

April 2007

Oracle Database Lite Developer's Guide 10*g* (10.3.0)

B28923-01

Contents

Preface

This preface introduces you to the *Oracle Database Lite 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

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation
Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation
This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services
Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Send Us Your Comments

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: olitedoc_us@oracle.com
- FAX: (650) 506-7355. Attn: Oracle Database Lite
- Postal service:

  Oracle Corporation
  Oracle Database Lite Documentation
  500 Oracle Parkway, Mailstop 1op2
  Redwood Shores, CA 94065
  U.S.A.

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

1 Overview

This chapter provides an introduction to Oracle Database Lite and presents an overview of the application development process, using the Mobile Development Kit and its components. This chapter discusses the following topics.

1.1 Introduction

Oracle Database Lite facilitates the development, deployment, and management of Mobile database applications for a large number of Mobile users. A Mobile application is an application that can run on Mobile devices without requiring constant connectivity to the server. An Oracle Lite database application requires a local database on the Mobile device, whose content is a subset of data that is stored in the enterprise data server. Modifications made to the local database by the application are occasionally reconciled with the server data. The technology used for reconciling changes between the Mobile database and the enterprise database is known as data synchronization.

Mobile database applications can be developed in many ways, as follows:

- The most common way is to develop native C or C++ applications for specific Mobile platforms. C++ applications can access the Oracle Database Lite database using the Simple Object Data Access API (SODA), an easy-to-use C++ interface that is optimized for the object-oriented and SQL functionality of Oracle Database Lite. For more information about SODA, refer the SODA API documentation, which is installed as part of the Mobile Development Kit.
- Applications that need a standard interface and work with multiple database engines can use either the Open Database Connectivity (ODBC) interface, Active Data Object (ADO) interface, or some other interface built on top of ODBC. ADO.NET can be used on Win32 and Windows CE. Another way to develop a Mobile database application is to use Java and the Java Database Connectivity (JDBC) interface. Oracle Database Lite also offers a third way to develop Mobile database applications using the servlet based Web model called Web-to-Go.

- Web-to-Go applications can be built using Web technologies, such as servlet, Java Sever Pages (JSP), applet, HTML, and JDBC.
- Symbian applications that need a standard interface and work with multiple database engines can use either the Open Database Connectivity (ODBC) interface or some other interface built on top of ODBC.

Once the application has been developed, it has to be deployed. Deployment of applications is concerned with setting up the server system so that end users can easily install and use the applications. The nerve center of the server system for Oracle Database Lite applications is the Mobile Server which is where the Mobile applications are deployed. Deployment consists of five major steps:

1. Designing the server system to achieve the required level of performance, scalability, security, availability, and connectivity. Oracle Database Lite provides tools such as the "Consperf" utility to tune the performance of data synchronization. It also provides benchmark data that can be used for capacity planning for scalability. Security measures such as authentication, authorization, and encryption are supported using the appropriate standards. Availability and scalability are also supported by means of load balancing, caching, and the transparent switch-over technologies of the Oracle Application Server (Oracle AS) and the Oracle database server.
2. Publishing the application to the server. This refers to installing all the components for an application on the Mobile Server. Oracle Database Lite provides a tool called the Packaging Wizard that can be used to publish applications to the Mobile Server.
3. Provisioning the applications to the Mobile users. This phase includes determining user accesses to applications with a specified subset of data. Oracle Database Lite provides a tool called the Mobile Manager to create users, grant privileges to execute applications, and define the data subsets for them, among others. You can also use the Java API to provision applications.
4. Testing for functionality and performance in a real deployment environment. A Mobile application system is a complex system involving many Mobile device client technologies (such as, operating systems, form factors, and so on), many connectivity options (such as, LAN, Wireless LAN, cellular, wireless data, and other technologies), and many server configuration options. Nothing can substitute for the real life testing and performance tuning of the system before it is rolled out. Particular attention should be paid to tuning the performance of the data subsetting queries, as it is the most frequent cause of performance problems.
5. Determining the method of initial installation of applications on Mobile devices (application delivery). Initial installation involves installing the Oracle Database Lite client, the application code, and the initial database. The volume of data required to install applications on a Mobile device for the first time could be quite high, necessitating the use of either a high-speed reliable connection between the Mobile device and the server, or using a technique known as offline instantiation. In offline instantiation, everything needed to install an application on a Mobile device is put on a CD or a floppy disk and physically mailed to the user. The user then uses this media to install the application on the device by means of a desktop machine. Oracle Database Lite provides a tool for offline instantiation.

After deployment, both the application and the data schema may change because of enhancements or defect resolution. The Oracle Database Lite Mobile Server takes care of managing application updates and data schema evolution. The only requirement is that the administrator must republish the application and the schema. The Mobile

Server automatically updates the Mobile clients that have older version of the application or the data.

Oracle Database Lite installation provides you with an option to install the Mobile Server or the Mobile Development Kit. For application development, you will need to install the *Mobile Development Kit* on your development machine. However, as discussed later in this document, the development examples require the Mobile Server to be running. Hence, if you intend to recreate the sample applications on your system, you must install the Mobile Server, preferably on a different machine. The installation of the Mobile Server requires an Oracle database instance to be running. You can use an existing test database as well. The Mobile Server stores its metadata in this database.

1.2 Supported Technologies for Application Development

Oracle Database Lite is an integrated framework that simplifies the development, management, and deployment of mobile applications on the following mobile platforms, operating systems, and hardware:

Platform	Programming Languages	Operating System	Hardware
Win32 on a laptop or notebook	When you develop on a laptop, you are using one of the Windows operating systems. You can use any of the languages mentioned in this book. However, C, C++ are better for creating applications with a good user interface. The languages available are C, C++, C#, Java, Visual Basic, JSPs and Servlets. You can use Visual Studio 2003 for development.	Windows 2000	Pentium processor
PocketPC	C, C++, Visual Basic, Java applications (no Servlet or JSP support), SODA, ADO.NET. You can use Visual Studio 2005 for development.	Windows CE (WinCE)	ARM
Linux	Java, C, C++	Linux Redhat 3.0	X86
Symbian OS on Nokia and Motorola	C, C++	Symbian OS versions 7.0 and 8.0	ARM

Oracle Database Lite provides the Mobile Development Kit, which includes facilities, tools, APIs, and sample code for you to develop your applications. There are three application models:

- Section 1.2.1, "Native Applications"
- Section 1.2.2, "Standalone Java Applications"
- Section 1.2.3, "Web Applications"

1.2.1 Native Applications

Native applications are built using C, C++, Visual C++, Visual Basic, Embedded Visual tools, ActiveX Data Objects (ADO), and MetroWerks CodeWarrior. The application must be compiled against the mobile device operating system, such as the Windows CE platform.

Use ODBC to access the Oracle Lite database on the client. Alternatively, you could use JDBC to access the local client database. See Section 2.4.1, "JDBC" and Section 2.4.2, "ODBC" for more information on accessing the database with either of these interfaces.

See Chapter 8, "Native Application Development" for more information on C and C++.

1.2.2 Standalone Java Applications

Standalone Java applications do not include Web and J2EE technology—such as Servlets and JSPs. Instead, Java applications revolve around using JDBC driver to access the Oracle Lite database on the client platform, and use AWT and SWING classes to build the application UI. In addition, the database supports Java stored procedures and triggers.

Your Java/JDBC application must be compiled for the particular mobile device JVM environment, which can be different across various client devices. Thus, when you are developing your Java application, do the following:

1. Check the environment: Verify that the `olite40.jar`, which is located in `OLITE_HOME/bin`, is in your `CLASSPATH`, which should have been modified during installation.
2. Load the JDBC driver in to your applications. The following is an example:

   ```
   Class.forName("oracle.lite.poljdbc.POLJDBCDriver");
   ```

3. Connect to the Oracle Lite database installed on the client. If your database is on the same machine as the JDBC application, connect using the native driver connection URL syntax, as follows:

   ```
   jdbc:polite:dsn
   ```

   Or if not local, connect as follows:

   ```
   jdbc:polite@[hostname]:[port]:dsn
   ```

See Chapter 9, "Java Application Development" and Chapter 10, "JDBC Programming" for more information.

1.2.3 Web Applications

You can execute existing Web applications using the J2EE Java technologies, such as servlets and JSPs, in a disconnected mode without modifying the code base. Web-to-Go is a development option for Web applications, and can be executed on laptops using Windows 2000. Web-to-Go applications use Java servlets and JSPs that may invoke JDBC to access the database, as opposed to using application APIs, such as C or C++.

For more information, see Chapter 6, "Developing Mobile Web-to-Go Applications".

1.3 Oracle Database Lite Application Model and Architecture

In the Oracle Database Lite application model, each application defines its data requirements using a publication. A publication is akin to a database schema and it contains one or more publication items. A publication item is like a parameterized view definition and defines a subset of data, using a SQL query with bind variables in it. These bind variables are called *subscription parameters* or *template variables*.

A subscription defines the relationship between a user and a publication. This is analogous to a newspaper or magazine subscription. Accordingly, once you subscribe to a particular publication, you begin to receive information associated with that publication. With a newspaper you receive the daily paper or the Sunday paper, or both. With Oracle Lite you receive snapshots, and, depending on your subscription parameter values, those snapshots are partitioned with data tailored for you.

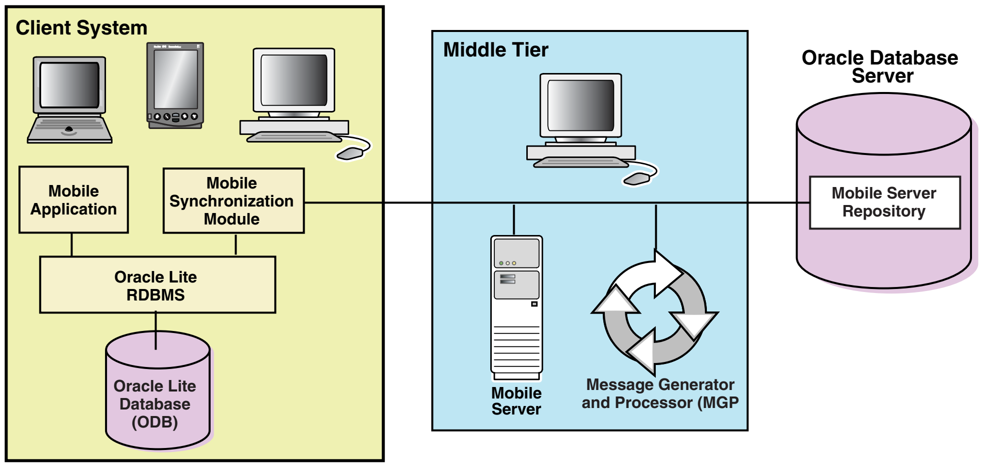
When a user synchronizes the Mobile client for the first time, the Mobile client creates an Oracle Database Lite database on the client machine for each subscription that is provisioned to the user. The Mobile client then creates a snapshot in this database for each publication item contained in the subscription, and populates it with data retrieved from the server database by running the SQL query (with all the variables bound) associated with the publication item. Once installed, Oracle Database Lite is transparent to the end user; it requires minimal tuning or administration.

As the user accesses and uses the application, changes made to Oracle Database Lite are captured by the snapshots. At a certain time when the connection to the Mobile Server is available, the user may synchronize the changes with the Mobile Server. Synchronization may be initiated by the user using the Oracle Database Lite Mobile Synchronization application (msync) directly, by programmatically calling the Mobile Synchronization API from the application, or in the case of Web applications, the synchronization option can be used from the Web-to-Go workspace to synchronize the data. The Mobile Synchronization application communicates with the Mobile Server and uploads the changes made in the client machine. It then downloads the changes for the client that are already prepared by the Mobile Server.

A background process called the Message Generator and Processor (MGP), which runs in the same tier as the Mobile Server, periodically collects all the uploaded changes from many Mobile users and then applies them to the server database. Next, MGP prepares changes that need to be sent to each Mobile user. This step is essential because the next time the Mobile user synchronizes with the Mobile Server, these changes can be downloaded to the client and applied to the client database.

Figure 1–1 illustrates the architecture of Oracle Database Lite applications.

Figure 1–1 Oracle Database Lite Architecture



Note: Web-to-Go clients have one additional component, a light weight HTTP listener that is not shown in the diagram.

1.3.1 Oracle Database Lite RDBMS

The Oracle Database Lite RDBMS is a small footprint, Java-enabled, secure, relational database management system created specifically for laptop computers, handheld computers, PDAs, and information appliances. The Oracle Database Lite RDBMS runs on Windows 2000/XP, Windows CE/Windows Mobile, Linux, and Symbian. Oracle Database Lite RDBMS provides JDBC, ODBC, and SODA interfaces to build database applications from a variety of programming languages such as Java, C/C++, and Visual Basic. These database applications can be used while the user is disconnected from the Oracle database server.

When you install the Mobile Development Kit, the Oracle Database Lite RDBMS and all the utilities listed in Appendix C are installed on your development machine. In a production system, when the Mobile Server installs Oracle Database Lite applications, only the RDBMS, the Mobile Sync, and Mobile SQL applications are installed on the client machine.

1.3.2 Mobile Sync

Mobile Sync (msync) is a small footprint application that resides on the Mobile device. Mobile Sync enables you to synchronize data between handheld devices, desktop and laptop computers and Oracle databases. Mobile Sync runs on Windows 2000/XP, Windows CE/Windows Mobile, and Linux.

Mobile Sync synchronizes the snapshots in Oracle Database Lite with the data in corresponding Oracle data server. These snapshots are created by the Mobile Server for each user from the publication items associated with a Mobile application. The Mobile Server also coordinates the synchronization process.

The Mobile Sync application communicates with the Mobile Server using any of the supported protocols (e.g., HTTP or HTTPS). When called by the Mobile user, the Mobile Sync application first collects the user information and authenticates the users with the Mobile Server. It then collects the changes made to Oracle Database Lite (from the snapshot change logs) and uploads them to the Mobile Server. It then downloads the changes for the user from the Mobile Server and applies them to the Oracle Database Lite.

In addition to this basic function, the Mobile Sync application can also encrypt, decrypt, and compress transmitted data.

When you install the Mobile Development Kit, the Mobile Sync application is also installed on your development machine. The Mobile Server also installs the Mobile Sync on the client machine as part of application installation.

Unlike base tables and views, snapshots cannot be created in Oracle Database Lite by using SQL statements. They can only be created by the Mobile Server based on subscriptions which are derived from publication items associated with an application. This point is discussed further later in this chapter and in Chapter 4.

1.3.3 Mobile Server

The Mobile Server is a mid-tier server that provides the following features.

- Application Publishing
- Application Provisioning
- Application Installation and Update
- Data Synchronization

The Mobile Server has two major modules called the Resource Manager and the Consolidator Manager. The Resource Manager is responsible for application publishing, application provisioning, and application installation. The Consolidator Manager is responsible for data and application synchronization.

Application publishing refers to uploading your application to the Mobile Server so that it can be provisioned to the Mobile users. Once you have finished developing your application, you can publish it to the Mobile Server by using the development tool called the Packaging Wizard.

Application provisioning is concerned with creating subscriptions for users and assigning application execution privilege to them. Application provisioning can also be done in one of two ways.

- Using the administration tool called the Mobile Manager, you can create users and groups, create subscriptions for users by assigning values to subscription parameters, and give users or groups privileges to use the application.
- Using the Resource Manager API, you can programmatically perform the above tasks.

End users install Mobile applications in two steps.

1. As the Mobile user, browse the setup page on the Mobile Server and choose the setup program for the platform you want to use. The setup program only runs on Windows 32 platforms. For Windows 32 based client systems, you can download the setup program directly to the Windows 32 system and execute it to set up the Oracle Database Lite client. For Windows CE devices, you must download the setup program to your Windows 32 desktop first and execute it there. Then you must use ActiveSync for Windows CE to install the Oracle Database Lite client on the device.
2. Run the Mobile Sync (msync) command on your Mobile device, which prompts for the user name and password. The Mobile Sync application communicates with the Consolidator Manager module of the Mobile Server and downloads the applications and the data provisioning for the user.

After the installation of the applications and data, you can start using the application. Periodically, use Mobile Sync or a custom command to synchronize your local database with the server database. This synchronization updates all applications that have changed.

1.3.4 Message Generator and Processor (MGP)

The Consolidator Manager module of the Mobile Server uploads the changes from the client database to the server, and it downloads the relevant server changes to the client. But it does not reconcile the changes. The reconciliation of changes and the resolution of any conflicts arising from the changes are handled by MGP. MGP runs as a background process which can be controlled to start its cycle at certain intervals.

Each cycle of MGP consists of two phases: Apply and Compose.

The Apply Phase

In the apply phase, MGP collects the changes that were uploaded by the users since the last apply phase and applies them to the server database. For each user that has uploaded his changes, the MGP applies the changes for each subscription in a single transaction. If the transaction fails, MGP will log the reason in the log file and stores the changes in the error file.

The Compose Phase

When the apply phase is finished, MGP goes into the compose phase, where it starts preparing the changes that need to be downloaded for each client.

Applying Changes to the Server Database

Because of the asynchronous nature of data synchronization, the Mobile user may sometimes get an unexpected result. A typical case is when the user updates a record that is also updated by someone else on the server. After a round of synchronization, the user may not get the server changes.

This happens because the user's changes have not been reconciled with the server database changes yet. In the next cycle of MGP, the changes will be reconciled with the server database, and any conflicts arising from the reconciliation will be resolved. Then a new record will be prepared for downloading the changes to the client. When the user synchronizes again (the second time), the user will get the record that reflects the server changes. If there is a conflict between the server changes and the client changes, the user will get the record that reflects either the server changes or the client changes, depending on how the conflict resolution policy is defined.

1.3.5 Mobile Server Repository

The Mobile Server repository contains all the information needed to run the Mobile Server. The information is usually stored in the same database where the application data reside. The only exception to this is in cases where the application data resides in a remote instance and there is a synonym defined in the Mobile Server to this remote instance.

Changes to the repository should only be made using the Mobile Server Mobile Manager or the Resource Manager API.

1.4 Execution Models for Oracle Lite Database

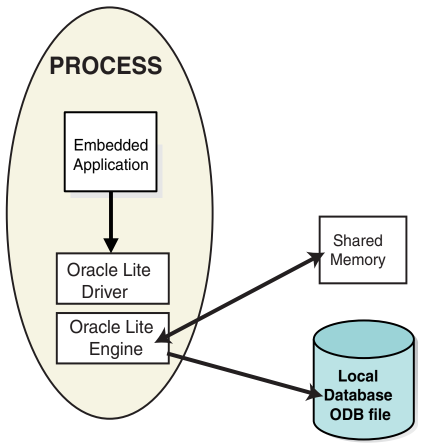
How your application integrates with the Oracle Lite database depends on how you design the execution model, as described in the following sections:

1. Do you want to use the Oracle Lite database to store data solely for a single application? Yes; use the embedded application option. See Section 1.4.1, "Embedded Application in Single Process" for more information.
    - Do you want to have multiple applications access the same database? See Section 1.4.2, "Multiple Processes Accessing the Same Database" for more information.
    - Do you want your application to access the database remotely? See Section 1.4.3, "Multiple Embedded Application Clients Accessing Remote Database" for more information.
2. Do you want to use the Oracle Lite database to store changes that will be synchronized with a back-end Mobile Server repository? Yes; use the Mobile client option. See Section 1.4.4, "Embedded Mobile Client in Single Process" for more details.
    - Do you want your application to access the database remotely? See Section 1.4.5, "Multiple Clients Accessing Remote Database" for more information.

1.4.1 Embedded Application in Single Process

As demonstrated in Figure 1–2, if you chose to build a standalone application with the Oracle Lite database embedded in the application, then when the application is launched, the Oracle Lite database libraries are loaded into the same process as the application.

Figure 1–2 Embedded Application With ODB Libraries in Single Process

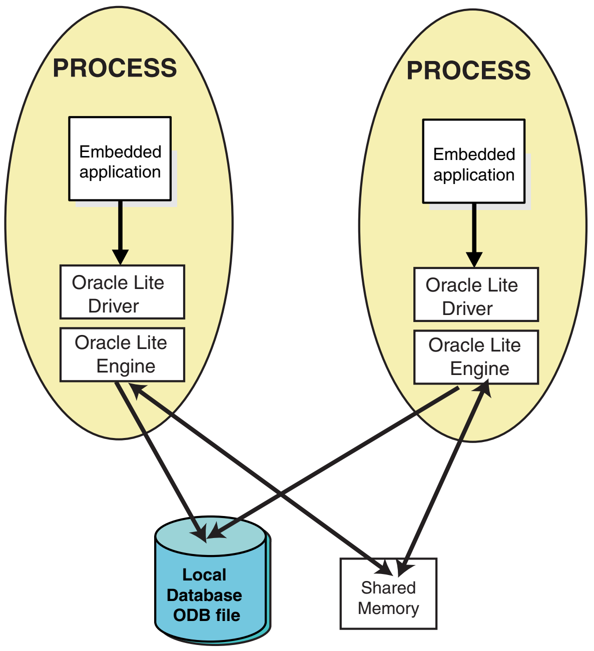


See Section 2.3, "Creating and Managing the Database in an Embedded Application" for more information on how to embed an Oracle Lite database into a standalone application.

1.4.2 Multiple Processes Accessing the Same Database

You can configure your application processes to share the same database. Thus, when each application is launched, each application exists in its own process and can access the same database independently. In this scenario, Oracle Database Lite libraries use shared memory to coordinate locking between both processes.

Figure 1–3 Applications in Multiple Processes Accessing Single Database



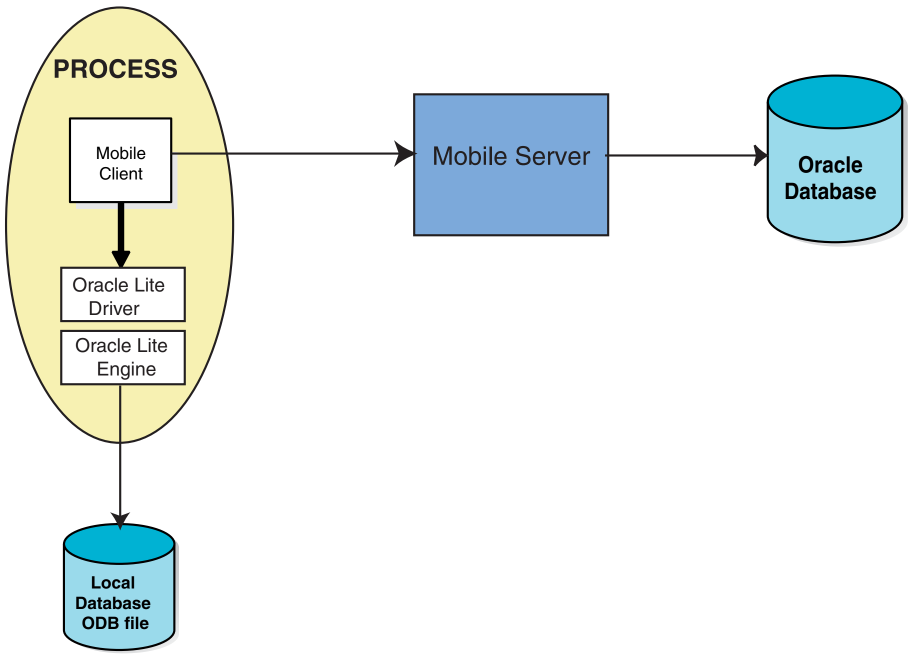
1.4.3 Multiple Embedded Application Clients Accessing Remote Database

If you are embedding a database into your application software, but you want the applications to exist remotely from the data, then use the client/server embedded approach with the multi-user service, as described in Section 2.5, "Oracle Database Lite Multi-User Service".

1.4.4 Embedded Mobile Client in Single Process

If you chose to install the Mobile client and synchronized your user on a single device, then when you launch your application, the Oracle Lite database libraries are loaded into the same process as your application. This is the default scenario and is demonstrated in Figure 1–4.

Figure 1–4 Diagram of Mobile Client and ODB Libraries in SIngle Process



For details of how to package an embedded Oracle Lite database in your application, see Section 2.2, "Creating and Managing the Database for a Mobile Client".

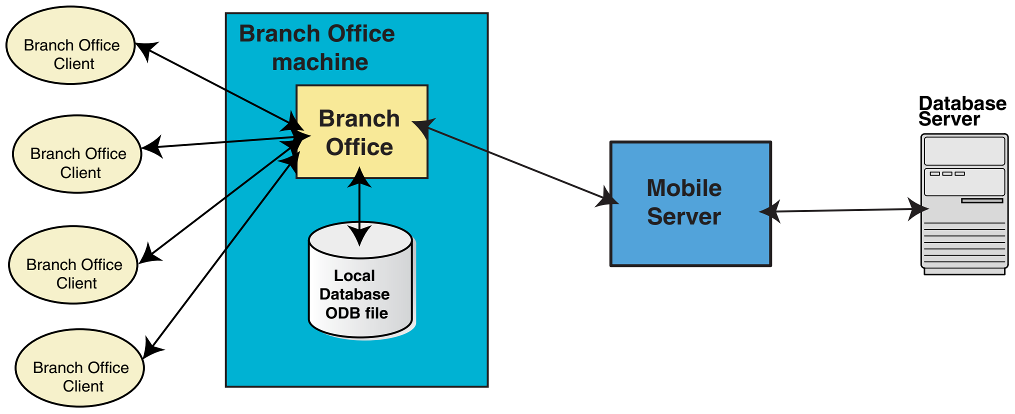
1.4.5 Multiple Clients Accessing Remote Database

If you have several remote clients accessing the same data, you can use Branch Office to facilitate the remote applications. Figure 1–5 demonstrates how multiple remote Branch Office clients access the data through the Branch Office machine to the Mobile Server and finally accessing the back-end Oracle database.

The Branch Office machine contains the Branch Office executables and the local Oracle Lite database, which all clients access for their information. When a synchronization is requested, information is communicated between the Branch Office and the back-end database through the Mobile Server.

Note: Oracle Database Lite is not identical to the Oracle database; thus, for large amounts of transferred data, use the Oracle Standard Edition Database.

Figure 1–5 Using Branch Office for Managing Multiple Clients that Access a Remote Database



See Chapter 10, "Manage Your Branch Office" in the *Oracle Database Lite Administration and Deployment Guide* for more information.

1.5 Mobile Development Kit (MDK)

Before you develop an application using Oracle Database Lite, you should install the Oracle Database Lite Mobile Development Kit (MDK) on the machine on which you intend to develop your application. For instructions on how to install the Mobile Development Kit, see Section 2.3, "Installing Oracle Databse Lite" in the *Oracle Database Lite Getting Started Guide*.

The Oracle Database Lite Mobile Development Kit includes the following components.

- Oracle Database Lite RDBMS—A lightweight, object-relational database management system
- Mobile Database Workbence (MDW)—A development tool for creating a publication.
- Packaging Wizard—A tool to publish applications to the Mobile Server
- Mobile Sync—A transactional synchronization engine
- mSQL—An interactive tool to create, access, and manipulate Oracle Database Lite on laptops and handheld devices

Using any C, C++, or Java development tool in conjunction with the Mobile Development Kit for Windows, you can develop your Mobile applications for Windows against Oracle Database Lite, and then publish the applications to the Mobile Server by using the Packaging Wizard. See Section 2.3, "Installing Oracle Databse Lite" in the *Oracle Database Lite Getting Started Guide* for instructions on how to install the Mobile Server.

Once you have published the applications to the Mobile Server, you can use the Mobile Manager to provision the applications to the Mobile users. Provisioning involves specifying the values of the subscription parameters used for subsetting the data needed by the application for a particular user. A user to whom an application has been provisioned can then log in to the Mobile Server and request it to set up everything the user needs to run the applications on the user's device.

The Mobile Development Kit is installed in `<`*`ORACLE_HOME`*`>\Mobile\Sdk`. The `bin` directory contains, among other things:

- The Oracle Database Lite RDBMS and its components, including Mobile SQL (`msql.exe`). Mobile SQL is written in Java. It requires the Java runtime environment JRE 1.4.2 or higher to be installed on your system before you can use it. If you have installed JDK 1.4.2 or higher, the JRE is already installed in your machine.
- Mobile Sync (msync), the executable (`msync.exe`) and the Java wrapper for it.
- Packaging Wizard (`runwtgpack.bat`)
- Mobile Database Workbench (MDW)

The Samples directory `<ORACLE_HOME>\Mobile\Sdk\Samples` directory contains some sample applications. Section 2.11, "Using Oracle Database Lite Samples" describes the sample programs and explains how to run them. You should familiarize yourself with the various Oracle Database Lite features by perusing the source code and running the samples.

When you install the MDK, it installs a starter database file in the `<ORACLE_HOME>\Mobile\Sdk\OLDB40` directory named `polite.odb`.

Note: The `polite.odb` starter database is not the name of the Mobile client database. For information on what Oracle Lite database (ODB) files are installed on the client, see Section 2.2, "Synchronizing or Executing Applications on the Mobile Client" in the *Oracle Database Lite Administration and Deployment Guide*.

When you install the Mobile Development Kit, the installer sets the `PATH` environment variable to include the bin directory of the Mobile Development Kit. You can use the Command Prompt on your Windows 32 machine to do the following quick test.

At the command prompt, enter the following.

```
msql system/manager@jdbc:polite:polite
...
SQL>create table test (c1 int, c2 int);
Table created
SQL>insert into test values(1,2)
1 row(s) created
SQL>select * from test;

                  C1 | C2
                 ----+----
                  1  |  2
SQL>rollback;
Rollback completed
SQL>exit
```

1.5.1 Mobile SQL (mSQL)

Mobile SQL is an interactive tool that allows you to create, access, and manipulate Oracle Database Lite on laptops and handheld devices. The mSQL installations on laptops cannot be used to create a database, but can create a database on hand-held devices. Using mSQL, you can perform the following actions.

- Create database objects such as tables and views
- View tables
- Execute SQL statements

The mSQL tool is installed with the Mobile Development Kit installation. It is also installed by the Mobile Server as part of application installation. The mSQL tool for the Windows 32 platform is a command line tool that is similar to the Oracle SQL*Plus tool, but does not provide compatibility with SQL*Plus. The mSQL tool for Windows CE supports a graphical user interface. See Appendix A.1, "The mSQL Tool" for details.

Note: UTF8 SQL Scripts are not supported in mSQL.

1.5.2 Using the Mobile Database Workbence

The Mobile Database Workbench (MDW) is a new tool that enables you to iteratively create and test publications—testing each object as you add it to a publication. Publications are stored within a project, which can be saved and restored from your file system, so that you can continue to add and modify any of the contained objects within it.

All work is created within a project, which can be saved to the file system and retrieved for further modifications later. Once you create the project, start creating the publication items, sequences, scripts and resources that are to be associated with the publication. You can create the publication and associated objects in any order, but you always associate an existing object with the publication. Thus, it saves time to start with creating the objects first and associating it with the publication afterwards.

For detailed information on how to use MDW, see Chapter 5, "Using Mobile Database Workbench to Create Publications".

1.5.3 Using the Packaging Wizard

The Packaging Wizard is a graphical tool that enables you to perform the following tasks.

1. Create a new Mobile application.
2. Edit an existing Mobile application.
3. Publish an application to the Mobile Server.

When you create a new Mobile application, you must define its components and files. In some cases, you may want to edit the definition of an existing Mobile application's components. For example, if you develop a new version of your application, you can use the Packaging Wizard to update your application definition. The Packaging Wizard also enables you to package application components in a JAR file which can be published using the Mobile Manager. The Packaging Wizard also enables you to create SQL scripts which can be used to execute any SQL statements in the Oracle database.

For detailed information on how to use the Packaging Wizard, see Chapter 7, "Using the Packaging Wizard".

1.6 Java Support

For more information, refer Chapter 9.1, "Java Support for Applications".

1.7 Data Source Name

For full details, refer to Chapter 8.2, "Data Source Name".

2 The Oracle Database Lite RDBMS

This chapter presents the Oracle Database Lite Relational Database Management System (RDBMS). It discusses the following topics:

2.1 Oracle Lite Database Overview

The Oracle Lite database is compliant to the SQL92 standard and compatible to Oracle databases. In addition, it is compliant to the ACID requirements for transaction support. Because it is a small database specifically designed for a client device, it has a small footprint and easy to administer. The Oracle Lite database can be installed on the following platforms: Linux, UNIX, Windows (Win32) and WinCE platforms.

You can use the Oracle Lite database either with the Mobile client and use synchronization to replicate data between the client and the Oracle database or you can embed the Oracle Lite database within an independent application of your own design. Either way, you use a small database that contains the client data—known as the Oracle Lite database. Most of the data is stored in a file with an ODB extension; any BLOB objects—either binary or character—and the indexes are stored in a file with an OBS extension. The Oracle Lite database exists solely to store and retrieve the user data specific to this device. It is not a replication of the entire Oracle database.

Because BLOB data and indexes are stored in an OBS file, there is no limit for BLOB data or indexes. The limitation for BLOB data and indexes is the space limitations of the operating system or 16 terabytes. There still exists a 4 GB limitation for the ODB file; however, this is not as much of an issue now that BLOB data can be stored in the OBS files.

Note: If you have been using the Oracle Lite database prior to the 10.3 release, you can upgrade your database to remove all BLOB data from within it and transfer the BLOB objects to OBS files by using the `defragdb` utility, which is documented in Section 3.6, "Defragmentation and Reducing Size of the Client Database" in the *Oracle Database Lite Administration and Deployment Guide*.

Oracle Database Lite creates all ODB and OBS files with an automatic name and assigns a data source name (DSN). The DSN is used to connect to the database using ODBC, JDBC or ADO.NET APIs. In order to make the connection, you must know the DSN name for your ODB file. When you install the Mobile Development Kit, a default database is installed with database name of `polite.odb` and DSN name of `polite`. However, when you synchronize, an Oracle Lite database is created for each publication (under a directory named after each user). For details on the name and location of the client Oracle Lite database, see Section 2.2, "Synchronizing or Executing Applications on the Mobile Client" in the *Oracle Database Lite Administration and Deployment Guide*. The DSN for these ODB files is a combination of the username followed by the ODB name.

The Oracle Lite database is an RDBMS that supports ODBC, JDBC, ADO.NET and SODA interfaces. SODA is an Oracle Database Lite specific C++ object API created to access the Oracle Lite database. SODA provides access to SQL as well as object-oriented functionality. See Section 2.4, "Data Access APIs" for more information on each language.

2.2 Creating and Managing the Database for a Mobile Client

When you use the Mobile client and Mobile Server to replicate data between the back-end Oracle database and your Mobile device, a small Oracle Lite database (ODB file) is created on your Mobile device to contain the data—that is stored in tables known as snapshots. The snapshot tables are used to track the modifications that the client makes on the data, which is then replicated during the synchronization process to the back-end database. All of this activity is transparent to the client. Your application queries and modifies data using SQL as if interacting with any Oracle database.

The Oracle Lite database for the Mobile client is automatically created on the first synchronization request. In addition, the data is replicated and updated with the data on the Oracle database automatically for you. See Section 3.3, "What is The Process for Setting Up a User For Synchronization?" for techniques that can be used to create publication items on the Mobile Server, which then automatically creates snapshots on the client when you synchronize with the database.

Note: For details on the name and location of the Oracle Lite database for the Mobile client, see Section 2.2, "Synchronizing or Executing Applications on the Mobile Client" in the *Oracle Database Lite Administration and Deployment Guide*.

2.3 Creating and Managing the Database in an Embedded Application

When you want to create your own application that does not use the formal Mobile client model and is installed on its own, with an embedded Oracle Lite database as the storage vehicle for your application, perform the following:

- Section 2.3.1, "Install Oracle Database Lite Runtime"
- Section 2.3.2, "Creating the Default Starter Oracle Lite Database for an Embedded Application"
- Section 2.3.3, "Creating a Unique Oracle Lite Database for an Embedded Application"
- Section 2.3.4, "Creating Users for Your Embedded Oracle Lite Database"
- Section 2.3.5, "Packaging Your Embedded Application With the Oracle Database Lite Runtime"

2.3.1 Install Oracle Database Lite Runtime

In order to create the Oracle Lite database and embed it into your application, you must include not only the Oracle Lite database, but certain DLLs in your application. In order to develop applications, you must install the Mobile Development Kit. See Section 2.3, "Installing Oracle Database Lite" in the *Oracle Database Lite Getting Started Guide* for full details.

2.3.2 Creating the Default Starter Oracle Lite Database for an Embedded Application

If you want to create an application that uses a small file-based database, you can develop your application around an Oracle Lite database. When you installed the Mobile Development Kit, the following was created automatically for you:

1. An ODBC data source name (DSN) `POLITE` and a starter database called `POLITE.ODB` are created. The location of the new database for the DSN `POLITE` is <*ORACLE_HOME*>`\Mobile\Sdk\oldb40`.
2. A default user named `SYSTEM` is created when the starter database is created. This user contains all database privileges and has a password of `MANAGER`.

You can develop your application to store and retrieve any information in the database using any of the APIs listed in Section 2.4, "Data Access APIs".

2.3.3 Creating a Unique Oracle Lite Database for an Embedded Application

If you do not want to use the default Oracle Lite database, described in Section 2.3.2, "Creating the Default Starter Oracle Lite Database for an Embedded Application", then you can create your own database file. First, create a data source name (DSN) for the database and then create the database itself, as described in the following sections:

- Section 2.3.3.1, "Creating a Data Source Name with ODBC Administrator"
- Section 2.3.3.2, "Creating a New Oracle Lite Database for the Embedded Application"
- Section 2.3.3.3, "Connecting to Your New Oracle Lite Database"

2.3.3.1 Creating a Data Source Name with ODBC Administrator

The data source name (DSN) points to the physical location of the ODB file. The DSN is used when creating the Oracle Lite database (ODB) file. How you create the DSN is platform-dependent, as described in the following sections:

- Section 2.3.3.1.1, "Creating DSN on a Windows System"
- Section 2.3.3.1.2, "Creating DSN on a LINUX System"

2.3.3.1.1 Creating DSN on a Windows System Create the DSN on a Windows system through the Microsoft ODBC Administrator, which is a tool that manages the ODBC.INI file and associated registry entries in Windows 2000/XP. Within this tool, add the data source name for your ODB file and specify the database file you want to dedicate as the default for the data source name.

The ODBC.INI file is available in Windows under %WINDIR% and in Linux under $OLITE_HOME/bin. For the Linux platform, you must have write permissions on the directory where this is located to be able to modify them.

For more information on DSN properties, see Table 2–1 and Table 2–2.

Note: The name of the ODB file is used in the next step: Section 2.3.3.2, "Creating a New Oracle Lite Database for the Embedded Application". For more information on the ODBC Administrator, and for instructions on creating a data source name using the tool, refer to Appendix A.7, "ODBC Administrator and the Oracle Database Lite ODBC Driver".

2.3.3.1.2 Creating DSN on a LINUX System In order to create a DSN on a LINUX platform, add the DSN in the ODBC.INI file. In this file, add the DSN in its own section, where the section name is the DSN name.

The ODBC.INI file is available in Windows under %WINDIR% and in Linux under $OLITE_HOME/bin. For the Linux platform, you must have write permissions on the directory where this is located to be able to modify them.

For example, the following ODBC.INI example contains two DSN configurations:

- The Polite DSN configuration is for a single Oracle Lite database installed on the Mobile client.
- The Politecl DSN configuration describes a multi-user service DSN, as shown with the ServerHostName and ServerPortNumber elements. This service is described further in Section 2.5, "Oracle Database Lite Multi-User Service".

```
[Polite]
Description=Oracle Lite 40 Data Source
Data_Directory=/home/olite
Database=polite
IsolationLevel=Read Committed
Autocommit=Off
CursorType=Forward Only

[Politecl]
Description=Oracle Lite 40 Data Source
Data_Directory=/home/olite
ServerHostName=localhost
ServerPortNumber=1160
Database=polite
IsolationLevel=Read Committed
Autocommit=Off
CursorType=Static
```

The default port number is 1160.

The parameters that you can use are listed in Table 2–1:

Table 2–1 POLITE.INI DSN Parameters

DSN Parameter	Description
`Description`	An optional description for the data source. Use only for Windows environment.
`Data Directory`	The path to the data directory where the database resides. This is an existing path.
`Database`	Oracle Database Lite database name to be created. Do not include the `.ODB` extension.
`Default Isolation Level`	Determines the degree to which operations in different transactions are visible to each other. For more information on the supported isolation levels, refer to Section 15.2, "What Are the Transaction Isolation Levels?" for more information. The default level is `Read Committed`. Other options are `Repeatable Read`, `Single User`, and `Serializable`.
`Autocommit`	Commits every database update operation in a transaction when that operation is performed. Auto-commit values are Off and On. The default value is Off. ▪ On: DML and DDLs are automatically committed. ▪ Off: An application has to explicitly issue the transaction commit or rollback commands. **Note**: In the Microsoft ODBC SDK, the ODBC driver defaults to auto-commit mode. However, the default for Oracle Database Lite is manual-commit mode. In this environment, if you execute SQLEndTrans / SQLTransact call with SQL_COMMIT option using the ODBC driver, you receive a SQL_SUCCESS, because ODBC believes that auto-commit is on. However, no commit actually occurs, because ODBC transfers the transaction to Oracle Database Lite, whose default is manual-commit. You must configure the Microsoft ODBC Driver Manager to transfer control of the SQLEndTrans / SQLTransact API call to Oracle Database Lite by explicitly setting autocommit to OFF in ODBC. When you do this, ODBC does not try to autocommit, but gives control of the transaction to Oracle Database Lite. To set auto-commit to off, execute either the `SQLSetConnectAtrr` or `SQLSetConnectOption` method with `SQL_AUTOCOMMIT_OFF` as the value of the `SQL_AUTOCOMMIT` option. Then, the `SQLEndTrans / SQLTransact` calls will commit as defaulted within Oracle Database Lite. Thus, if you want auto-commit on, turn it on only within Oracle Database Lite.

Table 2–1 (Cont.) POLITE.INI DSN Parameters

DSN Parameter	Description
`Cursor Type`	■ *Forward Only*: Default. A non-scrollable cursor which only moves forward but not backward through the result set. As a result, the cursor cannot go back to previously fetched rows. ■ *Dynamic*: Capable of detecting changes to the membership, order, or values of a result set after the cursor is opened. If a dynamic cursor fetches rows that are subsequently deleted or updated by another application, it detects those changes when it fetches those rows again. ■ *Keyset Driven*: Does not detect change to the membership or order of a result set, but detects changes to the values of rows in the result set. ■ *Static*: Does not detect changes to the membership, order or values of a result set after the cursor is opened. If a static cursor fetches a row that is subsequently updated by another application, it does not detect the changes even if it fetches the row again. See Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types" for details on the restrictions when combining cursor types and isolation levels.

If your DSN connects to a multi-user service—see Section 2.5, "Oracle Database Lite Multi-User Service"—then the DSN entries have the following additional parameters:

Table 2–2 DSN Configuration Parameters for Multi-User Service on LINUX

Parameter	Description
`ServerHostName`	Provide the server machine hostname or IP address where the database service is running.
`ServerPortNumber`	The port number where the database service is listening for incoming requests. The default port number is 1160.
`ServerDSN`	The server-side DSN. Thus, the client DSN name on the client machine can be different from the DSN on the server mahcine. This is required only if the client and server machines are not the same and the `Database Directory` and `Database` parameters are not required.

2.3.3.2 Creating a New Oracle Lite Database for the Embedded Application

To create a new Oracle Lite database, use the `CREATEDB` command-line utility providing the DSN name, the database name, and the system user password, as follows:

```
CREATEDB myDSN myDBname sysPwd
```

For example, if the name of the DSN is `POLITE`, the ODB name is `myDB`, and the system user password is `MANAGER`:

```
CREATEDB polite mydb manager
```

See Section A.2, "CREATEDB" for more information.

The new database file is located in the <*ORACLE_HOME*>`\Mobile\Sdk\oldb40` directory. For ease of maintenance, it is recommended that you use one database directory for all of your Oracle Lite databases.

2.3.3.3 Connecting to Your New Oracle Lite Database

Connect to the file-based Oracle Lite starter database using your application or mSQL, which is a command line interface. See Section A.1, "The mSQL Tool" for full details.

When connecting to the starter database from an ODBC application, use the default ODBC DSN `POLITE`. To connect to the `POLITE` database using mSQL with `SYSTEM` user, `MANAGER` password, and the `mydsn` data source name, perform the following:

```
C:>msql system/manager@jdbc:polite:mydsn
```

Note: On WinCE, the mSQL utility is a GUI installed on your platform.

You can replace `mydsn` with a previously defined ODBC data source name. To connect to the default DSN POLITE, the mSQL statement would be as follows:

```
C:>msql system/manager@jdbc:polite:polite
```

Note: Review the *Oracle Database Lite SQL Reference* before using the starter database to understand the SQL used to manage information in Oracle Database Lite.

2.3.4 Creating Users for Your Embedded Oracle Lite Database

A user is not a schema. When you create a user, Oracle Database Lite creates a schema with the same name and automatically assigns it to that user as the default schema. You can access database objects in the default schema without prefixing them with the schema name.

Users with the appropriate privileges can create additional schemas by using the `CREATE SCHEMA` command, but only the user can connect to the database. You cannot connect to the database using the schema name. These schemas are owned by the user who created them and require the schema name prefix in order to access their objects.

When you create a database using the `CREATEDB` utility or the `CREATE DATABASE` command, Oracle Database Lite creates a special user called `SYSTEM`, which has all database privileges.

Note: For more information on the `CREATEDB` utility, see Section A.2, "CREATEDB".

To access data and perform operations in another user schema, a user must grant you DBA or ADMIN privileges. Alternatively, the user can access data with the user name `SYSTEM`, as this username automatically holds DBA and ADMIN privileges.

You can create multiple users in your Oracle Lite database for your embedded application with the `CREATE USER` command. See the *Oracle Database Lite SQL Reference* for information on how to manage your user through SQL commands. However, if you are using Branch Office, then create the users with the Branch Office Admin Tool, as described in Section 10.4.3, "Managing Branch Office Users" in the *Oracle Database Lite Administration and Deployment Guide*.

Note: Both username and passwords are limited to a maximum of 28 characters.

While most information you need to understand about SQL and your Oracle Lite database can be gathered from the Oracle Database manuals and the *Oracle Database Lite SQL Reference*, the following sections help you understand concepts related specifically to the Oracle Lite database.

- Section 2.3.4.1, "Pre-Defined Roles"
- Section 2.3.4.2, "Building and Populating Demo Tables"

2.3.4.1 Pre-Defined Roles

Oracle Database Lite combines some privileges into pre-defined roles for convenience. In many cases it is easier to grant a user a pre-defined role than to grant specific privileges in another schema. Oracle Database Lite does not support creating or dropping roles. Following is a list of Oracle Database Lite pre-defined roles:

Table 2–3 Pre-Defined Roles

Role Name	Privileges Granted To Role
ADMIN	Enables the user to create other users and grant privileges other than DBA and ADMIN on any object in the schema: CREATE SCHEMA, CREATE USER, ALTER USER, DROP USER, DROP SCHEMA, GRANT, REVOKE
DBA	Enables the user to issue the following DDL statements which otherwise can only be issued by SYSTEM: All ADMIN privileges, CREATE TABLE, CREATE ANY TABLE, CREATE VIEW, CREATE ANY VIEW, CREATE INDEX, CREATE ANY INDEX, ALTER TABLE, ALTER VIEW, DROP TABLE, DROP VIEW, and DROP INDEX.
RESOURCE	The RESOURCE role grants the same level of control as the DBA role, but only over the user's own schema. The user can execute any of the following commands in a SQL statement: CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE CONSTRAINT, ALTER TABLE, ALTER VIEW, ALTER INDEX, ALTER CONSTRAINT, DROP TABLE, DROP VIEW, DROP INDEX, DROP CONSTRAINT, and GRANT or REVOKE privileges on any object under a user's own schema.

General Note: Unlike the Oracle database server, Oracle Database Lite does not commit data definition language (DDL) commands until you explicitly issue the COMMIT command.

2.3.4.2 Building and Populating Demo Tables

Oracle Database Lite comes with a script called POLDEMO.SQL, which enables you to build the same tables that are in your Oracle Lite default starter database (POLITE.ODB).

You can use SQL scripts to create tables and schema, and to insert data into tables. A SQL script is a text file, generally with a .SQL extension, that contains SQL commands. You can run the following SQL script from the Mobile SQL prompt.

```
SQL> @<ORACLE_HOME>Mobile\DBS\Poldemo.sql
```

You can also enter:

```
SQL> START Poldemo.sql
```

Note: You do not need to include the `.SQL` file extension when running the script.

2.3.5 Packaging Your Embedded Application With the Oracle Database Lite Runtime

In order to use the Oracle Lite database and embed it into your application, you must include not only the Oracle Lite database, but certain DLLs in your application. Perform the following:

1. Copy the following files from the Mobile Development Kit library, which is located in *`ORACLE_HOME`*`/Mobile/Sdk`, into the directory in your PATH where your application DLLs are located.
   - `olite40.msb`: Oracle Database Lite message file
   - `oljdbc40.dll`: JDBC JNI library
   - `olobj40.dll`: Oracle Database Lite object kernel
   - `olod2040.dll`: Oracle Database Lite ODBC driver
   - `olsql40.dll`: Oracle Database Lite SQL runtime library
   - `olstddll.dll`: Oracle Lite Common library
2. If you are using the Multi-User Service, copy `olsv2040.exe` and `olsvmsg.dll` into your PATH where your application DLLs are located.
3. To use any Java program with Oracle Database Lite, make sure that the `olite40.jar` file, which is installed in `OLITE_HOME/bin`, is in the application `CLASSPATH`. If the Java program uses the multi-user service, also place this JAR file in the SYSTEM `CLASSPATH`. This JAR file contains the JDBC driver for Oracle Database Lite. Your environment must provide a Java Runtime Environment from Sun, version JDK 1.4.2 version or higher.
4. If you want to support the mSQL command-line tool for querying and managing the Oracle Lite database, then you must place the following files in the `PATH`:
   - `msql.dll`
   - `msql.exe`
   - `msql.jar`
5. Manage ODBC for creating the DSN and registering the ODBC driver. On Linux, modify the `ODBC.INI` file. On Windows, perform the following:

   a. To use Microsoft ODBC for the ODBC environment—including DSN creation support—or to create and manage DSN names programmatically, place the `olad2040.dll` in the `PATH`. This DLL provides a plug-in to programmatically access the ODBC administration tool—`odbcad32`—that is used to create DSNs.

   b. Register the ODBC driver for the product in the Windows Registry, as follows:

```
KEY:HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI\Oracle Lite 40 ODBC
Driver
VALUE:32Bit = 1
VALUE:ApiLevel = 0
VALUE:ConnectFunctions = YYN
VALUE:Driver = <path to olod2040.dll>
VALUE:DriverODBCVer = 02.00
VALUE:SQLLevel = 0
```

```
VALUE:Setup = <path_to_olad2040.dll>
KEY:HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI\ODBC DRIVERS
VALUE:Oracle Lite 40 ODBC Driver = Installed
```

6. Configure the `POLITE.INI` file and place it in the system Windows directory, such as `c:\winnt`, as follows:

```
[All Databases]
NLS_LANGUAGE=ENGLISH
NLS_LOCALE=ENGLISH
DB_CHAR_ENCODING=Native
DATA_DIRECTORY=<default_directory_to_create_database_files>
```

Note: See Appendix G, "`POLITE.INI` Parameters" in the *Oracle Database Lite Administration and Deployment Guide* for more information on how to configure the `POLITE.INI` file.

2.4 Data Access APIs

To access the data within the ODB file from your application through one of the following APIs:

- For relational database development:
  - JDBC—See Section 2.4.1, "JDBC" for more information.
  - ODBC—See Section 2.4.2, "ODBC" for more information.
  - ADO.NET—See Section 2.4.3, "ADO.NET" for more information.

    Any interface that supports ODBC or JDBC data sources, such as ADO.Net, can also be used to access Oracle Database Lite. The interfaces can be used either independently or in combination.
- For object and relational database development:
  - Simple Object Data Access (SODA)—See Section 2.4.4, "SODA" for more information.

The following sections describe the different development interfaces that you can use to store and retrieve data from the file-based Oracle Lite database:

- Section 2.4.1, "JDBC"
- Section 2.4.2, "ODBC"
- Section 2.4.3, "ADO.NET"
- Section 2.4.4, "SODA"

2.4.1 JDBC

The Java Database Connectivity (JDBC) interface specifies a set of Java classes that provide an ODBC-like interface to SQL databases for Java applications. JDBC, part of the JDK core, provides an object interface to relational databases. Oracle Database Lite conforms to the JDBC 1.2 API specification standard.

Oracle Database Lite supports JDBC through an Oracle Database Lite Type 2 and Type 4 JDBC drivers that interpret the JDBC calls and pass them to Oracle Database Lite. The Type 4 JDBC driver can only be used for the multi-user service only, as described in Section 2.5, "Oracle Database Lite Multi-User Service".

For Mobile clients, all JDBC drivers are provided for you to use within the Oracle Database Lite binaries. However, for embedded applications, you must include the correct binaries when you package the application, as described in Section 2.3.5, "Packaging Your Embedded Application With the Oracle Database Lite Runtime".

See Chapter 10, "JDBC Programming" for more information on using JDBC.

2.4.2 ODBC

The Microsoft Open Database Connectivity (ODBC) interface is a procedural, call-level interface for accessing any SQL database, and is supported by most database vendors. It specifies a set of functions that allow applications to connect to the database, prepare and execute SQL statements at runtime, and retrieve query results.

Oracle Database Lite supports Level 3 compliant ODBC 2.0 and the ODBC 3.5 drivers through Oracle Database Lite ODBC drivers.

For more information on ODBC, see the following:

- Microsoft ODBC documentation.
- The Oracle Database Lite ODBC sample application, as described in Section 2.11, "Using Oracle Database Lite Samples".
- Section 11.4.2.1, "Returning Multiple Rows in ODBC".

2.4.3 ADO.NET

The Oracle Database Lite ADO.NET Provider implements the Microsoft ADO.NET specification. Use this programming interface to access Oracle Database Lite and trigger data synchronization in .NET applications. The Oracle Database Lite ADO.NET data provider supports both .NET and Compact .NET frameworks.

See Chapter 13, "Oracle Database Lite ADO.NET Provider" for a full description.

2.4.4 SODA

SODA is an interface for Oracle Database Lite development using C++. It provides object-oriented data access using method calls, relational access using SQL and object-relational mapping to bridge the gap between the two.

Object functionality is about three times faster than ODBC for simple operations. It allows rich datatypes—such as arrays, object pointers, and standard SQL columns. A programmer can store any data structure in the database and not worry about relational design or performing joins.

A C++ developer can use the interface for executing SQL statements. The resulting code is shorter and clearer than ODBC code. SQL queries can return objects, which can be examined and modified directly through the object-oriented layer without calling any additional SQL statements.

Finally, object-relational mapping enables the application to access relational data as if it was an object hierarchy. This is essential for replicating rich data types or object pointers to the Oracle database server.

For more information, see Chapter 12, "Using Simple Object Data Access (SODA) for PocketPC Platforms".

2.5 Oracle Database Lite Multi-User Service

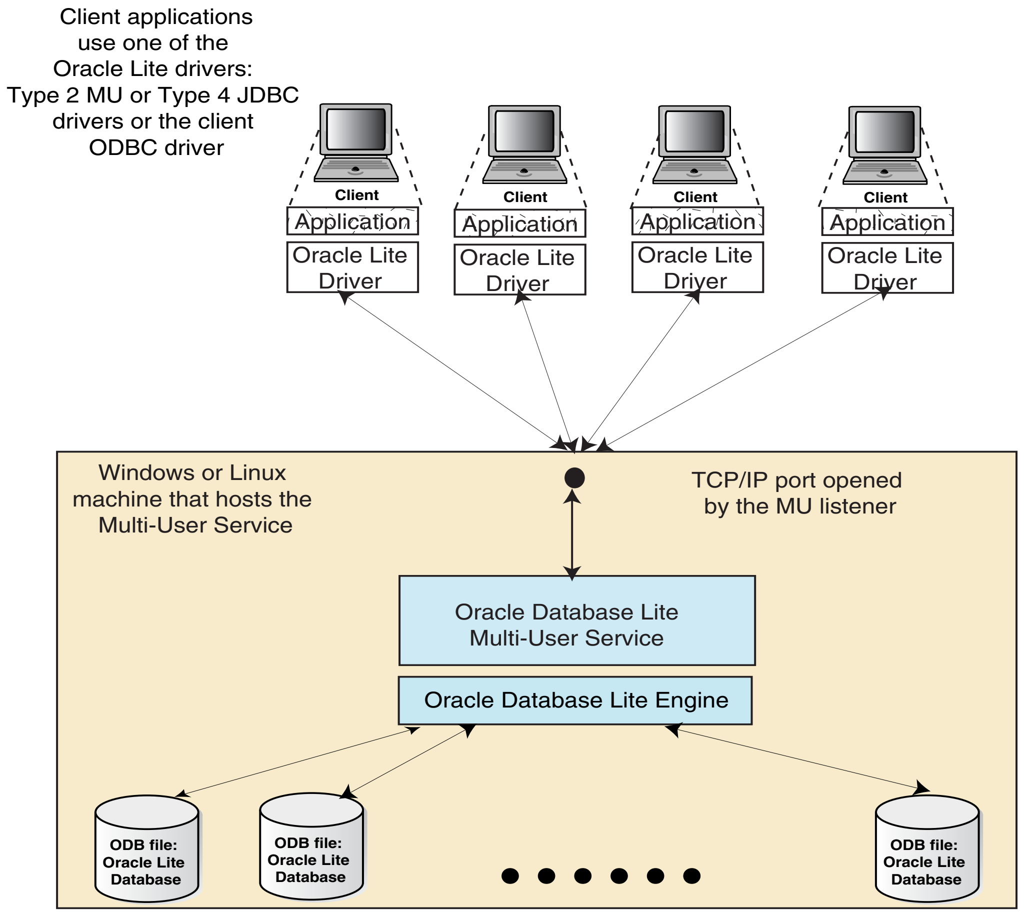
If you want to have multiple users accessing a single entry point for the Oracle Lite database, then use one of the following multi-user services:

- For multiple clients accessing a single Mobile client that synchronizes with a Mobile Server, use the Branch Office service. See Chapter 9, "Manage Your Branch Office" in the *Oracle Database Lite Administration and Deployment Guide* for full details.
- For multiple clients executing an application that accesses the same database, set up a listener to receive requests from each of these clients. See Section 2.5.1, "Accessing the Multi-User Oracle Database Lite Database Service" for full details.

2.5.1 Accessing the Multi-User Oracle Database Lite Database Service

When you are using the embedded application approach, there may be a time when you want to protect all of your data on a centralized machine and only allowing the clients to access the information remotely. Figure 2–1 demonstrates centralizing your Oracle Lite databases (ODB files) by installing them on a Windows or Linux host machine. The Oracle Database Lite Multi-User Service facilitates the communication between the remote application clients by starting the multi-user service, which opens the designated ports, and then translates the DSNs to the appropriate database.

Figure 2–1 Diagram of Multi-User Service


Client applications use one of the Oracle Lite drivers: Type 2 MU or Type 4 JDBC drivers or the client ODBC driver
Client
Application
Oracle Lite Driver
Client
Application
Oracle Lite Driver
Client
Application
Oracle Lite Driver
Client
Application
Oracle Lite Driver
Windows or Linux machine that hosts the Multi-User Service
TCP/IP port opened by the MU listener
Oracle Database Lite Multi-User Service
Oracle Database Lite Engine
ODB file: Oracle Lite Database
ODB file: Oracle Lite Database
ODB file: Oracle Lite Database

The Multi-User Service enables you to use up to sixty-four concurrent client connections, each of which can connect to up to five `ODB` database files on the remote machine. All clients and server must install the same binary with the same NLS. The server machine with the Multi-User Service and each of the clients can be installed on either Windows or Linux platforms.

When you implement the remote data access for your embedded application, the flow of events is as follows:

1. Remote client sends the connect request to the multi-user service.

   The client connection can use the ODBC 2.0 client driver, the JDBC Type 2 MU driver, or the JDBC Type 4 driver. In addition, the client provides the following in the connection string: remote host and port where the multi-user service is listening for incoming calls and the DSN of the Oracle Lite database (ODB file) where the data is stored. The default port number is 1160.
2. Multi-user service receives incoming call with the DSN from the remote client.
3. The multi-user service parses the DSN name and completes the request by connecting to the Oracle Lite database that maps to the DSN name.

The following sections describe how to set up the multi-user Oracle Database Lite database service:

- Section 2.5.1.1, "Administration for the Multi-User Service on the Windows Platform"
- Section 2.5.1.2, "Administration for the Multi-User Service on the Linux Platform"
- Section 2.5.1.3, "Debugging the Multi-User Service"
- Section 2.5.1.4, "Creating DSNs"
- Section 2.5.1.5, "Accessing the Database"
- Section 2.5.1.6, "Verifying the Connection Using mSQL"

2.5.1.1 Administration for the Multi-User Service on the Windows Platform

The following sections describe how to install, start, stop and query the multi-user service on Windows:

- Section 2.5.1.1.1, "Installation and Configuration on Windows"
- Section 2.5.1.1.2, "Starting the Multi-User Service on Windows"
- Section 2.5.1.1.3, "Stopping the Multi-User Service on Windows"
- Section 2.5.1.1.4, "Querying the Multi-User Service on Windows"

2.5.1.1.1 Installation and Configuration on Windows To install and configure the Oracle Database Lite multi-user service, perform the following steps:

1. Ensure that you install the `olsv2040.exe` in the following directory.

   ```
   <OLITE_HOME>\Mobile\Sdk\bin
   ```

   Note: This directory must also be included in your system `PATH`.

   If not already available, re-install the MDK to retrieve the component. A sample *<OLITE_HOME>* location is `C:\Olite`.

2. To install the service, start the Command Prompt and enter the following command.

   ```
   olsv2040.exe /install [/account=AccountName][/password=ValidPassword]
       [/wdir=WorkingDirectory] [/port=ServicePort]
   ```

   where the optional parameters can be as follows:

   - `AccountName`: Provide either the `DomainName\UserName` or `.\UserName`.
   - `ValidPassword`: If you specify an account, but don't want to give out the password in command prompt during service installation. You can provide the password to the "Log On" page of "Oracle Lite Multiuser Service Properties" dialog box which can be found in Services.
   - `WorkingDirectory`: If you use `'.'` in SQL scripts that load Java classes, you must specify a working directory.
   - `ServicePort`: The default port number is 1160.

3. If you are using Java Stored Procedures, then perform the following to set up the environment for Java Stored Procedures:

   a. If you have JDK, which should be a minimum of version 1.4.2, installed on your PC, ensure that the system `PATH` variable includes the following:

   ```
   <JDK_HOME>\bin
   <JDK_HOME>\jre\bin
   <JDK_HOME>\jre\bin\hotspot
   ```

   For example, the `<JDK_HOME>` directory could be `C:\jdk1.4.2`.

   b. If you have *JRE*, which should be a minimum of version 1.4.2, installed on your PC, ensure that the system `PATH` variable includes the following:

   ```
   <JRE_HOME>\bin
   <JRE_HOME>\bin\hotspot
   ```

   For example, the `<JRE_HOME>` directory could be `C:\Program Files\JavaSoft\JRE\1.4.2`

   Note: *JRE* does not include the Java compiler. Therefore, other attempts to load a Java source into the database such as the `CREATE JAVA SOURCE` command and the `loadjava` utility will fail.

   c. Ensure that your system `CLASSPATH` variable includes the following:

   ```
   <OLITE_HOME>\bin\Olite40.jar and '.'
   ```

4. You may change the startup type from the Windows NT service console. Highlight the Oracle Database Lite Multi-User Service and select **Properties**. When required, change the startup type to manual. The property also contains startup parameters, but has not been tested.

5. Reboot your PC.

2.5.1.1.2 Starting the Multi-User Service on Windows The Oracle Database Lite Multi-User Service can be started in many ways. By default, the service property "Startup Type" is automatic; thus, the service is started every time you reboot the machine. If you modify the "Startup Type" to "Manual", then you start Oracle Lite multi-user service by entering any one of the following startup commands from the Command Prompt:

- `olsv2040.exe /start`
- `net start "Oracle Lite Multiuser Service"`

2.5.1.1.3 Stopping the Multi-User Service on Windows To stop the multi-user service, use one of the following commands:

- `olsv2040.exe /stop`
- `net stop "Oracle Lite Multiuser Service"`

2.5.1.1.4 Querying the Multi-User Service on Windows You can query the multi-user status for the following details:

- current status
- current startup parameters
- configuration
- installed startup parameters

Issue the following command to see these details:

```
olsv2040.exe /query
```

The results of this command are as follows:

```
OliteService reports the following status:
  The service is running...
     port= 1160
     wdir = C:\WINDOWS\SYSTEM32

The current status of Oracle Lite Multiuser Service:
  Current State              : SERVICE_RUNNING
  Acceptable Control Code    : (0x1) SERVICE_ACCEPT_STOP

The configuration of Oracle Lite Multiuser Service:
  Service Type    : (0x10) SERVICE_WIN32_OWN_PROCESS
  Start Type      : SERVICE_AUTO_START
  Error Control   : SERVICE_ERROR_NORMAL
  Binary Path     : C:\Oracle\product\Mobile\Sdk\bin\olsv2040.exe
  Display Name    : Oracle Lite Multiuser Service
  Start Name      : LocalSystem

Installed Service startup parameters
  port = 1160
  wdir = \%WINDIR%\SYSTEM32
```

where the port number is 1160 and the working directory is `C:\WINDOWS\SYSTEM32`. The service is installed under the `LocalSystem` account, where the startup parameters for installation are port = 1160 and working directory = `\%WINDIR%\SYSTEM32`. In addition, the Start Type is `SERVICE_AUTO_START` and the binary path as `C:\Oracle\product\Mobile\Sdk\bin\olsv2040.exe`, which is where you installed the Oracle Lite Multi-User Service.

You can also set the default for the service port and working directory by modifying the `SERVICE_PORT` and `SERVICE_WDIR` parameters in the `polite.ini` file. However, if you do so, then it overrides any of the command-line options for port and working directory.

Note: When you execute the Multi-User Service with the `/debug` option, then the result of the current status from the `/query` shows that the service is stopped. Since the `/debug` option is executed in a console, the Service Control Manager does not know that the service is running.

2.5.1.2 Administration for the Multi-User Service on the Linux Platform

The following sections describe how to start, stop and query the multi-user service on Linux:

Note: There is no need to install this service on Linux.

- Section 2.5.1.2.1, "Starting and Stopping the Multi-User Service on Linux"
- Section 2.5.1.2.2, "Querying the Multi-User Service on Linux"

2.5.1.2.1 Starting and Stopping the Multi-User Service on Linux The Oracle Database Lite Multi-User Service can be started or stopped with the `olsv` executable. To see all of the options, execute `olsv -help`, which displays the following:

```
$ olsv -h
Usage: olsv [option]
Options are:
-start start the server as daemon
-stop | -s stop the server
-debug | -d start the server as console app for debugging
-query | -q get the server status
-kill | -k equivalent to kill -s SIGKILL PID
-port | -p PORT execute the server on the specified port—default port is 10000
-wdir | -w DIR run/debug the server in the specified dir
-help | -h display this message
if no option specified, -start by default
```

To stop the multi-user service, use the following command:

```
olsv -stop
```

2.5.1.2.2 Querying the Multi-User Service on Linux You can query the multi-user status with `olsv -query` which provides the following information:

- process id (PID)
- environment variables
- current status

Issue the following command to see these details:

```
olsv -query
```

The results of this command are as follows:

```
Oracle Lite Multiuser Server is running, PID = 2619
----------------------------------------------------------------------
Environment:
TERM=xterm
SHELL=/bin/csh
OLITE_HOME=/scratch/myuser/oracle/OraHome/mobile/sdk
```

```
JDKDIR=/usr/local/packages/jdk14
MOBILE_CLIENT=/scratch/myuser/mobileclient
USER=myuser
LD_LIBRARY_
PATH=/usr/local/packages/jdk14/jre/lib/i386:/usr/local/packages/jdk14/jre/lib/i386
/server:/scratch/myuser/olite/lib
HOSTTYPE=i386-linux
JAVA_HOME=/usr/local/packages/jdk14
LANG=en_US.UTF-8
HOME=/home/myuser
OSTYPE=linux
LOCAL_PACK=/usr/local/packages/icc_remote
JAVA13_HOME=/usr/local/packages/jdk14
VENDOR=intel
MACHTYPE=i386
ORACLE_HOME=/scratch/user/oracle/OraHome
CLASSPATH=.:/scratch
----------------------------------------------------------------------
Status Summary:
Database Version: 10.3.0
Time Started: 11-27-2006 17:58:15
Listening Port: 10000
Total Connections: 0
Current Connections: 0
No errors encountered
```

You can also set the default for the service port and working directory by modifying the `SERVICE_PORT` and `SERVICE_WDIR` parameters in the `polite.ini` file. However, if you do so, then it overrides any of the command-line options for port and working directory.

2.5.1.3 Debugging the Multi-User Service

If the service the does not start, debug the service using the following method:

1. Edit the `POLITE.INI` file, which is available in Windows under `%WINDIR%\POLITE.INI` and in Linux under `$OLITE_HOME/bin`, to add the following entries in the `[ALL_DATABASES]` section:
   - `OLITE_SERVER_TRACE=TRUE`
   - `OLITE_SERVER_LOG=<filename>`. This is used for the LINUX platform only.
2. Should the service fail, the Multi-User Service generates the `olsv.log` file in the current working directory. Ensure that the `PATH` and `CLASSPATH` variables are accurate and that the `PATH` includes the directory that contains `jvm.dll`.
3. Correct the cause and retry.

2.5.1.4 Creating DSNs

To access the database using an ODBC or VB application, you must create the DSN enabled from the embedded connection. When you add a DSN using the ODBC Administration tool, choose the **Oracle Lite 40 ODBC Driver(Client)**, which creates a client DSN. If you are executing the service on the same machine where the client application is running, leave the Database Host Name, Database Port Number, and Database Host DSN value empty. The remaining values must be included in the same manner as the 'Oracle Lite ODBC Driver' DSN. If you start the service on a port other than 1160, then you must specify the Database Port Number.

2.5.1.5 Accessing the Database

To access the database, you need not make any changes to the ODBC or VisualBasic application. The DSN automatically routes the request to the service through the ODBC driver `olcl2040.dll`. For a JDBC application, change the URL for the connect string, which is similar to the one used while connecting to the database using mSQL.

2.5.1.6 Verifying the Connection Using mSQL

Using the Command Prompt, verify the connection to the multi-user service in the following ways:

Connect to `A-DSN` on a Windows local host on port 1160.

```
msql system/passwd@jdbc:polite@::a-dsn
```

Connect to `A-DSN` on the local host on port 1000.

```
msql system/passwd@jdbc:polite@:1000:a-dsn
```

Connect to `A-DSN` on a Windows local host on port 1160 using the Type4 JDBC driver.

```
msql system/passwd@jdbc:polite4@::a-dsn
```

Note: Oracle Database Lite supports Type2 and Type4 JDBC drivers. Type4 is a pure Java JDBC driver that communicates with the service in the Oracle Database Lite network protocol.

For more information on JDBC and Oracle Database Lite, see Chapter 10, "JDBC Programming". For details on mSQL, see Section A.1, "The mSQL Tool".

2.6 Move Your Client Data Between an Oracle Lite Database and an External File

You can move data between an Oracle Lite database and an external file either through programmatic APIs or the Load Utility (`OLLOAD`). The following sections describe both methods:

- Section 2.6.1, "Move Data Between an Oracle Lite Database and an External File Using Programmatic APIs"
- Section 2.6.2, "Oracle Database Lite Load Utility (OLLOAD)"

2.6.1 Move Data Between an Oracle Lite Database and an External File Using Programmatic APIs

Using the Oracle Database Lite Load APIs, you can develop applications to load data from an external file into a table in Oracle Database Lite, or to unload (dump) data from a table in Oracle Database Lite to an external file. The details of the APIs and file formats are provided in Appendix A.10.2, "Oracle Database Lite Load Application Programming Interfaces (APIs)".

2.6.2 Oracle Database Lite Load Utility (OLLOAD)

The Oracle Database Lite Load Utility enables you to load data from an external file into a table in Oracle Database Lite, or to unload (dump) data from a table in Oracle Database Lite to an external file. For more information on the `OLLOAD` utility, see Appendix A.10.1, "OLLOAD".

2.7 Backing Up an Oracle Lite Database

For either the Mobile client or embedded solutions, you can back up the Oracle Lite database either by using the `backupdb` utility or by copying the files to another location.

Oracle Database Lite uses the ODB and OBS files with dependent log files that can be backed up by copying to another location. Before any files can be copied, disconnect all applications that access the database and shut down the multi-user service, if running. Once that has been accomplished, execute the backupdb utility, which copies the `*.odb`, `*.obs`, `*.opw`, and `*.plg` files to the filename of your choice to make a backup of the database.

```
backupdb DSN|NONE DBName backup_filename
```

For full details, see Section A.6, "BACKUPDB".

2.8 Encrypting a Database

For either the Mobile client of embedded solutions, you can encrypt the Oracle Lite database. Once encrypted, the data stored in the database files cannot be interpreted by examining the files. A password is used to derive a 128-bit encryption key. Oracle Database Lite uses the Advanced Encryption Standard (AES) encryption.

If you do not want to use AES encryption, then you can insert your own encryption module to supplant AES; see Section 16.2, "Providing Your Own Encryption Module for the Client Oracle Lite Database" for complete details.

For information on encrypting the database used by the Mobile client, see the `ENCRYPT_DB` parameter in Appendix G, "`POLITE.INI` Parameters" in the *Oracle Database Lite Administration and Deployment Guide*; for information on encrypting the database used by an embedded application, see Section A.6, "BACKUPDB".

2.9 Discover Oracle Lite Database Version Number

Use the ODBINFO utility to discover the version number and volume identifier of the Oracle Lite database. See Section A.7, "ODBC Administrator and the Oracle Database Lite ODBC Driver" for full details.

2.10 Support for Linguistic Sort

Linguistic sort is a feature for the ASCII version of the Oracle Lite database. It produces culturally acceptable order of strings for a specified language or collation sequence. The ASCII version supports several code pages defined by single-byte 8-bit encoding schemes. Each of these code pages is a super set of 7-bit ASCII, and the additional accented characters necessary to support certain European languages are included in the upper 128 bytes.

A new string comparison mechanism is provided that produces strings in a linguistically correct order by mapping each collation element of a string to the corresponding 8-bit value of the supported code page.

The only supported languages for linguistic sort are French, German, Czech and XCzech. The collation sequence for these Oracle Lite databases are defined when you specify the language in downloading the Mobile client in using the setup.exe. You can also specify this with the `NLS_SORT` parameter.

All other languages use the `BINARY` collation sequence, which does not enable linguistic sort.

2.10.1 Creating Linguistic Sort Enabled Databases

The linguistic sort capability must be enabled when the database is created using the `CREATEDB` command line utility with the `<collation_sequence>` enabled.

Note: For more information on the `CREATEDB` utility, see Section A.2, "CREATEDB".

The behavior of the `ORDER_BY` clause and the `WHERE` condition are determined by how the `NLS_SORT` parameter is implemented. Binary sorting is the default setting, and is used unless the `<collation_sequence>` parameter is set to use the linguistic sort ordering rules.

`NLSRT` is not supported in the current version of Oracle Database Lite. Therefore, `NCHAR` data type is not yet available.

2.10.2 How Collation Works

Collation refers to ordering of strings into a culturally acceptable sequence. A collation sequence is a sequence of all collation elements from an alphabet from smallest collation order to the largest. Once a collation sequence is given, orders of all strings from the same alphabet are fixed. As such, the collation sequence encodes the linguistic requirements on collation. A collation element is the smallest sub-string that can be used by the comparison function to determine the order of two strings.

2.10.3 Collation Element Examples

Normally, a collation element is just one character. In binary sorting, only one property, the code value that represents a character, is used. But in linguistic sorting, usually three properties. The primary level of difference is the base character. The secondary level of difference is for diacritical marks on a given base character. The tertiary level of difference is for the case of a given character. Punctuation can function as a fourth level of difference, but comparisons for punctuation occur last and are made at the binary rather than the linguistic level. These are used for each collation element. The following sections contain examples that demonstrate sorting priorities.

2.10.3.1 Sorting Normal Characters

This section lists a set of examples that describe how to sort normal characters.

Example 1

`'a' < 'b'`. There is a primary difference between them on the character level.

Example 2

`'À' > 'a'`. This difference occurs on the secondary level. Note that `'À'` and 'a' are considered "equal" on the primary level.

Example 3

`'À' < 'à'` in FRENCH but 'À' > 'à' in GERMAN. This difference on the tertiary level. Note that 'À' and 'à' are considered being "equal" on the primary and secondary level. Also note that the case convention may be different for different language.

Example 4

'às' < 'at'. This is a difference on the primary level. This example shows the role of difference levels: the lower level differences are ignored if there is a primary level difference anywhere in the strings.

Example 5

`'+data' < '-data' <'data' <'data-'`. If strings are compared and present no difference on the primary, secondary, or tertiary levels, they are compared for punctuation.

2.10.3.2 Reverse Sorting of French Accents

Some languages, particularly French, require words to be ordered on the secondary level according to the last accent difference. This behavior is known as French secondary sorting or French accent ordering.

Example

'côte' < 'coté' in FRENCH but 'coté' < 'côte' in GERMAN. Note that the secondary difference of 'e' and 'é' occurred later than those of 'ô' and 'o'.

2.10.3.3 Sorting Contracting Characters

There are some special cases where two or more characters in a group can function as a single collation element. These types of collation elements are called 'contracting characters' or 'group characters'. In these cases each of these characters properties are assigned appropriate values.

Example

'h' < 'ch' < 'i' in XCZECH. Here 'ch' is assigned a primary property value which differentiates it from 'h' and 'i', such that 'h' < 'ch' < 'i'. Note that 'ch' is treated as a single character.

2.10.3.4 Sorting Expanding Characters

If a letter sorts as if it were a sequence of more than one letter, it is called an 'expanding character'. For example, in German the sharp s (ß) is treated as if it were a string of two characters 'ss' when comparing with other letters.

2.10.3.5 Sorting Numeric Characters

Only sorting of single digit characters from '0' to '9' is currently supported. For the supported European languages a digit character is always sorted as greater than any alphabetic character. For other languages this may be not the same. Other numeric characters such as Roman numeric characters and counting sequences, such as "one", "two", "three", are not supported at this time.

Example

'1' > 'z' in any European language, '1' < 'a' in LATVIAN. Note that this difference occurs on the primary level.

2.11 Using Oracle Database Lite Samples

After you perform a complete installation of Oracle Database Lite, the samples are available in your `<`*`ORACLE_HOME`*`>\Mobile\Sdk` directory. The tools, locations for samples, and descriptions are listed in Table 2–4.

Table 2–4 Sample File Directory

Tool	Location of Sample Applications	Description
Java	`<ORACLE_HOME>\Mobile\Sdk\samples\jdbc`	Demonstrates programming with JDBC. See Chapter 10, "JDBC Programming" for more information.
ODBC	`<ORACLE_HOME>\Mobile\Sdk\samples\odbc\win32\c_samples`	Provides ODBC programs written in C.
Visual Basic	`<ORACLE_HOME>\Mobile\Sdk\samples\odbc\win32\update`	Demonstrates the ease of querying tables in Oracle Database Lite with Visual Basic tools. See Section 2.11.1, "Executing the Visual Basic Sample Application" for more information.
MFS	`<ORACLE_HOME>\Mobile\Sdk\samples\odbc\win32\mfs`	The MFS sample was documented as a tutorial. See the MFS example used in Chapter 19, "Tutorial for Building Mobile Applications for Win32".

Note: Most examples use the data source name (DSN) `POLITE`. If you need to drop and recreate, use the `REMOVEDB` and `CREATEDB` utilities, which are documented in Section A.2, "CREATEDB" and Section A.3, "REMOVEDB".

The following sections provide instructions on how to use Oracle Database Lite samples.

- Section 2.11.1, "Executing the Visual Basic Sample Application"
- Section 2.11.2, "Executing the ODBC Examples"

2.11.1 Executing the Visual Basic Sample Application

The Visual Basic Sample application example uses Visual Basic 5.0 or higher and demonstrates how to develop a Visual Basic application with Oracle Database Lite. It uses the ODBC DSN, `POLITE`. To use the `AddNew`, `Update`, and `Delete` macros, you need a unique `EMPNO` column of the `EMP` table. This is the default condition when you connect to the default database.

The following instructions for installing and running the Visual Basic sample application assume that you have already installed Oracle Database Lite and Visual Basic.

1. Section 2.11.1.1, "Open Visual Basic"
2. Section 2.11.1.2, "View the Sample Application Tables and Data"
3. Section 2.11.1.3, "Open the Sample Application"
4. Section 2.11.1.4, "View and Manipulate the Data in the EMP Table"

2.11.1.1 Open Visual Basic

Double-click the Visual Basic icon in your Visual Basic program group to open Visual Basic.

2.11.1.2 View the Sample Application Tables and Data

Use the Visual Data Manager, which is available only with Visual Basic 5.0. If you are using an earlier version of Visual Basic, then skip to Step 3.

1. From the Add-Ins menu, select **Visual Data Manager**.
2. In the VisData window, select **Open Database** from the File menu.
3. Select **ODBC**.
4. In the ODBC Logon dialog, enter values as described in Table 2–5.

Table 2–5 ODBC Logon Dialog Description

Field Name	Value
DSN	POLITE
UID	SYSTEM
PW	Enter at least one character
Database	POLITE

5. Click **OK**. The Oracle Database Lite tables are displayed in the database window. You can highlight a table and right click to open the table and display the records.

2.11.1.3 Open the Sample Application

1. To open the sample application, select **Open Project** from the File menu.
2. In the dialog box, navigate to the <*ORACLE_HOME*>`\Mobile\Sdk\samples\odbc\win32\update` directory.
3. Select `update.vbproj`, and click **Open**.
4. Follow the instructions in `readMe.txt` in the same location to execute the sample.

2.11.1.4 View and Manipulate the Data in the EMP Table

1. To view data in the `EMP` table:
   - Click **Show** to show the `EMP` table data.
   - Click **Next** to show the next record.
   - Click **Previous** to show the previous record.
2. To manipulate data in the `EMP` table, use the Add, Update, and Delete features.

2.11.2 Executing the ODBC Examples

The ODBC examples are located in <*ORACLE_HOME*>`\Mobile\Sdk\Samples` and must be compiled using a C++ complier. To build them, use `nmake`.

There are five ODBC examples: `odbctbl`, `odbcview`, `odbcfunc`, `odbctype`, and `long`. You use the `POLITE` DSN to execute these examples. The `POLITE` DSN is automatically created during the Mobile Development Kit installation.

The first four examples have their own output windows listing the activity log. Closing the current example window causes the next example to be run. The output displayed in the example windows is also printed in the following log files: `odbctbl.log`, `odbcview.log`, `odbcfunc.log`, `odbctype.log`. The `long` example output is collected in the output file: `long.out`.

The following sections describe the functionality of the samples:

- Section 2.11.2.1, "ODBCTBL"
- Section 2.11.2.2, "ODBCVIEW"
- Section 2.11.2.3, "ODBCFUNC"
- Section 2.11.2.4, "ODBCTYPE"
- Section 2.11.2.5, "LONG"

2.11.2.1 ODBCTBL

This is an ODBC SQL table example, which shows how to manipulate tables using the ODBC API. It creates the `EMP` table with columns ID, `NAME`, `START_DATE`, `SALARY`. After creation, it populates this table with data, performs an update on the salary column, selectively deletes some rows, then selects from the resulting table and shows the results of the fetch operation. At the end, the `EMP` table is dropped.

2.11.2.2 ODBCVIEW

This is an ODBC SQL view example, which demonstrates how to manipulate views using the ODBC API. It creates the `EMP` table and the `HIGH_PAID_EMP` view, selecting the full name (using the `CONCAT` scalar function), `HIRE_DATE` and `SALARY` from the `EMP` table. Then, the example populates the `EMP` table and selects from the `HIGH_PAID_EMP` view to show the populated data. The salary column of `EMP` is updated, some rows are delete, and a select from `HIGH_PAID_EMP` is issued to demonstrate how the changes are reflected in the view. Finally, the view and the table are dropped.

2.11.2.3 ODBCFUNC

This is an ODBC SQL scalar functions example, which shows you how to use scalar functions in the ODBC API. It creates table `EMP`, populates it with the data, then performs a select on `ID`, `FULL_NAME` from `EMP`. When it calculates the full name, it uses the ODBC scalar function `CONCAT`—with last and first names as arguments. The example updates the table, converting the last name to uppercase and first name to lowercase for IDs less than three using ODBC scalar functions `UCASE` and `LCASE`. The new data is selected and displayed again. Finally, the table `EMP` is dropped.

2.11.2.4 ODBCTYPE

This is ODBC SQL types example, which shows you how to manipulate different data types using the ODBC API. This test creates the `EMP` table, populates it with data, selects all the rows and displays the result. However, the columns are bound differently from the previous tests. First, it calls `SQLNumResultCols` to find the number of result columns. Then, for each result column, it calls `SQLDescribeCol` to retrieve all of the information about that column, such as column name, column name length, column type, column length, column scale, and so on. This information is used to bind the column. Thus, you can see how you can retrieve the type information from the database using the ODBC API.

2.11.2.5 LONG

This example exercises the basic read/write functions of `SQL LONG VARCHAR`. It first drops, then creates the `LONG_DATA` table with one `LONG VARCHAR` column and inserts the data into the table. For each row the data is put in frames, where each frame represents a buffer of long varchar data (of length 4096). The example uses `SQLParamData` and `SQLPutData` to send each frame to populate the row. Then, issues a select to fetch the rows and read long varchar data from the table. For each row, the data is also read in frames, using `SQLGetData` until `SQL_NO_DATA_FOUND` is returned. These actions are logged into the `long.out` file.

2.12 Limitations of the Oracle Database Lite Engine

Currently, the Oracle Database Lite engine cannot sort any row that exceeds 4040 bytes in length. If the selected columns exceed this length, then the database engine issues an error. Therefore, you cannot recover queries that use the `UNION` operation where both select clauses sort intermediate results, where the returned results are long rows with size greater than 4040 bytes.

3 Synchronization

The Oracle Lite database contains a subset of data stored in the Oracle database. This subset is stored in snapshots in the Oracle Lite database. Unlike a base table, a snapshot keeps track of changes made to it in a change log. Users can make changes in the Oracle Lite database and can synchronize them with the Oracle database.

The following sections describe how synchronization functions between Oracle Database Lite and an Oracle database using the Mobile Server. This chapter discusses how you can programmatically initiate the synchronization both from the client or the server side.

- Section 3.22, "Troubleshooting Synchronization Errors"
- Section 3.23, "Datatype Conversion Between the Oracle Server and Client Oracle Lite Database"

3.1 How Does Synchronization Work?

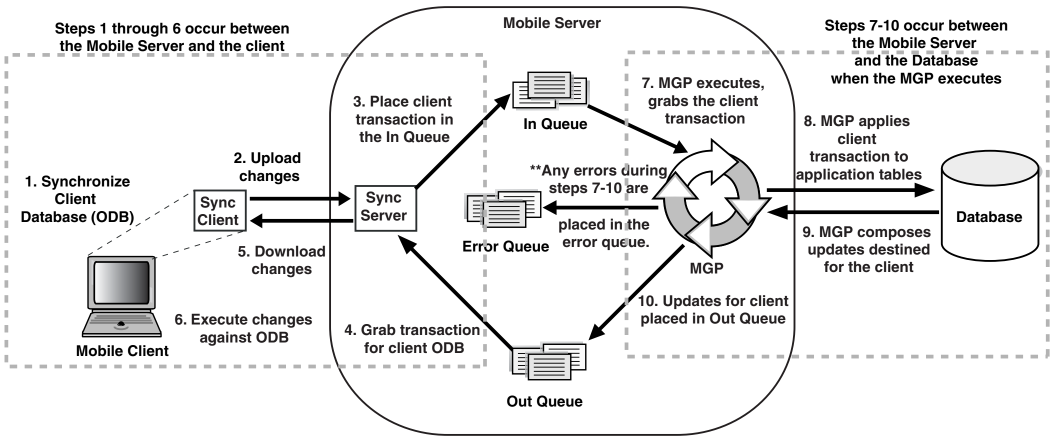
The following sections describe how synchronization works for Oracle Database Lite:

- Section 3.1.1, "Synchronization Overview"
- Section 3.1.2, "Automatic or Manual Synchronization"
- Section 3.1.3, "How Updates Are Propagated to the Back-End Database"

3.1.1 Synchronization Overview

The full description of how synchronization works is in the "Managing Synchronization" chapter in the *Oracle Database Lite Administration and Deployment Guide*. Each component and its function is described in the administration guide. The following graphic depicts these components for your reference:

Figure 3–1 Synchronization Architecture


Steps 1 through 6 occur between the Mobile Server and the client
Mobile Server
Steps 7-10 occur between the Mobile Server and the Database when the MGP executes
1. Synchronize Client Database (ODB)
Sync Client
2. Upload changes
5. Download changes
6. Execute changes against ODB
Mobile Client
3. Place client transaction in the In Queue
Sync Server
4. Grab transaction for client ODB
In Queue
**Any errors during steps 7-10 are placed in the error queue.
Error Queue
Out Queue
7. MGP executes, grabs the client transaction
MGP
10. Updates for client placed in Out Queue
8. MGP applies client transaction to application tables
9. MGP composes updates destined for the client
Database

1. A synchronization is initiated on the Mobile client either by the user or from automatic synchronization. Note that the Mobile client may be a Windows platform client or a PDA.
2. Mobile client software gathers all of the client changes into a transaction and the Sync Client uploads the transaction to the Sync Server on the Mobile Server.
3. Sync Server places the transaction into the In-Queue.

Note: When packaging your application, you can specify if the transaction is to be applied at the same time as the synchronization. If you set this option, then the transaction is immediately applied to the application tables. However, note that this may not be scaleable and you should only do this if the application of the transaction immediately is important and you have enough resources to handle the load.

---

4. Sync Server gathers all transactions destined for the Mobile client from the Out-Queue.
5. Sync client downloads all changes for client Oracle Lite database.
6. Mobile client applies all changes for client Oracle Lite database. If this is the first synchronization, the Oracle Lite database is created.

> **Note:** For information on what Oracle Lite database (ODB) files are installed on the client, see Section 2.2, "Synchronizing or Executing Applications on the Mobile Client" in the *Oracle Database Lite Administration and Deployment Guide*.

7. All transactions compiled from all Mobile clients are gathered by the MGP out of the In-Queue.
8. The MGP executes the apply phase by applying all transactions for the Mobile clients to their respective application tables to the back-end Oracle database. The MGP commits after processing each publication.
9. MGP executes the compose phase by gathering the client data into outgoing transactions for Mobile clients.
10. MGP places the composed data for Mobile clients into the Out-Queue, waiting for the next client synchronization for the Sync Server to gather the updates to the client.

When we discuss how to perform the tasks associated with synchronization, refer back to this graphic to discover what part of the synchronization process that we are discussing.

3.1.2 Automatic or Manual Synchronization

In the past, all that was available was manual synchronization. That is, a client manually requests a synchronization either through an application program executing an API or by a user manually pushing the Sync button.

Currently, you can configure for synchronization to automatically occur under specific circumstances and conditions. When these conditions are met, then Oracle Database Lite automatically performs the synchronization for you without locking your database, so you can continue to work while the synchronization happens in the background. This way, synchronization can happen seamlessly without the client's knowledge.

> **Note:** Within a publication, you can have one or more publication items. You can define both manual and automatic publication items within the same publication.

For example, you may choose to enable automatic synchronization for the following scenarios:

- If you have a user who changes data on their handheld device, but does not sync as often as you would prefer.
- If you have multiple users who all sync at the same time and overload your system.

These are just a few examples of how automatic synchronization can make managing your data easier, be more timely, and occur at the moment you need it to be uploaded.

Note: When a manual synchronization is requested by the client, ALL publication items are synchronized at that time—including those defined as manual and automatic synchronization. However, if an automatic synchronization is currently executing, the manual synchronization request is delayed until the automatic synchronization completes. You can stop the automatic synchronization to allow the manual synchronization to occur. After the manual synchronization is finished, re-start the automatic synchronization.

The differences between the two types of synchronization are as follows:

Table 3–1 Difference Between Automatic and Manual Synchronization

	Manual Synchronization	**Automatic Synchronization**
Initiation	After the snapshot is set up, you can initiate either by the user initiating mSync or by an application invoking one of the synchronization APIs.	All of the set up for automatic synchronization is configured. Once configured, it happens automatically, so there is no synchronization API. Configuration for automatic synchronization can be defined when you create the publication item, publication or the platform.
Controlling synchronization	Synchronization occurs exactly when the user/application requests it.	Synchronization occurs without the user being aware of it occuring. You may have to manage synchronization through the Sync Control API if you have publications that contain both manual and automatic synchronization publication items.

Automatic synchronization is based on a different model than manual synchronization. Automatic synchronization operates on a transactional basis. Thus, when the conditions are correct, any new data transactions are uploaded to the server, in the order of the specified priority for the data. In the manual synchronization model, you can synchronize all data or use the selective sync option, where you can detail only certain portions of the data to be synchronized. The selective sync option is not supported in automatic synchronization, since we are no longer concerned with synchronization of only a subset of data.

For more information on Automatic Synchronization, see Section 3.2, "Automatic Synchronization Overview".

3.1.3 How Updates Are Propagated to the Back-End Database

The synchronization process applies client operations to the tables in the back-end database, as follows:

1. The operations for each publication item are processed according to table weight. The publication creator assigns the table weight to publication items within a specific publication. This value can be an integer between 1 and 1023. For example, a publication can have more than one publication item of weight "2" which would

have `INSERT` operations performed after those for any publication item of a lower weight within the same publication. You define the order weight for tables when you add a publication item to the publication. See Section 3.4.1.7.2, "Using Table Weight" for more information.

2. Within each publication item being processed, the SQL operations are processed as follows:

   a. Client `INSERT` operations are executed first, from lowest to highest table weight order.

   b. Client `DELETE` operations are executed next, from highest to lowest table weight order.

   c. Client `UPDATE` operations are executed last, from highest to lowest table weight order.

For details and an example of exactly how the weights and SQL operations are processed, see Section 3.4.1.7.2, "Using Table Weight".

Note: This order of executing operations can cause constraint violations. See Section 3.11, "Synchronizing With Database Constraints" for more information.

In addition, the order in which SQL statements are executed against the client Oracle Lite database is not the same as how synchronization propagates these modifications. Instead, synchronization captures the end result of all SQL modifications as follows:

1. Insert an employee record 4 with name of Joe Judson.
2. Update employee record 4 with address.
3. Update employee record 4 with salary.
4. Update employee record 4 with office number
5. Update employee record 4 with work email address.

When synchronization occurs, all modifications are captured and only a single insert is performed on the back-end database. The insert contains the primary key, name, address, salary, office number and email address. Even though the data was created with multiple updates, the Synch Server only takes the final result and makes a single insert.

3.2 Automatic Synchronization Overview

Automatic synchronization occurs in the background, so that the user does not have to perform a synchronization; thus, the client appears continually connected to the back-end database without user interaction. All modifications to each record are saved in a log within the client Oracle Lite database. When you requested synchronization manually, Oracle Database Lite locked the database while processing your request. However, with automatic synchronization, it could be occurring while you are performing other tasks to the Oracle Lite database.

When synchronization occurs, all of the modified records stored in the log are uploaded to the server. In addition, any modified records from the server are downloaded into the client Oracle Lite database. This occurs in the same manner as manual synchronization. The only difference is when the synchronization is executed and how the modified records are stored.

The following are details about automatic synchronization:

Table 3–2 Automatic Synchronization

Steps for Automatic Synchronization	See the Following for Details
The developer enables the publication item to use automatic synchronization.	Section 3.2.1, "Enable Automatic Synchronization at the Publication Item Level"
The client can disable and enable automatic synchronization through the client Workspace or with the Sync Control API.	Section 3.2.2, "Enable/Disable Automatic Synchronization"
You can configure under what rules the automatic synchronization occurs.	Section 3.2.3, "Define the Rules Under Which the Automatic Synchronization Starts"
The server can notify the client of data waiting for download.	Section 3.19.3, "Selecting How/When to Notify Clients of Composed Data"
The client application can request status of the outcome of an automatic synchronization.	Section 3.2.5, "Notify Application on Completion of Automatic Synchronization Cycle"

The following sections detail how you can configure for automatic synchronization:

- Section 3.2.1, "Enable Automatic Synchronization at the Publication Item Level"
- Section 3.2.2, "Enable/Disable Automatic Synchronization"
- Section 3.2.3, "Define the Rules Under Which the Automatic Synchronization Starts"
- Section 3.2.4, "Enable the Server to Notify the Client to Initiate a Synchronization to Download Data"
- Section 3.2.5, "Notify Application on Completion of Automatic Synchronization Cycle"

3.2.1 Enable Automatic Synchronization at the Publication Item Level

Automatic synchronization can be enabled at publication item level. It is only the "enabled" publication items within a snapshot that can have automatic synchronization. All other publication items use manual synchronization. See Section 5.4, "Create a Publication Item" for details of how to enable synchronization in a publication item using MDW or Section 3.4.1.3, "Create Publication Items" using the API.

Within a publication, you can have one or more publication items. You can define both manual and automatic synchronization publication items within the same publication. However, if you have automatic synchronization enabled, then an automatic sync may be occurring when the client asks for a manual synchronization. In this case, the manual synchronization stops the automatic synchronization so that all snapshots are synchronized. Automatic synchronization is restarted after the manual synchronization completes.

If you want the manual synchronization to occur at that moment, you can stop the automatic synchronization to allow the manual synchronization to occur. After the manual synchronization is finished, re-start the automatic synchronization. You can start and stop automatic synchronization either programmatically or through the client Workspace. See Section 3.2.2, "Enable/Disable Automatic Synchronization" for full details.

3.2.2 Enable/Disable Automatic Synchronization

Automatic synchronization is enabled by default if a publication is enabled for automated synchronization. However, there may be a situation where you want to disable this automated ability, as follows:

- Enable/Disable—If you decide to disable the automatic synchronization; then, even if you restart the client, automatic synchronization will not occur. Use enable/disable for permanently disabling automatic synchronization.
- Start/Stop—If you decide to stop automatic synchronization; then, if you restart the client, automatic synchronization is restarted. Use start/stop for temporarily stopping automatic synchronization while a manual synchronization occurs.

If you are using the mSync GUI to initiate a synchronization, the underlying code performs the following for you:

1. Stops the automatic synchronization with the Sync Control API.
2. Initiates a manual synchronization with the programmatic API.
3. Starts the automatic synchronization with the Sync Control API.

However, if you are performing the synchronization programmatically with the doSynchronize method, then you may need to perform the stop/start methods in your application to ensure that automatic synchronization is not executing.

The following control APIs can be used to manage the automatic synchronization or enable/disable automatic synchronization:

- Section 3.2.2.1, "POLITE.INI Configuration to Enable/Disable Automatic Synchronization"
- Section 3.2.2.2, "Overview of the Start/Stop Methods from the Sync Control API."
- Section 3.2.2.3, "C/C++ Sync Control APIs to Start/Stop Automatic Synchronization"
- Section 3.2.2.4, "C# Sync Control APIs to Start/Stop Automatic Synchronization"
- Section 3.2.2.5, "JAVA Sync Control APIs to Start/Stop Automatic Synchronization"

3.2.2.1 POLITE.INI Configuration to Enable/Disable Automatic Synchronization

The start and stop methods only control the automatic synchronization temporarily. To fully disable automatic synchronization, so that it is not restarted when a device is powered on, perform one of the following:

- Enable/disable automatic synchronization using the Client Workspace. For details, see Section 2.3.1.1.2 "Configuration Tab" in the *Oracle Database Lite Administration and Deployment Guide*.
- Enable/disable automatic synchronization in the `polite.ini` file. Set the `ENABLE` parameter, which is a part of the `SYNC_AGENT` section, in the `polite.ini` to `Yes` to enable and `No` to disable. This can also be accomplished through the `syncagent.exe` UI. See the following sections for details: Section F.3.2.14, "SYNC_AGENT" or Section 5.4.2, "Start, Stop, or Get Status for Automatic Synchronization," which are located in the *Oracle Database Lite Administration and Deployment Guide*.

3.2.2.2 Overview of the Start/Stop Methods from the Sync Control API.

Stop/start automatic synchronization using the Sync Control API. The stop API has one parameter for input, which is a timeout. You can supply one of the following values for the timeout, which is a long that specifies a time in milliseconds to wait for any current activity in the automatic synchronization to complete.

- `BG_STOP_TIMEOUT`: A value in seconds that allows the automatic synchronization process to complete before stopping the service. By default, this is set to 5 seconds.
- `BG_KILL_AGENT`: A value of -1 that makes the automatic synchronization service stop immediately, even if it is in the middle of a synchronization. If an automatic synchronization is in process, it will be terminated. NO errors or messages are returned.
- Any long value in milliseconds: If the automatic synchronization does not stop within the time designated, then the method returns with an error of `BG_ERROR_TIMEOUT`. At this point, you reissue the stop method to terminate the automatic synchronization immediately by supplying `BG_KILL_AGENT` or -1 as the input value.

Note: There is also a GUI for starting, stopping the automatic synchronization process. See Section 5.3.1, "Start, Stop, or Get Status for Automatic Synchronization" in the *Oracle Database Lite Administration and Deployment Guide* for more details.

3.2.2.3 C/C++ Sync Control APIs to Start/Stop Automatic Synchronization

Use the control APIs for starting and stopping automatic synchronization. These APIs are as follows:

```
olError olStartSyncAgent() ;
olError olStopSyncAgent(long timeout);
```

3.2.2.4 C# Sync Control APIs to Start/Stop Automatic Synchronization

```
public class BGSyncControl
{
  public void start();
  public void stop(int timeout);
}
```

All methods throw an `OracleException` in case of failure.

3.2.2.5 JAVA Sync Control APIs to Start/Stop Automatic Synchronization

```
package oracle.lite.msync;
class BGSyncControl {
  public void start() throws SyncException;
  public void stop(long timeout) throws SyncException;
}
```

3.2.3 Define the Rules Under Which the Automatic Synchronization Starts

You can configure under what circumstances a synchronization should occur and then Oracle Database Lite performs the synchronization for you automatically. The circumstances under which an automatic synchronization occurs is defined within the synchronization rules, which includes the following:

- Events—An event is variable, as follows:

  - Data events: For example, you can specify that a synchronization occurs when there are a certain number of modified records in the client database.
  - System events: For example, you can specify that if the battery drops below a predefined minimum, you want to synchronize before the battery is depleted.
- Conditions—A condition is an aspect of the client that needs to be present for a synchronization to occur. This includes conditions such as battery life or network availability.

The relationship between events and conditions when evaluating if an automatic synchronization occurs is as follows:

```
when EVENT and if (CONDITIONS), then SYNC
```

For example, if the event for new data inserted and the condition specified is that the network must be available, then a synchronization only occurs when the network is available and there is new data.

You can define the rules for automatic synchronization within certain parts of the normal snapshot setup and platform configuration, as follows:

- Publication level: Within the publication, you specify the rules under which the synchronization occurs for all publication items in that publication.
- Platform level: Some of the rules are specific to the platform of the client, such as battery life, network bandwidth, and so on. These rules apply to all enabled publication items that exist on this particular platform, such as WinCE.

The following sections detail all of the rules you can configure for automatic synchronization:

- Section 3.2.3.1, "Configure Publication-Level Automatic Synchronization Rules"
- Section 3.2.3.2, "Configure Platform-Level Automatic Synchronization Rules"

3.2.3.1 Configure Publication-Level Automatic Synchronization Rules

Within the publication, you specify the rules under which the synchronization occurs for all publication items in that publication. These rules are defined when you create the publication either using MDW or programmatically with the APIs. To create this through MDW, see Section 5.5, "Define the Rules Under Which the Automatic Synchronization Starts" ; to add publication-level automatic synchronization rules with the API, see Section 3.4.1.4, "Define Publication-Level Automatic Synchronization Rules".

When you are creating the publication, you can define events that will cause an automatic synchronization. Although these are defined at the publication level, they enable only the publication items within this publication that has automatic synchronization enabled.

Table 3–3 describes the publication level events for automatic synchronization. The lowest value that can be provided is 1.

Table 3–3 Automatic Events for the Publication

Events	Description
Client commit	Upon commit to the Oracle Lite database, the Mobile client detects the total number of record changes in the automatic synchronization log. If the number of modifications is equal to or greater than your pre-defined number, automatic synchronization occurs. This rule is on by default and set to start an automatic synchronization if only one record is changed.

Table 3–3 (Cont.) Automatic Events for the Publication

Events	Description
Server MGP compose	If after the MGP compose cycle, the number of modified records for a user is equal to or greater than your pre-defined number, then an automatic synchronization occurs. Thus, if there are a certain number of records contained in an Out Queue destined for a client on the server, these modifications are synchronized to the client.

Note: If you want to modify the publication-level automatic synchronization rules after you publish the appliation, you can do so through the Mobile Manager, as follows:

1. Click **Data Synchronization**.
2. Click **Repository**.
3. Click **Publications**.
4. Select the publication and click **Automatic Synchronization Rules**.

3.2.3.2 Configure Platform-Level Automatic Synchronization Rules

Some of the rules are specific to the platform of the client, such as battery life, network bandwidth, and so on. These rules apply to all enabled publication items that exist on this particular platform, such as WinCE. You configure these rules through Mobile Manager or MDW. This section describes Mobile Manager.

The platform-level synchronization rules apply to a selected client platform and all publications that exist on that platform. You can specify both platform events and conditions using the Mobile Manager.

To assign platform-level automatic synchronization rules, perform the following in Mobile Manager:

1. Click **Data Synchronization**.
2. Click **Platform Settings**, which brings up a page with the list of all the platforms that support automatic synchronization.
3. Click on the desired platform.
4. You can modify the following for each platform:
   - Event Rules—See Section 3.2.3.2.1, "Event Rules for Platforms".
   - Conditions—See Section 3.2.3.2.2, "Condition Rules for Platforms".
   - Network settings—See Section 3.2.3.2.3, "Network Configuration for the Client Platform".

3.2.3.2.1 Event Rules for Platforms Table 3–4 shows the platform events for automatic synchronization.

Table 3–4 Automatic Event Rules for the Client Platform

Event	Description
Network bandwidth	If the Mobile client detects that it is connected to a network with a pre-defined minimum bandwidth, then automatic synchronization occurs.
Battery life	If the battery life drops below a pre-defined minimum, then synchronization is automatically triggered.

Table 3–4 (Cont.) Automatic Event Rules for the Client Platform

Event	Description
AC Power	As soon as AC power is detected, then synchronization is automatically triggered.
Time	Synchronize at a specific time or time interval. You can configure an automatic synchronization to occur at a specific time each day or as an interval. ■ Select **Specify Time** if you want to automatically synchronize at a specific hour, such as 8:00 AM, everyday. ■ Select **Specify Time Interval** if you want to synchronize at a specific interval. For example, if you want to synchronize every hour, then specify how long to wait in-between synchronization attempts.

3.2.3.2.2 Condition Rules for Platforms Table 3–5 shows the platform conditions for automatic synchronization.

Table 3–5 Automatic Condition Rules for Client Platform

Condition	Description
Battery level	Specify the minimum battery level required in order for an automatic synchronization to start. The battery level is specified as a percentage.
Network conditions	Network quality can be specified using several properties. This condition enables you to specify a minimum value for the following network properties: ■ Minimum network bandwidth, which is measured in bits per second. ■ Maximum ping delay, which is measured in milliseconds. ■ Data priority, which is either high or regular. You can specify the priority of your data in the table row. For example, you can define a rule where all high priority data is automatically synchronized at a specified network bandwidth. The ping delay is optional. If not specified, the ping is not calculated.

3.2.3.2.3 Network Configuration for the Client Platform You can set proxy information for your network provider, if required for accessing the internet.

Note: If you are not using a proxy, then you do not need to define proxy information on this page.

You could have two types of networks, as follows:

- Always-on: Define the proxy and port number. Click **Apply** when finished.
- Dial-up:
  - Click **Add Dial-up Network** to add a a new entry for dial-up configuration.
  - To edit an existing configuration, select the name of the existing configuration.
  - To delete an existing configuration, select the checkbox next to the desired configuration and click **Delete**.

If the platform has an always-on network, then this network is always tried first for the connection. If this network is not available, then the dial-up networks are tried in the order specified. You can rearrange the order of the dial-up networks by selecting one of the networks and clicking the up or down button.

3.2.4 Enable the Server to Notify the Client to Initiate a Synchronization to Download Data

If you have designed the compose yourself—that is, you do not use the MGP—then, you can notify the client if any data exists on the server that can be downloaded to the client through enqueue notification APIs. You can also use these APIs to manage the automatic synchronization schedule for your clients.

For more information on enqueue notification APIs, see Section 3.19.3, "Selecting How/When to Notify Clients of Composed Data".

3.2.5 Notify Application on Completion of Automatic Synchronization Cycle

You can develop your client application to be notified when an automatic synchronization cycle occurs. The application is notified from the Sync Agent when the automatic synchronization completes as well as when a critical event occurs in the client device. For example, when the device battery runs critically low, Oracle Database Lite can notify the application.

In the client application, create a procedure that executes one of the following message APIs. When your application calls the get message API, it blocks until an event occurs within an automatic synchronization. It returns a structure that describes this event.

The following sections provide implementation details for each development language:

- Automatic Synchronization Notification for C/C++ Application
- Automatic Synchronization Notification for C# Application
- Automatic Synchronization Notification for Java Application
- Input Parameters for Automatic Synchronization Notification

Automatic Synchronization Notification for C/C++ Application

Use the `olGetSyncMsg` method in your client application to receive the automatic synchronization notification when implementing for C/C++ applications. In order to block for the status, you need to perform the following:

1. Start the application messaging service with the `olStartSyncMsg` method, providing a queue handle of type `olAppMsgQ`. This message starts the messaging service and returns the queue handle in the `olAppMsgQ`.
2. Execute the `olGetSyncMsg` with the `olAppMsgQ` message handle and the defined `olSyncMsg` structure for the returned automatic synchronization information.

The following provides the method definitions:

```
typedef void *olAppMsgQ
/* start application messaging, get queue handle */
olError olStartSyncMsg(olAppMsgQ *q);
/*Provide the queue handle and block to retrieve automatic sync event */
olError olGetSyncMsg(olAppMsgQ q, olSyncMsg *m);
```

The `olGetSyncMsg` method blocks until an event occurs, then the Sync Agent returns the `olSyncMsg` class, which you provide as an input parameter, with the information on what happened, as follows:

```
typedef struct _olSyncMsg {
    ol2B type;
    ol2B id;
    char msg[BG_MAX_MSG];
} olSyncMsg;
```

See "Input Parameters for Automatic Synchronization Notification" for a description of the input parameters in the structure.

The C/C++ application performs in a different manner than the Java and C# versions in that this creates a message service with its own message queue. Thus, when finished you must perform some cleanup to ensure that the message queue handle is released. Use the `olStopSyncMsg` method to stop the messaging service and release the handle. This must be performed for every message queue that is opened with the `olStartSyncMsg` method.

```
olError olStopSyncMsg(olAppMsgQ q);
```

If you want to force an existing `olGetSyncMsg` to return, use the `olCancelSyncMsg` from another thread in the application. This causes the `olGetSyncMsg` to return with the `BG_ERR_APP_MSG_CANCEL` error.

```
olError olCancelSyncMsg(olAppMsgQ q);
```

Automatic Synchronization Notification for C# Application

Use the `GetMessage` method in your client application to receive the automatic synchronization notification when implementing for C# applications, as follows:

```
public BGSyncMsg GetMessage();
```

This method blocks until an event occurs, then the Sync Agent returns the `BGSyncMsg` class with the information on what happened, as follows:

```
public class BGSyncMsg
{
 public int Type;
 public int Id;
 public string Msg;
}
```

See "Input Parameters for Automatic Synchronization Notification" for a description of the input parameters in the class.

Automatic Synchronization Notification for Java Application

Use the `getMessage` method in your client application to receive the automatic synchronization notification when implementing for Java applications, as follows:

```
public class BGSyncControl
{
 public BGSyncMsg getMessage() throws SyncException;
}
```

This method blocks until an event occurs, then the Sync Agent returns the `BGSyncMsg` class with the information on what happened, as follows:

```
public class BGSyncMsg{
    public int type;
    public int id;
    public String msg;
}
```

See "Input Parameters for Automatic Synchronization Notification" for a description of the input parameters in the class.

Input Parameters for Automatic Synchronization Notification

The input parameters in the input structure/class are as follows:

Table 3–6 The Sync Message Variables

Variable	Description
Event type	The event can be of three types, each of which indicate the level of severity of this notification: ■ `INFO` ■ `ERROR` ■ `WARNING`
Event identifier for `INFO` types:	The INFO event identifer describes what occurred, as follows: ■ `SYNC_STARTED`: The Sync Agent has started the synchronization task. ■ `SYNC_SUCCEEDED`: Data synchronization completed successfully. ■ `APPLY_STARTED`: The Sync Agent has started the apply task. ■ `APPLY_SUCCEEDED`: The apply phase completed successfully. ■ `SVR_NOTIF`: The Sync Agent has received a server notification. The message contains information about the server notification, such as publication name, number of modified records and the record priority (high priority or normal). ■ `NETWORK_CHANGE`: Device has moved into a different network ■ `AGENT_STARTED`: The Sync Agent started. ■ `AGENT_STOPPED`: The Sync Agent stopped.
Event identifier for the `WARNING` type:	The WARNING event identifier describes in more detail what occurred, as follows: ■ `BATTERY_LOW`: Device's battery is running low ■ `MEMORY_LOW`: Device's memory is running low
Event identifier for the `ERROR` type:	The `ERROR` event identifier describes in more detail what occurred, as follows: ■ `APPLY_FAILED`: The apply failed. In this case, 'message' contains the reason for failure. ■ `SYNC_FAILED`: Data synchronization failed. In this case, 'message' contains the reason for failure. ■ `AGENT_ERROR`: An internal error condition occurred. The message contains the actual error message. Examples would be failure to load a rule, failure to process server notification, failure to evaluate system power, and so on. In spite of this error, the Sync Agent continues to execute. Fatal errors are written to the `olSyncAgent.err` file.
Event Message	String message that expounds on the information provided by the event type and identifier.

3.2.6 Request Status for Automatic Synchronization Cycle

If you want to know at what stage the automatic synchronization cycle is, you can request status from the Sync Agent. In the client application, execute the get status API, which will return immediately with at what stage the automatic synchronization cycle is executing. This is different from the notification message API, which only returns when an event is completed within the synchronization cycle.

The get status API returns a structure that describes this event.

The following sections provide implementation details for each development language:

- Retrieving Status for C/C++ Application
- Retrieving Status for C# Application
- Retrieving Status for Java Application
- Input Parameters for Retrieving Messages

Retrieving Status for C/C++ Application

Use the `olGetSyncStatus` method in your C/C++ client application to retrieve status on the automatic synchronization, as follows:

```
olError olGetSyncStatus(olSyncStatus *s);
```

The Sync Agent returns the `olSyncStatus` class, which you provide as an input parameter, with the information on what happened, as follows:

```
typedef struct _olSyncStatus {
    char clientId[BG_MAX_USERNAME];
    ol2B syncState;
    ol2B syncProgress;
    char syncStateStr[BG_MAX_STATUS_STR];
    olError lastSyncError;
    ol2B lastSyncType;
    ol8B lastSyncTime;
    ol2B applyState;
    ol2B applyProgress;
    char applyStateStr[BG_MAX_STATUS_STR];
    olError lastApplyError;
    olU2B _reserved;
    ol8B lastApplyTime;
    char networkName[BG_MAX_STATUS_STR];
    ol4B networkSpeed;
    ol4B batteryPower;
} olSyncStatus;
```

See "Input Parameters for Retrieving Messages" for a description of the input parameters in the structure.

Retrieving Status for C# Application

Use the `GetStatus` method in your C/C++ client application to retrieve status on the automatic synchronization, as follows:

```
public BGSyncStatus GetStatus();
```

This method returns the `BGSyncStatus` class with the status information on the automatic synchronization, as follows:

```
public class BGSyncStatus
```

```
{
  public string clientId;
  public short  syncState;
  public string syncStateStr;
  public short syncProgress;
  public short lastSyncError;
  public short lastSyncType;
  public long lastSyncTime;
  public short applyState;
  public string applyStateStr;
  public short applyProgress;
  public short lastApplyError;
  public ushort  _reserved;
  public long  lastApplyTime;
  public string networkName;
  public int networkSpeed;
  public int batteryPower;
}
```

See "Input Parameters for Retrieving Messages" for a description of the input parameters in the structure.

Retrieving Status for Java Application

Use the `getStatus` method in your Java client application to retrieve status on the automatic synchronization, as follows:

```
public BGSyncStatus getStatus() throws SyncException
```

This method returns the `BGSyncStatus` class with the status information on the automatic synchronization, as follows:

```
public class BGSyncStatus
{
   public String clientId;
   public short  syncState;
   public String syncStateStr;
   public short syncProgress;
   public short lastSyncError;
   public short lastSyncType;
   public Date lastSyncTime;

   public short applyState;
   public String applyStateStr;
   public short applyProgress;
   public short lastApplyError;
   public Date  lastApplyTime;

   public String networkName;
   public int networkSpeed;
   public int batteryPower;
}
```

See "Input Parameters for Retrieving Messages" for a description of the input parameters in the structure.

Input Parameters for Retrieving Messages

The input parameters in the input structure/class are as follows:

Table 3–7 Status Class Fields

Field	Description
clientId	Username
syncState	A numeric value that denotes the current synchronization stage, such as compose, send, or receive.
syncStateStr	String describing the state, as denoted in the `syncState`, for the automatic synchronization.
syncProgress	A percentage that indicates the current progress for the automatic synchronization.
lastSyncError	If an error occurred in the last synchronization, this is the error code. If no error, this value is zero.
lastSyncType	The priority of the data for the last synchronization. If 1, then high priority data; if 0, then regular priority data was synchronized.
lastSyncTime	Time of the last automatic synchronization.
applyState	Code that indicates the state for the apply phase.
applyStateStr	String describing the state for the apply phase, as denoted in the `applyState` variable.
applyProgress	A percentage that indicates the current progress for the apply phase.
lastApplyError	If an error occurred in the last apply phase, this is the error code. If no error, this value is zero.
lastApplyTime	Time of the last apply phase.
networkName	The network name assigned to this network.
networkSpeed	Current bandwidth of the network.
batteryPower	Current battery power percentage.

3.3 What is The Process for Setting Up a User For Synchronization?

Before you can perform the synchronization, the publication must be created, the user created and granted access to the publication, and optionally, the publication packaged up with an application and published to the Mobile Server. This is referred to as the publish and subscribe model, which can be implemented in one of two ways:

- Declaratively, using MDW to create the publication and the Packaging Wizard to package and publish the applications. This is the recommended method. See Section 3.3.1, "Creating a Snapshot Definition Declaratively" for details.
- Programmatically, using the Resource Manager and the Consolidator Manager APIs to invoke certain advanced features or customize an implementation. This technique is recommended for advanced users requiring specialized functionality. See Section 3.3.2, "Creating the Snapshot Definition Programmatically" for details.

Once created and subscribed, the user can be synchronized, as follows:

- Using manual synchronization where the user initiates it from the device or programmatically from within an application. This chapter discusses how to start the synchronization programmatically in Section 3.8, "Initiating Client Synchronization With Synchronization APIs".
- Using automatic synchronization which is enabled within the publication item itself or the platform configuration.

On the back-end of the synchronization process, you have the option to customize how the apply and compose phase are executed. See Section 3.6, "Customize the Compose Phase Using MyCompose".

3.3.1 Creating a Snapshot Definition Declaratively

Use the Mobile Database Workbench (MDW), a GUI based tool of Oracle Database Lite—described fully in Chapter 5, "Using Mobile Database Workbench to Create Publications"—to create snapshots declaratively. The convenience of a graphical tool is a safer and less error prone technique for developers to create a Mobile application. Before actual application programming begins, the following steps must be executed:

1. Verify that the base tables exist on the server database; if not, create the base table.
2. Use MDW to define an application and the snapshot with the necessary publicatino and its publication items. See Chapter 5, "Using Mobile Database Workbench to Create Publications" for details.
3. Use the Packaging Wizard to publish the application to the Mobile Server. This creates the publication items associated with the application. See Chapter 7, "Using the Packaging Wizard" for details.
4. Use the Mobile Manager to create a subscription for a given user.
5. Install the application on the development machine.
6. If using manual synchronization, then initiate synchronization for the Mobile client with the Mobile Server to create the client-side snapshots, which creates the Mobile client Oracle Lite database automatically.

3.3.1.1 Manage Snapshots

The Mobile Server administrator can manage a snapshot, which is a full set or a subset of rows of a table or view. Create the snapshot by executing a SQL query against the base table. Snapshots are either read-only or updatable.

The following sections describes how to manage snapshots using MDW:

- Section 3.3.1.1.1, "Read-only Snapshots"
- Section 3.3.1.1.2, "Updatable Snapshots"
- Section 3.3.1.1.3, "Refresh a Snapshot"
- Section 3.3.1.1.4, "Snapshot Template Variables"

3.3.1.1.1 Read-only Snapshots Read-only snapshots are used for querying purposes only. The data is downloaded from the Oracle server to the client; no data on the client is ever uploaded to the server. Any data added on the client in a read-only snapshot can be lost, since it is never uploaded to the server. Changes made to the master table in the back-end Oracle database server are replicated to the Mobile client. See Section 5.9.2, "Publication Item Tab Associates Publication Items With the Publication" for instructions on how to define the publication item as read-only.

Note: A subscription created as complete refresh and read-only is light weight; thus, to keep the subscription light weight, the primary keys are not included in the replication. If you want to include primary keys, then create them with the `createPublicationItemIndex` API.

3.3.1.1.2 Updatable Snapshots When you define a snapshot as updatable, then the data propagated within a synchronization is bi-directional. That is, any modifications made on the client are uploaded to the server; any modifications made on the back-end Oracle server are downloaded to the client. See Section 5.9.2, "Publication Item Tab Associates Publication Items With the Publication" for instructions on how to define the publication item as updatable.

A snapshot can only be updated when all the base tables that the snapshot is based on have a primary key or virtual primary key. If the base tables do not have a primary key, a snapshot cannot be updated and becomes read-only. Table 3–8 shows each refresh method type and whether it is updatable or read-only depending on primary key or virtual primary key:

Table 3–8 Which Refresh Methods Can Be Updatable or Read-Only

	Fast	Complete	Queue-Based
Table Uses a Primary Key	Updatable or Read-Only	Updatable or Read-Only	Updatable or Read-Only
Table Uses a Virtual Primary Key	Updatable or Read-Only	Updatable or Read-Only	Updatable or Read-Only
No Primary Key or Virtual Primary Key Used	Not applicable since all Fast Refresh tables use a primary or virtual primary key.	Read-Only	Read-Only

3.3.1.1.3 Refresh a Snapshot Your snapshot definition determines whether an updatable snapshot uses the complete or fast refresh method.

- The complete refresh method recreates the snapshot every time it is refreshed. Note that when it recreates the snapshot, all of the data on the client Oracle Lite database is erased and then the snapshot for this user on the back-end Oracle database is brought down to the client.
- The fast refresh method refreshes only the modified data within the snapshot definition on both the client and server. In general, the simpler your snapshot definition, the faster it is updated. All fast refresh methods require a primary key or a virtual primary key.

See Section 5.4, "Create a Publication Item" and Section 3.9, "Understanding Your Refresh Options"

3.3.1.1.4 Snapshot Template Variables Snapshots are application-based. In some cases, you may quantify the data that your application downloads for each user by specifying all of the returned data match a predicate. You can accomplish this by using snapshot templates.

A snapshot template is an SQL query that contains data subsetting parameters. A data subsetting parameter is a colon `(:)`, followed by an identifier name, as follows:

```
:var1
```

Note: If the subsetting parameter is on a CHAR column of a specified length, then you should either preset all characters to spaces before setting the value or pad for the length of the column with spaces after setting the parameter.

When the Mobile client creates snapshots on the client machine, the Mobile Server replaces the snapshot variables with user-specific values. By specifying different values for different users, you can control the data returned by the query for each user.

You can use MDW to specify a snapshot template variable in the same way that you create a snapshot definition for any platform.

Data subsetting parameters are bind variables and so should not be enclosed in quotation marks `(')`. If you want to specify a string as the value of the data subsetting parameter, then the string contains single quotation marks. You can specify the values for the template variables within the Mobile Manager.

The following examples specify a different value for every user. By specifying a different value for every user, the administrator controls the behavior and output of the snapshot template.

```
select * from emp where deptno = :dno
```

You define this select statement in your publication item. See Section 5.4.1, "Create SQL Statement for Publication Item" for instructions. Then, modify the user in the Mobile Manager to add the value for `:dno`. Then, when the user synchronizes, the value defined for the user is replaced in the select script. See Section 5.3, "Managing Application Parameter Input (Data Subsetting)" in the *Oracle Database Lite Administration and Deployment Guide* for information on how to define the value of the variable. This value can only be defined after the application is published and the user is associated with it.

Table 3–9 provides a sample set of snapshot query values specified for separate users.

Table 3–9 Snapshot Query Values for Separate Users

User	Value	Snapshot Query
John	10	`select * from emp where deptno = 10`
Jane	20	`select * from emp where deptno = 20`

```
select * from emp where ename = :ename
```

Table 3–10 provides another sample snapshot query value.

Table 3–10 Snapshot Query Value for User Names

User	Value	Snapshot Query
John	'KING'	`select * from emp where ename = 'KING'`

3.3.2 Creating the Snapshot Definition Programmatically

You can use the Resource Manager or Consolidator Manager APIs to programmatically create the publication items on the Mobile Server. Create publication items from views and customize code to construct snapshots.

Note: The Consolidator Manager API can only create a publication, which cannot be packaged with an application. In addition, a publication created with the Consolidator Manager API cannot be packaged with an application. See Section 3.4, "Creating Publications Using Oracle Database Lite APIs" for information on the Consolidator Manager API. Use the Resource Manager APIs to create the publication, package it with an application, and publish it to the Mobile Server. See the `oracle.mobile.admin.ResourceManager` Javadoc in the *Oracle Database Lite API Specification*, which you can link to off the *`ORACLE_HOME`*`/Mobile/index.htm` page.

The base tables must exist before the Consolidator Manager API can be invoked. The following steps are required to create a a subscription:

- Create a publication
- Create a publication item and add it to the publication
- Create a user
- Creating a subscription for the user based on the publication

The details of how to create a publication are documented in Chapter 5, "Using Mobile Database Workbench to Create Publications". Anything that you can do with the MDW tool, you can also perform programmatically using the Consolidator Manager API. Refer to the Javadoc for the syntax.

3.4 Creating Publications Using Oracle Database Lite APIs

Mobile Server uses a publish and subscribe model to centrally manage data distribution between Oracle database servers and Oracle Database Lite clients. Basic functions, such as creating publication items and publications, can be implemented easily using the Mobile Development Workspace (MDW). See Chapter 5, "Using Mobile Database Workbench to Create Publications" for more information.

These functions can also be performed using the Consolidator Manager or Resource Manager APIs by writing Java programs to customize the functions as needed. Some of the advanced functionality can only be enabled programmatically using the Consolidator Manager or Resource Manager APIs.

The publish and subscribe model can be implemented one of two ways:

- Declaratively, using MDW to create the publication and the Packaging Wizard to package and publish the applications. This is the recommended method. This method is described fully in Chapter 5, "Using Mobile Database Workbench to Create Publications" and Chapter 7, "Using the Packaging Wizard".
- Programmatically, using the Consolidator Manager or Resource Manager APIs to invoke certain advanced features or customize an implementation. This technique is recommended for advanced users requiring specialized functionality.
  - Publications created with the Consolidator Manager API cannot be packaged with an application. See Section 3.4.1, "Defining a Publication With Java Consolidator Manager APIs".
  - Use the Resource Manager APIs to create the publication, package it with an application, and publish it to the Mobile Server. See the `oracle.mobile.admin.MobileResourceManager` Javadoc in the API

Specification section, which is located off the *ORACLE_HOME*/`Mobile/index.htm` page.

3.4.1 Defining a Publication With Java Consolidator Manager APIs

While we recommend that you use MDW (see Chapter 5, "Using Mobile Database Workbench to Create Publications") or the Packaging Wizard (see Chapter 7, "Using the Packaging Wizard") for creating your publications, you can also create them, including the publication items and the user, with the Consolidator Manager API. Choose this option if you are performing more advanced techniques with your publications.

After creating the database tables in the back-end database, create the Resource Manager and Consolidator Manager objects to facilitate the creation of your publication:

- The Resource Manager object enables you to create users to associate with the subscription.
- The Consolidator Manager object enables you to create the subscription.

The order of creating the elements in the publication is the same as if you were using MDW. You must create a publication first and then add the publication items and other elements to it. Once the publications are created, subscribe users to them. See the Javadoc for full details on each method. See Chapter 5, "Using Mobile Database Workbench to Create Publications" for more details on the order of creating each element.

Note: The following sections use the `sample11.java` sample to demonstrate the Resource Manager and Consolidator Manager methods used to create the publication and the users for the publication. The full source code for this sample can be found in the following directories:

On UNIX: <*ORACLE_HOME*>`/mobile/server/samples`

On Windows: <*ORACLE_HOME*>`\Mobile\Server\Samples`

---

1. Section 3.4.1.1, "Create the Mobile Server User"
2. Section 3.4.1.2, "Create Publications"
3. Section 3.4.1.3, "Create Publication Items"
4. Section 3.4.1.4, "Define Publication-Level Automatic Synchronization Rules"
5. Section 3.4.1.5, "Data Subsetting: Defining Client Subscription Parameters for Publications"
6. Section 3.4.1.6, "Create Publication Item Indexes"
7. Section 3.4.1.7, "Adding Publication Items to Publications"
8. Section 3.4.1.8, "Creating Client-Side Sequences for the Downloaded Snapshot"
9. Section 3.4.1.9, "Subscribing Users to a Publication"
10. Section 3.4.1.10, "Instantiate the Subscription"
11. Section 3.4.1.11, "Bringing the Data From the Subscription Down to the Client"
12. Section 3.4.1.12, "Modifying a Publication Item"

13. Section 3.4.1.13, "Callback Customization for DML Operations"
14. Section 3.4.1.14, "Restricting Predicate"

Note: To call the Publish and Subscribe methods, the following JAR files must be specified in your `CLASSPATH`.

- `<`*`ORACLE_HOME`*`>\jdbc\lib\ojdbc14.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\consolidator.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\classgen.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\servlet.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\xmlparserv2.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\jssl-1_2.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\javax-ssl-1_2.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\Server\bin\devmgr.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\share.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\oracle_ice.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\phaos.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\jewt4.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\jewt4-nls.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\wtgpack.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\jzlib.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\Server\bin\webtogo.jar`

3.4.1.1 Create the Mobile Server User

Use the `createUser` method of the `MobileResourceManager` object to create the user for the publication.

1. Create the `MobileResourceManager` object. A connection is opened to the Mobile Server. Provide the schema name, password, and JDBC URL for the database the contains the schema (the repository).
2. Create one or more users with the `createUser` method. Provide the user name, password, the user's real name, and privilege, which can be one of the one of the following: "O" for publishing an application, "U" for connecting to Web-to-Go as user, or "A" for administrating the Web-to-Go. If NULL, no privilege is assigned.

Note: Always request a drop user before you execute a create, in case this user already exists.

3. Commit the transaction, which was opened when you created the `MobileResourceManager` object, and close the connection.

```
MobileResourceManager mobileResourceManager =
    new MobileResourceManager(CONS_SCHEMA, DEFAULT_PASSWORD, JDBC_URL);
mobileResourceManager.createUser("S11U1", "manager", "S11U1", "U");
mobileResourceManager.commitTransaction();
mobileResourceManager.closeConnection();
```

Note: If you do not want to create any users, you do not need to create the `MobileResourceManager` object.

3.4.1.1.1 Change Password

You can change passwords for Mobile Server users with the `setPassword` method, which has the following syntax:

```
public static void setPassword
   (String userName,
    String newpwd) throws Throwable
```

Note: Both username and passwords are limited to a maximum of 28 characters.

Execute the `setPassword` method before you commit the transaction and release the connection. The following example changes the password for the user `MOBILE`:

```
mobileResourceManager.setPassword("MOBILE","MOBILENEW");
```

3.4.1.2 Create Publications

A subscription is a combination of publications and the users who access the information gathered by the publications. Create any publication through the `ConsolidatorManager` object.

1. Create the `ConsolidatorManager` object.
2. Connect to the database using the `openConnection` method. Provide the schema name, password, and JDBC URL for the database the contains the schema (the repository).
3. Create the publication with the `createPublication` method, which creates an empty publication.

Note: Always request a drop publication before you execute a create, in case this publication already exists.

```
ConsolidatorManager consolidatorManager = new ConsolidatorManager();
consolidatorManager.openConnection(CONS_SCHEMA, DEFAULT_PASSWORD, JDBC_URL);
consolidatorManager.createPublication("T_SAMPLE11",
 Consolidator.OKPI_CREATOR_ID, "OrdersODB.%s", null);
```

Note: Special characters including spaces are supported in publication names. The publication name is case-sensitive.

3.4.1.3 Create Publication Items

An empty publication does not have anything that is helpful until a publication item is added to it. Thus, after creating the publication, it is necessary to create the publication item, which defines the snapshot of the base tables that is downloaded for your user.

When you create each publication item, you can specify the following:

- Automatic or Manual Synchronization: Whether the publication item is to be synchronization automatically or manually.

- Refresh Mode: The refresh mode of the publication item is specified during creation to be either fast, complete-refresh, or queue-based.
- Data-Subsetting Parameters: You can also establish the data-subsetting parameters when creating the publication item, which provides a finer degree of control on the data requirements for a given client.

Publication item names are limited to twenty-six characters and must be unique across all publications. The publication item name is case-sensitive. The following examples create a publication item named `P_SAMPLE11-M`.

> **Note:** Always drop the publication item in case an item with the same name already exists.

The following example uses the `createPublicationItem` method, which creates a manual synchronization publication item `P_SAMPLE11-M` based on the `ORD_MASTER` database table with fast refresh. Use the `addPublicationItem` method to add this publication item to the publication.

> **Note:** For full details on the method parameters, see the Javadoc.

```
consolidatorManager.createPublicationItem("P_SAMPLE11-M", "MASTER",
    "ORD_MASTER", "F", "SELECT * FROM MASTER.ORD_MASTER", null, null);
```

When you create a publication item that uses automatic synchronization through the `createPublicationItem` method, you can also define the following:

- Automatic Synchronization: Set the publication to use automatic synchronization by setting the `isLogBased` flag to true.
- Server-initiated change notifications: If you set the `doChangeNtf` flag to true, then the Mobile Server sends a notification to the client if any changes are made on the server for this publication item.
- Set what constraints are replicated to the client: If you set the `setDfltColOptions` flag to true, then the default values and not null constraints are replicated to the client.
- Create a client sub-query to return unique client ids in the `cl2log_rec_stmt` parameter. The client sub-query correlates the primary key of the changed records in the log table with the Consolidator client id. The log table contains the changes for the table and is named `clg$<tablename>`.

Notes:

- If you are creating a fast refresh publication item on a table with a composite primary key, the snapshot query must match the primary key columns in the order that they are present in the table definition. This automatically happens during the column selection when MDW is used or when a SELECT * query is used. Note that the order of the primary key columns in the table definition may be different from those in the primary key constraint definition.
- A subscription created as complete refresh and read only is light weight; thus, to keep the subscription light weight, the primary keys are not included in the replication. If you want to include a primary key, then you can create it with the `createPublicationItemIndex` API.

For example, if the publication item SQL query is as follows:

```
SELECT * FROM scott.emp a
   WHERE deptno in
       (select deptno from scott.emp b
        where b.empno = :empno )
```

Assuming that the Consolidator client id is `empno` and the snapshot table is `emp`, then the client sub-query queries for data changes in the `clg$emp` log table as follows:

```
SELECT empno as clid$$cs FROM scott.clg$emp
   UNION SELECT empno as clid$$cs FROM scott.emp
         WHERE deptno in (select deptno from scott.clg$dept)
```

The following example uses the automatic synchronization version of `createPublicationItem` method, which uses the `PubItemProps` class to define all publication item definitions, including automatic synchronization, as follows:

```
PubItemProps pi_props = new PubItemProps();
pi_props.owner = "MASTER";                  // owner schema
pi_props.store = STORES[i][0];              // store
pi_props.refresh_mode = "F"; //default      // uses fast refresh
pi_props.select_stmt =                  // specify select statement for snapshot
   "SELECT * FROM "+"MASTER"+"."+STORES[i][0]+ " WHERE C1 =:CLIENTID";
pi_props.cl2log_rec_stmt = "SELECT base.C1 FROM "   // client sub-query to
      + "MASTER"+"."+STORES[i][0] + " base,"        // return unique clientids
      + "MASTER"+".CLG$"+STORES[i][0] + " log"
      + "  WHERE base.ID = log.ID";
// Setting "isLogBased" to True enables automatic sync for this pub item.
pi_props.isLogBased = true;
// If doChangeNtf is true, automatic publication item sends notifications
// from server about new/modified records
pi_props.doChangeNtf = true;

cm.createPublicationItem(PUBITEMS[i], pi_props);
cm.addPublicationItem(PUB,PUBITEMS[i],null,null,"S",null,null);
```

3.4.1.3.1 Defining Publication Items for Updatable Multi-Table Views Publication items can be defined for both tables and views. When publishing updatable multi-table views, the following restrictions apply:

- The view must contain a parent table with a primary key defined.
- `INSTEAD OF` triggers must be defined for data manipulation language (DML) operations on the view. See Section 3.9, "Understanding Your Refresh Options" for more information.
- All base tables of the view must be published.

3.4.1.4 Define Publication-Level Automatic Synchronization Rules

Once the publication is created, you can create and add automatic synchronization rules that apply to all enabled publication items in this publication. Perform the following to add a rule to a publication:

1. The rule is made up of a rule name and a `String` that contains the rule definition. The rules can be created using the `Rules` classes and `RuleInfo` objects.

   a. Define the rule and convert it to a `String` using the `RuleInfo` object and the `setSyncRuleParams` method.

   ```
   RuleInfo ri = Rules.RULE_MAX_DB_REC_ri;
   ri.params.put(Rules.PARAM_NREC,"5");
   String ruleText = cm.setSyncRuleParams(ri.type,ri.params);
   ```

   There are `RuleInfo` objects for all of the main automatic synchronization rules. So, in order to specify a rule, you obtain the appropriate `RuleInfo` object from the Rules class and then define the variable. Table 3–11, " Automatic Synchronization Rule Info Objects" describe the different types of rules you can specify for triggering automatic synchronization:

   Note: See the Javadoc for examples and the parameters that you need to set for each rule.

Table 3–11 Automatic Synchronization Rule Info Objects

Rule Info Object	Description
`RULE_MAX_DB_REC_ri`	Synchronize if the client database for all publication items on the client contains more than `NREC` modified records, where you specify the `NREC` of modifed records in the client database to trigger an automatic synchronization.
`RULE_NOTIFY_MAX_PUB_REC_ri`	Synchronize if the out queue contains more than `NREC` modified records, where you specify the `NREC` of modifed records in the server database to trigger an automatic synchronization.
`RULE_MAX_PI_REC_ri`	Client automatically synchronizes if the number of modified records for a publication item is greater than `NREC`.
`RULE_HIGH_BANDWIDTH_ri`	Synchronize when the network bandwidth is greater than `<number>` bits/second. Where `<number>` is an integer that indicates the bandwidth bits/seconds. When the bandwidth is at this value, the synchronization occurs.
`RULE_LOW_PWR_ri`	Synchronize when the battery level drops to `<number>`%, where `<number>` is a percentage. Often you may wish to synchronize before you lose battery power.

Table 3–11 (Cont.) Automatic Synchronization Rule Info Objects

Rule Info Object	Description
`RULE_AC_PWR_ri`	Synchronize when the AC power is detected; that is, when the device is plugged in.
`RULE_MIN_MEM_ri`	Specify the minimum battery level required in order for an automatic synchronization to start. The battery level is specified as a percentage.
`RULE_NET_PRIORITY_ri`	Network conditions can be specified using the following properties: data priority, ping delay and network bandwidth.
`RULE_MIN_PWR_ri`	If the battery life drops below a pre-defined minimum, then synchronization is automatically triggered.
`NET_CONFIG_ri`	Configure network parameters (currently only the network specific proxy configuration is supported) The configuration rule contains a vector of hashtables with a hashtable representing properties of each individual network.
`RULE_TIME_INTERVAL_ri`	Schedule sync at a given time of day with a certain frequency (interval). Specify the time (`PARAM_START_TIME`) for an automatic synchronization to start. The format of time is standard date string: H24:MI:SS e.g. 00:00:00 or 23:59:00 The time is GMT. If not set, the synchronization starts when the Sync Agent starts and all other conditions are satisfied Set the period (`PARAM_PERIOD`), in seconds, to specify the frequency of scheduled synchronization events.

b. Define a name for the rule, which should be a name not attached to any particular publication, so you can use the rule for several publications.

2. Create the rule with the `createSyncRule` method, which creates the rule with the name, the `String` containing the rule, and a boolean on whether to replace the rule if it already exists. Once completed, then this rule can be associated with any publication.

```
boolean replace = true;
cm.createSyncRule ( ruleName, ruleText, replace );
```

3. Associate the rule with the desired publication or platform using the `addSyncRule` method. This method can add any existing rule to a designated publication. To add to a publication, use the publication name as the first parameter, as follows:

```
cm.addSyncRule( PUB, ruleName );
```

To add a rule to a client platform—Win32 or WINCE platform—perform the following:

```
cm.addSyncRule( Consolidator.DEFAULT_TEMPLATE_WIN32, rulename );
```

Where the platform name is a constant defined in the Consolidator class as either `DEFAULT_TEMPLATE_WIN32` or `DEFAULT_TEMPLATE_WCE`.

You can also perform the following:

- Section 3.4.1.4.1, "Retrieve All Publications Associated with a Rule"
- Section 3.4.1.4.2, "Retrieve Rule Text"

- Section 3.4.1.4.3, "Check if Rule is Modified"
- Section 3.4.1.4.4, "Remove Rule"

3.4.1.4.1 Retrieve All Publications Associated with a Rule Just as you can with resources, scripts and sequences that are associated with publications, you can retrieve all publications that are associated with a rule with the `getPublicationNames` method. The following retrieves all publications that are associated with the rule within the `ruleName` variable. The object type is defined as `Consolidator.RULES_OBJECT`.

```
String[] pubs  = cm.getPublicationNames ( ruleName , Consolidator.RULES_OBJECT);
```

3.4.1.4.2 Retrieve Rule Text You can retrieve the text of the rule using the `getSyncRule` and providing the rule name. This is useful if you are not sure what the rule is and need to discover the text before associating it with another publication.

```
String retStr = cm.getSyncRule ( ruleName );
```

3.4.1.4.3 Check if Rule is Modified You can compare the rule within the repository with a provided string to see if the rule has been modified with the `isSyncRuleModified` method. A boolean value of true is returned if the provided `ruleText` is different from what exists in the repository.

```
boolean ismod = cm.isSyncRuleModified ( ruleName, ruleText );
```

3.4.1.4.4 Remove Rule You can remove the association of a rule from a publication by using the `removeSyncRule` method. You can delete the entire rule from the repository by using the `dropSyncRule` method. If you drop the rule and it is still associated with one or more publications, the rule is automatically unassociated from these publications.

3.4.1.5 Data Subsetting: Defining Client Subscription Parameters for Publications

Data subsetting is the ability to create specific subsets of data and assign them to a parameter name that can be assigned to a subscribing user. When creating publication items, a parameterized `Select` statement can be defined. Subscription parameters must be specified at the time the publication item is created, and are used during synchronization to control the data published to a specific client.

Creating a Data Subset Example

```
consolidatorManager.createPublicationItem("CORP_DIR1",
   "DIRECTORY1", "ADDRLRL4P", "F" ,
   "SELECT LastName, FirstName, company, phone1, phone2, phone3, phone4,
    phone5, phone1id, phone2id, phone3id, displayphone, address, city, state,
    zipcode, country, title, custom1, custom2, custom3, note
    FROM directory1.addrlrl4p WHERE company = :COMPANY", null, null);
```

In this sample statement, data is being retrieved from a publication named `CORP_DIR1`, and is subset by the variable `COMPANY`.

> **Note:** Within the select statement, the parameter name for the data subset must be prefixed with a colon, for example `:COMPANY`.

When a publication uses data subsetting parameters, set the parameters for each subscription to the publication. For example, in the previous example, the parameter `COMPANY` was used as an input variable to describe what data is returned to the client. You can set the value for this parameter with the `setSubscriptionParameter` method. The following example sets the subscription parameter `COMPANY` for the client `DAVIDL` in the `CORP_DIR1` publication to `DAVECO`:

```
consolidatorManager.setSubscriptionParameter("CORP_DIR1", "DAVIDL",
       "COMPANY", "'DAVECO'");
```

Note: This method should only be used on publications created using the Consolidator Manager API. To create template variables, a similar technique is possible using MDW.

3.4.1.6 Create Publication Item Indexes

The Mobile Server supports automatic deployment of indexes in Oracle Database Lite on clients. The Mobile Server automatically replicates primary key indexes from the server database. The Consolidator Manager API provides calls to explicitly deploy unique, regular, and primary key indexes to clients as well.

By default, the primary key index of a table is automatically replicated from the server. You can create secondary indexes on a publication item. If you do not want the primary index, you must explicitly drop it from the publication items.

If you want to create other indexes on any columns in your application tables, then use the `createPublicationItemIndex` method. The following demonstrates how to set up indexes on the name field in our publication item `P_SAMPLE11-M`:

```
consolidatorManager.createPublicationItemIndex("P_SAMPLE11M-I3",
   "P_SAMPLE11-M", "I", "NAME");
```

An index can contain more than one column. You can define an index with multiple columns, as follows:

```
consolidatorManager.createPublicationItemIndex("P_SAMPLE11D-I1", "P_SAMPLE11-D",
      "I", "KEY,NAME");
```

3.4.1.6.1 Define Client Indexes Client-side indexes can be defined for existing publication items. There are three types of indexes that can be specified:

- P - Primary key
- U - Unique
- I - Regular

Note: When an index of type 'U' or 'P' is defined on a publication item, there is no check for duplicate keys on the server. If the same constraints do not exist on the base object of the publication item, synchronization may fail with a duplicate key violation. See the *Oracle Database Lite API Specification* for more information.

3.4.1.7 Adding Publication Items to Publications

Once you create a publication item, you must associate it with a publication using the `addPublicationItem` method, as follows:

```
consolidatorManager.addPublicationItem("T_SAMPLE11", "P_SAMPLE11-M",
     null, null, "S", null, null);
```

See Section 3.4.1.12, "Modifying a Publication Item" for details on how to change the definition.

3.4.1.7.1 Defining Conflict Rules When adding a publication item to a publication, the user can specify winning rules to resolve synchronization conflicts in favor of either the client or the server. See Section 3.13, "Resolving Conflict Resolution with Winning Rules" for more information.

3.4.1.7.2 Using Table Weight Table weight is an integer associated with publication items that determines in what order the transactions for all publications are processed. For example, if three publication items exist—`emp`, `dept`, `mgr`, you can define the order in which the transactions associated with each publication item are executed. In our example, assign table weight of 1 to `dept`, table weight of 2 to `mgr`, and table weight of 3 to `emp`. In doing this, you ensure that the master table `dept` is always updated first, followed by `mgr`, and lastly by `emp`.

The insert, update, and delete client operations are executed in the following order:

1. Client `INSERT` operations are executed first, from lowest to highest table weight order. This ensures that the master table entries are added before the details table entries.
2. Client `DELETE` operations are executed next, from highest to lowest table weight order. Processing the delete operations ensures that the details table entries are removed before the master table entries.
3. Client `UPDATE` operations are executed last, from highest to lowest table weight order.

In our example with `dept`, `mgr`, and `emp` tables, the execution order would be as follows:

1. All insert operations for `dept` are processed.
2. All insert operations for `mgr` are processed.
3. All insert operations for `emp` are processed.
4. All delete operations for `emp` are processed.
5. All delete operations for `mgr` are processed.
6. All delete operations for `dept` are processed.
7. All update operations for `emp` are processed.
8. All update operations for `mgr` are processed.
9. All update operations for `dept` are processed.

Table weight is applied to publication items within a specific publication; for example, a publication can have more than one publication item of weight 2. In this case, it does not matter which publication is executed first.

Define the order weight for tables when you add a publication item to the publication.

3.4.1.8 Creating Client-Side Sequences for the Downloaded Snapshot

A sequence is a database schema object that generates sequential numbers. After creating a sequence, you can use it to generate unique sequence numbers for transaction processing. These unique integers can include primary key values. If a transaction generates a sequence number, the sequence is incremented immediately whether you commit or roll back the transaction.

If you have more than a single client, you want to assign who gets which sequence numbers, so that when you synchronize, none of the records have duplicate sequence numbers. Thus, if you have multiple clients, then specify a distinct range of numbers for each client, so that they are not using the same numbers.

- Specify a range of values for each client. In our example, client A would be assigned sequence numbers 1 through 100, client B would be assigned sequence numbers 101 to 200, and client C would be assigned sequence numbers 201 through 300. If they ran out of sequence numbers, they are assigned another 100, which is the defined window size in our example, during the next synchronization. Since none of the clients checked to generate server-side sequence, the database, in order to never collide with the sequence numbers, starts its sequence number at -1 and decrements for each subsequent sequence number.
- You could specify that all clients are allowed to have only odd numbers and the database has all even numbers. That is, you could start the client at 1 and increment by 2 for all of its sequence numbers. This enables you to avoid having negative numbers for your sequence numbers. The clients still have a window size, which in this example is 100, but they start with an odd number within that window and always increment by 2 to avoid any positive numbers. Thus, client A would still have the window of 1 to 100, but the sequence numbers would be 1, 3, 5, and so on up to 99.

Thus, for each client that uses sequences, you must define what numbers each client can use through the Consolidator Manager API, which allow you to manage the sequences with methods that create/drop a sequence, add/remove a sequence from a publication, modify a sequence, and advance a sequence window for each user.

Note: The sequence name is case-sensitive.

Once you have created the sequence, you place it into the publication with the publication item to which it applies.

Note: If the sequences do not work properly, check your parent publications. All parent publications must have at least one publication item. If you do not have any publication items for the parent publication, then create a dummy publication item within the parent.

See the *Oracle Database Lite API Specification* (included on the CD) for a complete listing of the APIs to define and administrate sequences.

3.4.1.8.1 Specifying Sequence Threshold for Window Management Oracle Database Lite also allows you to set a threshold. If you know that you need a minimum number of records between synchronizations to perform your work, set this number as the threshold. That way, if you have less than this number available to you, Oracle Database Lite provides the client with a new window to work from.

For example, if a client has a window of 100 and retrieves the first window of 1-100. If the sequence numbers retrieved is currently at record number 97, then—if no threshold is set—the Oracle Database Lite does not provide a new window since this window is not complete. However, if you state that you need at least 20 records to perform your duties, Oracle Database Lite would notice that there are less than 20 records left in the window and assigsn the client the next window, which in this case would be sequence numbers 101-200.

3.4.1.8.2 Description of Sequence Support The following sequence support is available:

- **True sequence support on the client**—The Sync Server supports replication of true sequence objects to the client.
- **Clear association with a publication**—In a manner similar to publication items, adding sequences to a publication propagates the corresponding sequence objects to all subscribing users. Note that a publication and a sequence have a one-to-many relationship. This means a publication can contain many different sequences, but a single sequence cannot exist in more than one publication.
- **Offline and Online**—There are two types of sequences, as follows:
  - Offline: The developer specifies the increment value of the sequence used by the client. The sequence exists solely for the client.
  - Online: An online sequence is designed to support online Web-to-Go applications. This is accomplished by creating the same sequence object on both the server and the client. The paired sequences are incremented by two and started with staggered values; one starts with an even number and one starts with an odd number. By using an odd/even window model such as the one described above, the Consolidator Manager ensures uniqueness—regardless of whether the application is running while connected to the back-end Oracle database or not.
- **Sequence management** - Once the sequences have been defined and associated with a publication, the Sync Server manages all aspects of administration for sequences, including allocation of new windows.

3.4.1.9 Subscribing Users to a Publication

Subscribe the users to a publication using the `createSubscription` function. The following creates a subscription between the `S11U1` user and the `T_SAMPLE11` publication:

```
consolidatorManager.createSubscription("T_SAMPLE11", "S11U1");
```

3.4.1.10 Instantiate the Subscription

After you subscribe a user to a publication, you complete the subscription process by instantiating the subscription, which associates the user with the publication in the back-end database. The next time that the user synchronizes, the data snapshot from the publication is provided to the user.

```
consolidatorManager.instantiateSubscription("T_SAMPLE11", "S11U1");

//Close the connection.
consolidatorManager.closeConnection();
```

> **Note:** If you need to set subscription parameters for data subsetting, this must be completed before instantiating the subscription. See Section 3.4.1.5, "Data Subsetting: Defining Client Subscription Parameters for Publications" for more information.

3.4.1.11 Bringing the Data From the Subscription Down to the Client

You can perform the synchronization and bring down the data from the subscription you just created. The client executes SQL queries against the client ODB to retrieve any information. This subscription is not associated with any application, as it was created using the low-level Consolidator Manager APIs.

3.4.1.12 Modifying a Publication Item

You can add additional columns to existing publication items. These new columns are pushed to all subscribing clients the next time they synchronize. This is accomplished through a complete refresh of all changed publication items.

- An administrator can add multiple columns, modify the WHERE clause, add new parameters, and change data type.
- This feature is supported for all Mobile client platforms.
- The client does not upload snapshot information to the server. This also means the client cannot change snapshots directly on the client database, for example, you could not alter a table using Mobile SQL.
- Publication item upgrades will be deferred during high priority synchronizations. This is necessary for low bandwidth networks, such as wireless, because all publication item upgrades require a complete refresh of changed publication items. While the high priority flag is set, high priority clients will continue to receive the old publication item format.
- The server needs to support a maximum of two versions of the publication item which has been altered.

To change the definition, use one of the following:

- If the publication item is read-only, then modify the publication item either with the `reCreatePublicationItem` method or by dropping and creating the publication item with the `dropPublicationItem` and `createPublicationItem` APIs.
- If the publication item is updatable, then you can use the `alterPublicationItem` method. This method enables a smooth transition of changing any table structure on both the client and the server for updatable publications.

  If you use the `alterPublicationItem` method, you must follow it up by executing the `resetCache` method. The metadata cache should be reset every time a change is made to the publication or publication items. If you make the change though Mobile Manager, then the Mobile Manager calls the `resetCache` method. You can reset the metadata cache from the Mobile Manager or execute the `resetCache` method, part of the `ConsolidatorManager` class.

  You may use the `alterPublicationItem` method for schema evolution to add columns to an existing publication item. The WHERE clause may also be altered. If additional parameters are added to the WHERE clause, then these parameters must be set before the alter occurs. See the `setSubscriptionParams` method. However, if you are creating a fast refresh publication item on a table with a composite primary key, the snapshot query must match the primary key columns in the order that they are present in the table definition. This automatically happens during the column selection when MDW is used or when a SELECT * query is used. Note that the order of the primary key columns in the table definition may be different from those in the primary key constraint definition.

  ```
  consolidatorManager.alterPublicationItem("P_SAMEPLE1", "select * from EMP");
  ```

  Note: If the select statement does not change, then the call to the `alterPublicationItem()` method has no effect.

See Section 3.15, "Facilitating Schema Evolution" and the `alterPublicationItem` method definition in the *Oracle Database Lite API Specification* for more information.

3.4.1.13 Callback Customization for DML Operations

Once a publication item has been created, a user can use the Consolidator Manager API to specify a customized PL/SQL procedure that is stored in the Mobile Server repository to be called in place of all DML operations for that publication item. There can be only one Mobile DML procedure for each publication item. The procedure should be created as follows:

```
AnySchema.AnyPackage.AnyName(DML in CHAR(1), COL1 in TYPE, COL2 in TYPE, COLn..,
PK1 in TYPE, PK2 in TYPE, PKn..)
```

The parameters for customizing a DML operation are listed in Table 3–12:

Table 3–12 Mobile DML Operation Parameters

Parameter	Description
`DML`	DML operation for each row. Values can be "D" for DELETE, "I" for INSERT, or "U" for UPDATE.
`COL1 ... COLn`	List of columns defined in the publication item. The column names must be specified in the same order that they appear n the publication item query. If the publication item was created with "SELECT * FROM exp", the column order must be the same as they appear in the table "exp".
PK1 ... PKn	List of primary key columns. The column names must be specified in the same order that they appear in the base or parent table.

The following defines a DML procedure for publication item `exp`:

```
select A,B,C from publication_item_exp_table
```

Assuming `A` is the primary key column for `exp`, then your DML procedure would have the following signature:

```
any_schema.any_package.any_name(DML in CHAR(1), A in TYPE, B in TYPE, C
                        in TYPE,A_OLD in TYPE)
```

During runtime, this procedure is invoked with 'I', 'U', or 'D' as the DML type. For insert and delete operations, `A_OLD` will be null. In the case of updates, it will be set to the primary key of the row that is being updated. Once the PL/SQL procedure is defined, it can be attached to the publication item through the following API call:

```
consolidatorManager.addMobileDmlProcedure("PUB_exp","exp",
                                           "any_schema.any_package.any_name")
```

where `exp` is the publication item name and `PUB_exp` is the publication name.

Refer to the *Oracle Database Lite API Specification* for more information.

3.4.1.13.1 DML Procedure Example The following piece of PL/SQL code defines an actual DML procedure for a publication item in one of the sample publications. As described below, the `ORD_MASTER` table. The query was defined as:

```
SELECT * FROM "ord_master", where ord_master has a single column primary key
             on "ID"
```

ord_master Table

```
SQL> desc ord_master
Name                                      Null?    Type
----------------------------------------- -------- -------------
ID                                        NOT NULL NUMBER(9)
DDATE                                              DATE
STATUS                                             NUMBER(9)
NAME                                               VARCHAR2(20)
DESCRIPTION                                        VARCHAR2(20)
```

Code Example

```
CREATE OR REPLACE  PACKAGE "SAMPLE11"."ORD_UPDATE_PKG"  AS
 procedure  UPDATE_ORD_MASTER(DML CHAR,ID NUMBER,DDATE DATE,STATUS
NUMBER,NAME VARCHAR2,DESCRIPTION VARCHAR2, ID_OLD NUMBER);
END ORD_UPDATE_PKG;
/
CREATE OR REPLACE  PACKAGE BODY "SAMPLE11"."ORD_UPDATE_PKG" as
  procedure  UPDATE_ORD_MASTER(DML CHAR,ID NUMBER,DDATE DATE,STATUS
NUMBER,NAME VARCHAR2,DESCRIPTION VARCHAR2, ID_OLD NUMBER) is
  begin
    if DML = 'U' then
     execute immediate 'update ord_master set id = :id, ddate = :ddate,
status = :status, name = :name, description = '||''''||'from
ord_update_pkg'||''''||' where id = :id_old'
      using id,ddate,status,name,id_old;
    end if;
    if DML = 'I' then
 begin
      execute immediate 'insert into ord_master values(:id, :ddate,
:status, :name, '||''''||'from ord_update_pkg'||''''||')'
        using id,ddate,status,name;
 exception
  when others then
   null;
 end;
    end if;
    if DML = 'D' then
     execute immediate 'delete from ord_master where id = :id'
      using id;
    end if;
  end UPDATE_ORD_MASTER;
end ORD_UPDATE_PKG;
/
```

The API call to add this DML procedure is as follows:

```
consolidatorManager.addMobileDMLProcedure("T_SAMPLE11",
          "P_SAMPLE11-M","SAMPLE11.ORD_UPDATE_PKG.UPDATE_ORD_MASTER")
```

where `T_SAMPLE11` is the publication name and `P_SAMPLE11-M` is the publication item name.

3.4.1.14 Restricting Predicate

A restricting predicate can be assigned to a publication item as it is added to a publication.The predicate is used to limit data downloaded to the client. The parameter, which is for advanced use, can be null. When using a restricting predicate, the synchronization uses the high priority replication mode. For using a restricting predicate, see Section 1.2.5 "Priority-Based Replication" in the *Oracle Database Lite Troubleshooting and Tuning Guide*.

3.5 Client Device Database DDL Operations

The first time a client synchronizes, Oracle Database Lite automatically creates the Oracle Lite database with the snapshot tables for the user subscriptions on the Mobile client. If you would like to execute additional DDL statements on the database, then add the DDL statements as part of your publication. Oracle Database Lite executes these DDL statements when the user synchronizes.

This is typically used for adding constraints and check values.

For example, you can add a foreign key constraint to a publication item. In this instance, if the Oracle Database Lite created snapshots S1 and S2 during the initial synchronization, where the definition of S1 and S2 are as follows:

```
S1 (C1 NUMBER PRIMARY KEY, C2 VARCHAR2(100), C3 NUMBER);
S2 (C1 NUMBER PRIMARY KEY, C2 VARCHAR2(100), C3 NUMBER);
```

If you would like to create a foreign key constraint between C3 on S2 and the primary key of S1 , then add the following DDL statement to your publication item:

```
ALTER TABLE S2
   ADD CONSTRAINT S2_FK FOREIGN KEY (C3)
   REFERENCES S1 (C1);
```

Then, Oracle Database Lite executes any DDL statements after the snapshot creation or, if the snapshot has already been created, after the next synchronization.

See the *Oracle Database Lite API Specification* for more information on these APIs.

3.6 Customize the Compose Phase Using MyCompose

The compose phase takes a query for one or more server-side base tables and puts the generated DML operations for the publication item into the Out Queue to be downloaded into the client. The Consolidator Manager manages all DML operations using the physical DML logs on the server-side base tables. This can be resource intensive if the DML operations are complex—for example, if there are complex data-subsetting queries being used. The tools to customize this process include an extendable `MyCompose` with compose methods which can be overridden, and additional Consolidator Manager APIs to register and load the customized class.

When you want to customize the compose phase of the synchronization process, you must perform the following:

1. Section 3.6.1, "Create a Class That Extends MyCompose to Perform the Compose"
2. Section 3.6.2, "Implement the Extended MyCompose Methods in the User-Defined Class"
3. Section 3.6.3, "Use Get Methods to Retrieve Information You Need in the User-Defined Compose Class"
4. Section 3.6.4, "Register the User-Defined Class With the Publication Item"

3.6.1 Create a Class That Extends MyCompose to Perform the Compose

The `MyCompose` class is an abstract class, which serves as the super-class for creating a user-written sub-class, as follows:

```
public class ItemACompose extends oracle.lite.sync.MyCompose
{
...
}
```

All user-written classes—such as `ItemACompose`—produce publication item DML operations to be sent to a client device by interpreting the base table DML logs. The sub-class is registered with the publication item, and takes over all compose phase operations for that publication item. The sub-class can be registered with more than one publication item—if it is generic—however, internally the Composer makes each instance of the extended class unique within each publication item.

3.6.2 Implement the Extended MyCompose Methods in the User-Defined Class

The `MyCompose` class includes the following methods—`needCompose`, `doCompose`, `init`, and `destroy`—which are used to customize the compose phase. One or more of these methods can be overridden in the sub-class to customize compose phase operations. Most users customize the compose phase for a single client. In this case, only implement the `doCompose` and `needCompose` methods. The `init` and `destroy` methods are only used when a process is performed for all clients, either before or after individual client processing.

The following sections describe how to implement these methods:

- Section 3.6.2.1, "Implement the needCompose Method"
- Section 3.6.2.2, "Implement the doCompose Method"
- Section 3.6.2.3, "Implement the init Method"
- Section 3.6.2.4, "Implement the destroy Method"

> **Note:** To retrieve information, use the methods described in Section 3.6.3, "Use Get Methods to Retrieve Information You Need in the User-Defined Compose Class".

3.6.2.1 Implement the needCompose Method

The `needCompose` method to identifies a client that has changes to a specific publication item that is to be downloaded. Use this method as a way to trigger the `doCompose` method.

```
public int needCompose(Connection conn,
  String clientid) throws Throwable
```

The parameters for the `needCompose` method are listed in Table 3–13:

Table 3–13 needCompose Parameters

Parameter	Definition
`conn`	Database connection to the Mobile Server repository.
`clientid`	Specifies the client that is connecting to the database.

The following example examines a client base table for changes—in this case, the presence of dirty records. If there are changes, then the method returns `MyCompose.YES`, which triggers the `doCompose` method.

```
    public int needCompose(String clientid) throws Throwable{
        boolean baseDirty = false;
        String [][] baseTables = this.getBaseTables();

        for(int i = 0; i < baseTables.length; i++){
            if(this.baseTableDirty(baseTables[i][0], baseTables[i][1])){
                baseDirty = true;
                break;
            }
        }

        if(baseDirty){
            return MyCompose.YES;
        }else{
            return MyCompose.NO;
        }
    }
```

This sample uses subsidiary methods discussed in Section 3.6.3, "Use Get Methods to Retrieve Information You Need in the User-Defined Compose Class" to check if the publication item has any tables with changes that need to be sent to the client. In this example, the base tables are retrieved, then checked for changed, or dirty, records. If the result of that test is true, a value of Yes is returned, which triggers the call for the `doCompose` method.

3.6.2.2 Implement the doCompose Method

The `doCompose` method populates the DML log table for a specific publication item, which is subscribed to by a client.

```
public int doCompose(Connection conn,
   String clientid) throws Throwable
```

The parameters for the `doCompose` method are listed in Table 3–14:

Table 3–14 doCompose Parameters

Parameter	Definition
`conn`	Database connection to the Mobile Server repository.
`clientid`	Specifies the client that is connecting to the database.

The following example contains a publication item with only one base table where a DML (Insert, Update, or Delete) operation on the base table is performed on the publication item. This method is called for each client subscribed to the publication item.

```
 public int doCompose(Connection conn, String clientid) throws Throwable {
        int rowCount = 0;

        String [][] baseTables = this.getBaseTables();
        String baseTableDMLLogName =
            this.getBaseTableDMLLogName(baseTables[0][0], baseTables[0][1]);
        String baseTablePK =
            this.getBaseTablePK(baseTables[0][0],baseTables[0][1]);
        String pubItemDMLTableName = this.getPubItemDMLTableName();

        String sql = "INSERT INTO " + pubItemDMLTableName
            + " SELECT " +  baseTablePK + ", DMLTYPE$$ FROM " +
             baseTableDMLLogName;
        Statement st = conn.createStatement();
        rowCount = st.executeUpdate(sql);
        st.close();
        return rowCount;
    }
```

This code uses subsidiary methods discussed in Section 3.6.3, "Use Get Methods to Retrieve Information You Need in the User-Defined Compose Class" to create a SQL statement. The MyCompose method retrieves the base table, the base table primary key, the base table DML log name and the publication item DML table name using the appropriate get methods. You can use the table names and other information returned by these methods to create a dynamic SQL statement, which performs an insert into the publication item DML table of the contents of the base table primary key and DML operation from the base table DML log.

3.6.2.3 Implement the init Method

The init method provides the framework for user-created compose preparation processes. The init method is called once for all clients prior to the individual client compose phase. The default implementation has no effect.

```
public void init(Connection conn)
```

The parameter for the init method is described in Table 3–15:

Table 3–15 init Parameters

Parameter	Definition
conn	Database connection to the Mobile Server repository.

3.6.2.4 Implement the destroy Method

The destroy method provides the framework for compose cleanup processes. The destroy method is called once for all clients after to the individual client compose phase. The default implementation has no effect.

```
public void destroy(Connection conn)
```

The parameter for the destroy method is described in Table 3–16:

Table 3–16 destroy Parameters

Parameter	Definition
conn	Database connection to the Mobile Server repository.

3.6.3 Use Get Methods to Retrieve Information You Need in the User-Defined Compose Class

The following methods return information for use by primary MyCompose methods.

- Section 3.6.3.1, "Retrieve the Publication Name With the getPublication Method"
- Section 3.6.3.2, "Retrieve the Publication Item Name With the getPublicationItem Method"
- Section 3.6.3.3, "Retrieve the DML Table Name With the getPubItemDMLTableName Method"
- Section 3.6.3.4, "Retrieve the Primary Key With the getPubItemPK Method"

- Section 3.6.3.5, "Retrieve All Base Tables With the getBaseTables Method"
- Section 3.6.3.6, "Retrieve the Primary Key With the getBaseTablePK Method"
- Section 3.6.3.7, "Discover If Base Table Has Changed With the baseTableDirty Method"
- Section 3.6.3.8, "Retrieve the Name for DML Log Table With the getBaseTableDMLLogName Method"
- Section 3.6.3.9, "Retrieve View of the Map Table With the getMapView Method"

3.6.3.1 Retrieve the Publication Name With the getPublication Method

The `getPublication` method returns the name of the publication.

```
public String getPublication()
```

3.6.3.2 Retrieve the Publication Item Name With the getPublicationItem Method

The `getPublicationItem` method returns the publication item name.

```
public String getPublicationItem()
```

3.6.3.3 Retrieve the DML Table Name With the getPubItemDMLTableName Method

The `getPubItemDMLTableName` method returns the name of the DML table or DML table view, including schema name, which the `doCompose` or `init` methods are supposed to insert into.

```
public String getPubItemDMLTableName()
```

You can embed the returned value into dynamic SQL statements. The table or view structure is as follows:

```
<PubItem PK> DMLTYPE$$
```

The parameters for `getPubItemDMLTableName` are listed in Table 3–17:

Table 3–17 getPubItemDMLTableName View Structure Parameters

Parameter	Definition
`PubItemPK`	The value returned by `getPubItemPK()`
`DMLTYPE$$`	This can have the values 'I' for insert, 'D' for delete, or 'U' for Update.

3.6.3.4 Retrieve the Primary Key With the getPubItemPK Method

Returns the primary key for the listed publication in comma separated format in the form of `<col1>,<col2>,<col3>`.

```
public String getPubItemPK() throws Throwable
```

3.6.3.5 Retrieve All Base Tables With the getBaseTables Method

Returns all the base tables for the publication item in an array of two-string arrays. Each two-string array contains the base table schema and name. The parent table is always the first base table returned, in other words, `baseTables[0]`.

```
public string [][] getBaseTables() throws Throwable
```

3.6.3.6 Retrieve the Primary Key With the getBaseTablePK Method

Returns the primary key for the listed base table in comma separated format, in the form of `<col1>, col2>,<col3>`.

```
public String getBaseTablePK (String owner, String baseTable) throws Throwable
```

The parameters for `getBaseTablePK` are listed in Table 3–18:

Table 3–18 getBaseTablePK Parameters

Parameter	Definition
`owner`	The schema name of the base table owner.
`baseTable`	The base table name.

3.6.3.7 Discover If Base Table Has Changed With the baseTableDirty Method

Returns the a boolean value for whether or not the base table has changes to be synchronized.

```
public boolean baseTableDirty(String owner, String store)
```

The parameters for `baseTableDirty` are listed in Table 3–19:

Table 3–19 baseTableDirty Parameters

Parameter	Definition
`owner`	The schema name of the base table.
`store`	The base table name.

3.6.3.8 Retrieve the Name for DML Log Table With the getBaseTableDMLLogName Method

Returns the name for the physical DML log table or DML log table view for a base table.

```
public string getBaseTableDMLLogName(String owner, String baseTable)
```

The parameters for `getBaseTableDMLLogName` are listed in Table 3–20:

Table 3–20 getBaseTableDMLLogName Parameters

Parameter	Definition
`owner`	The schema name of the base table owner.
`baseTable`	The base table name.

You can embed the returned value into dynamic SQL statements. There may be multiple physical logs if the publication item has multiple base tables. The parent base table physical primary key corresponds to the primary key of the publication item. The structure of the log is as follows:

```
<Base Table PK> DMLTYPE$$
```

The parameters for `getBaseTableDMLLogName` view structure are listed in Table 3–21:

Table 3–21 getBaseTableDMLLogName View Structure Parameters

Parameter	Definition
`Base Table PK`	The primary key of the parent base table.
`DMLTYPE$$`	This can have the values 'I' for insert, 'D' for delete, or 'U' for Update.

3.6.3.9 Retrieve View of the Map Table With the getMapView Method

Returns a view of the map table which can be used in a dynamic SQL statement and contains a primary key list for each client device. The view can be an inline view.

```
public String getMapView() throws Throwable
```

The structure of the map table view is as follows:

```
CLID$$CS <Pub Item PK> DMLTYPE$$
```

The parameters of the map table view are listed in Table 3–22:

Table 3–22 getMapView View Structure Parameters

Parameter	Definition
`CLID$$CS`	This is the client ID column.
`Base Table PK`	The primary key columns of the publication item.
`DMLTYPE$$`	This can have the values 'I' for insert, 'D' for delete, or 'U' for Update.

3.6.4 Register the User-Defined Class With the Publication Item

Once you have created your sub-class, it must be registered with a publication item. The Consolidator Manager API now has two methods `registerMyCompose` and `deRegisterMyCompose` to permit adding and removing the sub-class from a publication item.

- The `registerMyCompose` method registers the sub-class and loads it into the Mobile Server repository, including the class byte code. By loading the code into the repository, the sub-class can be used without having to be loaded at runtime.
- The `deRegisterMyCompose` method removes the sub-class from the Mobile Server repository.

3.7 Customize What Occurs Before and After Synchronization Phases

You can customize what happens before and after certain synchronization processes by creating one or more PL/SQL packages. The following sections detail the different options you have for customization:

- Section 3.7.1, "Customize What Occurs Before and After Every Phase of Each Synchronization"
- Section 3.7.2, "Customize What Occurs Before and After Compose/Apply Phases for a Single Publication Item"

3.7.1 Customize What Occurs Before and After Every Phase of Each Synchronization

You can customize the MGP phase of the synchronization process through a set of predefined callback methods that add functionality to be executed before or after certain phases of the synchronization process. These callback methods are defined in the `CUSTOMIZE` PL/SQL package. Note that these callback methods are called before or after the defined phase for every publication item.

Note: If you want to customize certain activity for only a specific publication item, see Section 3.7.2, "Customize What Occurs Before and After Compose/Apply Phases for a Single Publication Item" for more information.

Manually create this package in the Mobile Server repository. The methods and their respective calling sequence are as follows:

Note: Some of the procedures in the package are invoked for each client defined in your Mobile Server, such as the `BeforeClientCompose` and `AfterClientCompose` methods.

- Section 3.7.1.1, "NullSync"
- Section 3.7.1.2, "BeforeProcessApply"
- Section 3.7.1.3, "AfterProcessApply"
- Section 3.7.1.4, "BeforeProcessCompose"
- Section 3.7.1.5, "AfterProcessCompose"
- Section 3.7.1.6, "BeforeProcessLogs"
- Section 3.7.1.7, "AfterProcessLogs"
- Section 3.7.1.8, "BeforeClientCompose"
- Section 3.7.1.9, "AfterClientCompose"
- Section 3.7.1.10, "Example Using the Customize Package"
- Section 3.7.1.11, "Error Handling For CUSTOMIZE Package"

3.7.1.1 NullSync

The `NullSync` procedure is called at the beginning of every synchronization session. It can be used to determine whether or not a particular user is uploading data.

```
procedure NullSync (clientid varchar2, isNullSync boolean);
```

3.7.1.2 BeforeProcessApply

The `BeforeProcessApply` procedure is called before the entire apply phase of the MGP process.

```
procedure BeforeProcessApply;
```

3.7.1.3 AfterProcessApply

The `AfterProcessApply` procedure is called after the entire apply phase of the MGP process.

```
procedure AfterProcessApply;
```

3.7.1.4 BeforeProcessCompose

The `BeforeProcessCompose` procedure is called before the entire compose phase of the MGP process.

```
procedure BeforeProcessCompose;
```

3.7.1.5 AfterProcessCompose

The `AfterProcessCompose` procedure is called after the entire compose phase of the MGP process.

```
procedure AfterProcessCompose;
```

3.7.1.6 BeforeProcessLogs

The `BeforeProcessLogs` procedure is called before the database log tables (CLG$) are generated for the compose phase of the MGP process. This log tables capture changes for MGP and should not be confused with the trace logs.

```
procedure BeforeProcessLogs;
```

3.7.1.7 AfterProcessLogs

The `AfterProcessLogs` procedure is called after the database log tables (CLG$) are generated for the compose phase of the MGP process. This log tables capture changes for MGP and should not be confused with the trace logs.

```
procedure AfterProcessLogs;
```

3.7.1.8 BeforeClientCompose

The `BeforeClientCompose` procedure is called before each user is composed during the compose phase of the MGP process.

```
procedure BeforeClientCompose (clientid varchar2);
```

3.7.1.9 AfterClientCompose

The `AfterClientCompose` procedure is called after each user is composed during the compose phase of the MGP process.

```
procedure AfterClientCompose (clientid varchar2);
```

3.7.1.10 Example Using the Customize Package

If a developer wants to use any of the procedures listed above, perform the following:

- Manually create the `CUSTOMIZE` package in the Mobile Server schema.
- Define all of the methods with the following specification:

```
create or replace package CUSTOMIZE as
    procedure NullSync (clientid varchar2, isNullSync boolean);
    procedure BeforeProcessApply ;
    procedure AfterProcessApply ;
    procedure BeforeProcessCompose ;
    procedure AfterProcessCompose ;
    procedure BeforeProcessLogs ;
    procedure AfterProcessLogs ;
    procedure BeforeClientCompose(clientid varchar2);
    procedure AfterClientCompose(clientid varchar2);
    end CUSTOMIZE;
```

WARNING: It is the developer's responsibility to ensure that the package is defined properly and that the logic contained does not jeopardize the integrity of the synchronization process.

3.7.1.11 Error Handling For CUSTOMIZE Package

Errors are logged for the `CUSTOMIZE` package only if logging is enabled for the MGP component for the finest level for all event types. Thus, you should set the logging level to `ALL` and the type to `ALL`.

If any errors occur due to an invalid `CUSTOMIZE` package, they are logged only on the first MGP cycle after the Mobile Server restarts. On subsequent synchronizations, the errors are not re-written to the logs, sine the MGP does not attempt to re-execute the `CUSTOMIZE` package until the Mobile Server is restarted.

Note: One requirement is that the `CUSTOMIZE` package can only be executed as user `mobileadmin`.

To locate these errors easily within the `MGP_<x>.log` files, search for the `MGP.callBoundCallBack` method. Another option is to restart the Mobile Server and check the MGP log right after the next synchronization.

3.7.2 Customize What Occurs Before and After Compose/Apply Phases for a Single Publication Item

When creating publication items, the user can define a customizable PL/SQL package that MGP calls during the Apply and Compose phase of the MGP background process **for that particular publication item**. To customize the compose/apply phases for a publication item, perform the following:

1. Create the PL/SQL package with the customized before/after procedures.
2. Register this PL/SQL package with the publication item.

Then when the publication item is being processed, MGP calls the appropriate procedures from your package.

Client data is accumulated in the in queue prior to being processed by the MGP. Once processed by the MGP, data is accumulated in the out queue before being pulled to the client by Mobile Sync.

You can implement the following PL/SQL procedures to incorporate customized code into the MGP process. The `clientname` and `tranid` are passed to allow for customization at the user and transaction level.

- The `BeforeApply` method is invoked before the client data is applied:

  ```
  procedure BeforeApply(clientname varchar2)
  ```

- The `AfterApply` method is invoked after all client data is applied.

  ```
  procedure AfterApply(clientname varchar2)
  ```

- The `BeforeTranApply` method is invoked before the client data with `tranid` is applied.

  ```
  procedure BeforeTranApply(tranid number)
  ```

- The `AfterTranApply` method is invoked after all client data with `tranid` is applied.

  ```
  procedure AfterTranApply(tranid number)
  ```

- The `BeforeCompose` method is invoked before the out queue is composed.

  ```
  procedure BeforeCompose(clientname varchar2)
  ```

- The `AfterCompose` method is invoked after the out queue is composed.

  ```
  procedure AfterCompose(clientname varchar2)
  ```

The following is a PL/SQL example that creates a callback package and registers it when creating the `P_SAMPLE3` publication item. The `BeforeApply` procedure disables constraints before the apply phase; the `AfterApply` procedure enables these constraints. Even though you are only creating procedures for the before and after apply phase of the MGP process, you still have to provide empty procedures for the other parts of the MGP process.

1. Create PL/SQL package declaration with callback owner/schema name of `SAMPLE3` and callback package name of `SAMP3_PKG`.
2. Create the package definition, with all MGP process procedures with callback owner.callback package name of `SAMPLE3.SAMP3_PKG`. Provide a null procedure for any procedure you do not want to modify.
3. Register the package as the callback package for the `SAMPLE3` publication item. If you are creating the publication item, provide the callback schema/owner and the callback package names as input parameters to the createPublicationItem method. If you want to add the callback package to an existing publication item, do the following:
   a. Retrieve the template metadata with `getTemplateItemMetaData` for the publication item.
   b. Modify the attributes that specify the callback owner/schema (`cbk_owner`) and the callback package (`cbk_name`).
   c. Register the package by executing the `setTemplateItemMetaData` method.

```
// create package declaration
  stmt.executeUpdate("CREATE OR REPLACE PACKAGE SAMPLE3.SAMP3_PKG as"
  + " procedure BeforeCompose(clientname varchar2);"
  + " procedure AfterCompose(clientname varchar2);"
  + " procedure BeforeApply(clientname varchar2);"
  + " procedure AfterApply(clientname varchar2);"
  + " procedure BeforeTranApply(tranid number);"
  + " procedure AfterTranApply(tranid number);"
  + " end;"
  );
// create package definition
  stmt.executeUpdate("CREATE OR REPLACE PACKAGE body SAMPLE3.SAMP3_PKG as"
  + " procedure BeforeTranApply(tranid number) is"
  + " begin"
    + " null;"
  + " end;"
  + " procedure AfterTranApply(tranid number) is"
  + " begin"
    + " null;"
  + " end;"
  + " procedure BeforeCompose(clientname varchar2) is"
  + " begin"
```

```
      + "   null;"
  + " end;"
  + " procedure AfterCompose(clientname varchar2) is"
  + " begin"
  + "   null;"
  + " end;"
  + " procedure BeforeApply(clientname varchar2) is"
  + "   cur integer;"
  + "   ign integer;"
  + "   begin"
  + "     cur := dbms_sql.open_cursor;"
  + "     dbms_sql.parse(cur,'SET CONSTRAINT SAMPLE3.address14_fk DEFERRED',
                           dbms_sql.native);"
  + "     ign := dbms_sql.execute(cur);"
  + "     dbms_sql.close_cursor(cur);"
  + "   end;"
  + " procedure AfterApply(clientname varchar2) is"
  + "   cur integer;"
  + "   ign integer;"
  + "   begin"
  + "     cur := dbms_sql.open_cursor;"
  + "     dbms_sql.parse(cur, 'SET CONSTRAINT SAMPLE3.address14_fk IMMEDIATE',
                            dbms_sql.native);"
  + "     ign := dbms_sql.execute(cur);"
  + "     dbms_sql.close_cursor(cur);"
  + "   end;"
  + " end;"
  );
```

Then, register the callback package with the `createPublicationItem` method call, as follows:

```
// register SAMPLE3.SAMP3_PKG as the callback for MGP processing of
// P_SAMPLE3 publication item.

cm.createPublicationItem("P_SAMPLE3","SAMPLE3","ADDRESS", "F",
    "SELECT * FROM SAMPLE3.ADDRESS", "SAMPLE3", "SAMP3_PKG");
```

In the previous code example, the following is required:

- `stmt`, which is used when creating the package definition, is an instance of `java.sql.Statement`
- `cm`, which is used when registering the callback package, is an instance of `oracle.lite.sync.ConsolidatorManager`
- The callback package must have the following procedures defined:
  - `BeforeCompose (clientname varchar2);`
  - `AfterCompose (clientname varchar2);`
  - `BeforeApply (clientname varchar2);`
  - `AfterApply (clientname varchar2);`
  - `BeforeTranApply (tranid number);`
  - `AfterTranApply (tranid number);`

3.8 Initiating Client Synchronization With Synchronization APIs

You can modify the client-side application to start the synchronization programmatically. This section describes how to perform the synchronization upload and download phases for the client using the Synchronization APIs.

Note: Currently, there are no APIs to perform the upload activity on the UNIX platforms.

To execute the upload portion of synchronization from the client (see steps 1 and 2 in Figure 3–1) from within your C, C++, or Java application, perform the following steps:

1. Initialize the synchronization parameters.
2. Set up the transport parameters.
3. Initialize the synchronization options and environment, such as username, password, and selective synchronization.
4. Perform the synchronization.

The following sections demonstrates how you can perform these steps in each of the allowed programming languages:

- Section 3.8.1, "Starting Synchronization Upload and Download Phases With C or C++ Applications"
- Section 3.8.2, "Starting Synchronization Upload and Download Phases With Java Applications"
- Section 3.8.3, "Starting Synchronization Upload and Download Phases With the ADO.NET Provider"

3.8.1 Starting Synchronization Upload and Download Phases With C or C++ Applications

You can initiate and monitor synchronization from a C or C++ client application. The synchronization methods for the C/C++ interface are contained in `ocapi.h` and `ocapi.dll`, which are located in the <*ORACLE_HOME*>`\Mobile\bin` directory. See Section 4.1, "Synchronization APIs For C or C++ Applications" for full details.

3.8.2 Starting Synchronization Upload and Download Phases With Java Applications

You can initiate and monitor synchronization from a Java client application. See Section 4.2, "Synchronization API for Java Applications" for more information.

3.8.3 Starting Synchronization Upload and Download Phases With the ADO.NET Provider

You can initiate and monitor synchronization from an ADO.NET provider application. See Section 13.1.6, "Data Synchronization With the OracleSync Class" for full details.

3.9 Understanding Your Refresh Options

The Mobile Server supports several refresh options. During a fast refresh, incremental changes are synchronized. However, during a complete refresh, all data is refreshed with current data. The refresh mode is established when you create the publication item using the `createPublicationItem` API call. In order to change the refresh mode, first drop the publication item and recreate it with the appropriate mode.

The following sections describe the types of refresh for your publication item that can be used to define how to synchronize:

- Fast Refresh: The most common method of synchronization is a fast refresh publication item where changes are uploaded by the client, and changes for the client are downloaded. Meanwhile, the MGP periodically collects the changes uploaded by all clients and applies them to database tables. It then composes new data, ready to be downloaded to each client during the next synchronization, based on predefined subscriptions.
- Complete Refresh: During a complete refresh, all data for a publication is downloaded to the client. For example, during the very first synchronization session, all data on the client is refreshed from the Oracle database. This form of synchronization takes longer because all rows that qualify for a subscription are transferred to the client device, regardless of existing client data.
- Queue-Based: The developer creates their own queues to handle the synchronization data transfer. This can be considered the most basic form of publication item, for the simple reason that there is no synchronization logic created with it. The synchronization logic is left entirely in the hands of the developer. A queue-based publication item is ideally suited for scenarios that do not require actual synchronization but require something somewhere in between. For instance, data collection on the client. With data collection, there is no need to worry about conflict detection, client state information, or server-side updates. Therefore, there is no need to add the additional overhead normally associated with a fast refresh or complete refresh publication item.
- Forced Refresh: This is actually NOT a refresh option; however, we discuss it here in order to inform you of the consequences of performing a forced refresh. When a Forced Refresh is initiated all data on the client is removed. The client will then bring 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.

The following sections describe the refresh options in more detail:

- Section 3.9.1, "Fast Refresh"
- Section 3.9.2, "Complete Refresh for Views"
- Section 3.9.3, "Queue-Based Refresh"
- Section 3.9.4, "Forced Refresh"

3.9.1 Fast Refresh

Publication items are created for fast refresh by default. Under fast refresh, only incremental changes are replicated. The advantages of fast refresh are reduced overhead and increased speed when replicating data stores with large amounts of data where there are limited changes between synchronization sessions.

The Mobile Server performs a fast refresh of a view if the view meets the following criteria:

- Each of the view base tables must have a primary key.
- All primary keys from all base tables must be included in the view column list.

- If the item is a view, and the item predicate involves multiple tables, then all tables contained in the predicate definition must have primary keys and must have corresponding publication items.

The view requires only a unique primary key for the parent table. The primary keys of other tables may be duplicated. For each base table primary key column, you must provide the Mobile Server with a hint about the column name in the view. You can accomplish this by using the `primaryKeyHint` method of the Consolidator Manager object. See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.9.2 Complete Refresh for Views

A complete refresh is simply a complete execution of the snapshot query. When application synchronization performance is slow, tune the snapshot query. Complete refresh is not optimized for performance. Therefore, to improve performance, use the fast refresh option. The Consperf utility analyzes only fast refresh publication items.

Publication items can be created for complete refresh using the `C` refresh mode in the `createPublicationItem` API from the Consolidator Manager API. When this mode is specified, client data is completely refreshed with current data from the server after every sync. An administrator can force a complete refresh for an entire publication through an API call. This function forces complete refresh of a publication for a given client.

See the Javadoc in the *Oracle Database Lite API Specification* for more information.

The following lists what can cause a complete refresh, ordered from most likely to least likely:

1. The same Mobile user synching from multiple devices on the same platform, or synching from different platforms when the publications are not platform specific.
2. Republishing the application.
3. An unexpected server apply condition, such as constraint violations, unresolved conflicts, and other database exceptions.
4. Modifying the application, such as changing subsetting parameters or adding/altering publication items. This refresh only affects the publication items.
5. A force refresh requested by server administrator or a force refresh requested by the client.
6. Restoring an old Oracle Lite database (ODB file).
7. Two separate applications using the same backend store.
8. An unexpected client apply conditions, such as a moved or deleted database, database corruption, memory corruption, other general system failures.
9. Loss of transaction integrity between the server and client. The server fails post processing after completing the download and disconnects from the client.
10. Data transport corruptions.

3.9.3 Queue-Based Refresh

You can create your own queue. Mobile Server uploads and downloads changes from the user. Perform customized apply/compose modifications to the back-end database with your own implementation. See the Section 3.19, "Customizing Replication With Your Own Queues" for more information.

3.9.4 Forced Refresh

This is actually NOT a refresh option; however, we discuss it here in order to inform you of the consequences of performing a forced refresh. Out of all the different synchronization options, the Forced Refresh synchronization architecture is probably the most misunderstood synchronization type. This option is commonly confused with the Complete Refresh synchronization. This confusion may result in tragic consequences and the loss of critical data on the client.

The Forced Refresh option is an emergency only synchronization option. This option is for when a client is so corrupt or malfunctioning so severely that the determination is made to replace the Mobile client data with a fresh copy of data from the enterprise data store. When this option is selected, any data transactions that have been made on the client are lost.

When a Forced Refresh is initiated all data on the client is removed. The client will then bring 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.

3.10 Resuming an Interrupted Synchronization

With client/server networking, communication may be interrupted by unreliable network conditions, physical disconnections, limited transport bandwidth, and so on. To efficiently cope with these conditions, the transport protocol between the client and server resumes a synchronization session from the last acknowledged byte. For example, the client starts to upload 10 MB of data and the connection fails after sending 9MB of the data. In this instance, the client does not resend the 9MB that was acknowledged, but resumes the synchronization by uploading the last 1 MB of data. The resume feature works the same for both the upload and download phases of the transport.

Configure the resume feature parameters, as follows:

- Section 3.10.1, "Defining Temporary Storage Location for Client Data"
- Section 3.10.2, "Controlling Server Load"
- Section 3.10.3, "Client Configuration."

3.10.1 Defining Temporary Storage Location for Client Data

By default, the client data is buffered in memory and maximum of 16MB is allocated for the buffering. If more space is needed, new clients are blocked until space is freed. Alternatively, you can configure where the client data is temporarily stored and how much space to allocate with the `RESUME_FILE` and `RESUME_FILE_SIZE` parameters in the `CONSOLIDATOR` section of the `webtogo.ora` file on the Mobile Server, as follows:

```
RESUME_FILE=d:\path\file
RESUME_FILE_SIZE =NNN (MB)
```

Setting the `RESUME_FILE_SIZE` parameter configures the amount of memory allocated for the buffering. Setting `RESUME_FILE` allows using a disk file instead of RAM, which is more efficient if JDK1.4 or later is installed and memory mapping can be used.

If there are multiple disks available on the Mobile Server host, one spool file should be created per disk to optimize performance. You can specify several spool files with multiple `RESUME_FILE` and `RESUME_FILE_SIZE` parameters, each designated with a unique suffix, as follow: `RESUME_FILE_2`, `RESUME_FILE_SIZE_2`.

Normally, 64KB blocks are used to buffer client data. Resume block size can be specified in KB, with the `RESUME_BLOCKSIZE` parameter. If you are using disk files to minimize fragmentation, then the block size should be specified as a larger number.

3.10.2 Controlling Server Load

If too many clients connect to a Mobile Server at once, it can become overloaded, run out of memory, or have poor performance when responding to the clients. The `RESUME_MAXACTIVE` parameter controls the maximum number of connections that the Mobile Server handles at a single time. If more clients try to connect, they are queued until existing connections complete. The default is 100 connections.

The `RESUME_TIMEOUT` parameter indicates how long to keep client data while the client is not connected. The default is 0, which means that resume is disabled and after disconnection, the client data is discarded. A short timeout, such as 15 minutes, is suitable to resume any accidentally dropped connections. A longer timeout may be needed if users explicitly pause and resume synchronization to switch networks or use a dialup connection for another purpose.

The `RESUME_MAXCHUNK` parameter causes the server to drop the connection after sending the specified data size, in KB. This forces the client to reconnect and inform the server on how much data it already has. The server can the discard all data before that offset. The fault value is 1024 KB.

These parameters are all configured within the `webtogo.ora` file on the Mobile Server.

3.10.3 Client Configuration.

Configure the client-side parameters for timeout and maximum data size in the `CONSOLIDATOR` section of the `polite.ini` file on the client, as follows:

- The `RESUME_CLIENT_TIMEOUT` parameter is the number of seconds that the client should use to timeout network operations. The default is 60 seconds.
- The `RESUME_CLIENT_MAXSEND` parameter is the maximum data size, in KB, that the client should send in a single POST request. This is used in cases where there is a proxy with a small limit on the data size in one request. Specifying a reasonable value, such as 256 KB, can also help clients with limited storage space, as they can free the chunks that have already been transmitted and acknowledged. The default is 1024 KB.

3.11 Synchronizing With Database Constraints

When you have database constraints on your table, you must develop your application in a certain way to facilitate the synchronization of the data and keeping the database constraints. The following sections detail each constraint and what issues you must take into account:

- Section 3.11.1, "Synchronization And Database Constraints"
- Section 3.11.2, "Primary Key is Unique"
- Section 3.11.3, "Foreign Key Constraints"
- Section 3.11.4, "Unique Key Constraint"
- Section 3.11.5, "Not Null Constraint"
- Section 3.11.6, "Generating Constraints on the Mobile Client"

3.11.1 Synchronization And Database Constraints

Oracle Database Lite does not keep a record of the SQL operations executed against the database; instead, only the final changes are saved and synchronized to the back-end database.

For example, if you have a client with a unique key constraint, where the following is executed against the client Oracle Lite database:

1. Record with primary key of one and unique field of ABC is deleted.
2. Record with primary key of 4 and unique field of ABC is inserted.

When this is synchronized, according the Section 3.4.1.7.2, "Using Table Weight" discussion, the insert is performed before the delete. This would add a duplicate field for ABC and cause a unique key constraint violation. In order to avoid this, you should defer all constraint checking until after all transactions are applied. See Section 3.11.3.2, "Defer Constraint Checking Until After All Transactions Are Applied".

Another example of how synchronization captures the end result of all SQL modifications is as follows:

1. Insert an employee record 4 with name of Joe Judson.
2. Update employee record 4 with address.
3. Update employee record 4 with salary.
4. Update employee record 4 with office number
5. Update employee record 4 with work email address.

When synchronization occurs, all modifications are captured and only a single insert is performed on the back-end database. The insert contains the primary key, name, address, salary, office number and email address. Even though the data was created with multiple updates, the Synch Server only takes the final result and makes a single insert.

Note: If you want these constraints to apply on the Mobile client, see Section 3.11.6, "Generating Constraints on the Mobile Client".

3.11.2 Primary Key is Unique

When you have multiple clients, each updating the same table, you must have a method for guaranteeing that the primary key is unique across all clients. Oracle Database Lite provides you a sequence number that you can use as the primary key, which is guaranteed to be unique across all Oracle Database Lite clients.

For more information on the sequence number, see Section 3.4.1.8, "Creating Client-Side Sequences for the Downloaded Snapshot".

3.11.3 Foreign Key Constraints

A foreign key exists in a details table and points to a row in the master table. Thus, before a client adds a record to the details table, the master table must first exist.

For example, two tables `EMP` and `DEPT` have referential integrity constraints and are an example of a master-detail relationship. The `DEPT` table is the master table; the `EMP` table is the details table. The `DeptNo` field (department number) in the `EMP` table is a foreign key that points to the `DeptNo` field in the `DEPT` table. The `DeptNo` value for each employee in the `EMP` table must be a valid `DeptNo` value in the `DEPT` table.

When a user adds a new employee, first the employee's department must exist in the DEPT table. If it does not exist, then the user first adds the department in the DEPT table, and then adds a new employee to this department in the EMP table. The transaction first updates DEPT and then updates the EMP table. However, Oracle Database Lite does not store the sequence in which these operations were executed.

Oracle Database Lite does not keep a record of the SQL operations executed against the database; instead, only the final changes are saved and synchronized to the back-end database. For our employee example, when the user replicates with the Mobile Server, the Mobile Server could initiate the updates the EMP table first. If this occurs, then it attempts to create a new record in EMP with an invalid foreign key value for DeptNo. Oracle database detects a referential integrity violation. The Mobile Server rolls back the transaction and places the transaction data in the Mobile Server error queue. In this case, the foreign key constraint violation occurred because the operations within the transaction are performed out of their original sequence.

In order to avoid this violation, you can do one of two things:

- Section 3.11.3.1, "Set Update Order for Tables With Weights"
- Section 3.11.3.2, "Defer Constraint Checking Until After All Transactions Are Applied"

3.11.3.1 Set Update Order for Tables With Weights

Set the order in which tables are updated on the back-end Oracle database with weights. To avoid integrity constraints with a master-details relationship, the master table must always be updated first in order to guarantee that it exists before any records are added to a details table. In our example, you must set the DEPT table with a lower weight than the EMP table to ensure that all records are added to the DEPT table first.

You define the order weight for tables when you add a publication item to the publication. For more information on weights, see Section 3.4.1.7.2, "Using Table Weight".

3.11.3.2 Defer Constraint Checking Until After All Transactions Are Applied

You can use a PL/SQL procedure avoid foreign key constraint violations based on out-of-sequence operations by using DEFERRABLE constraints in conjunction with the BeforeApply and AfterApply functions. DEFERRABLE constraints can be either INITIALLY IMMEDIATE or INITIALLY DEFERRED. The behavior of DEFERRABLE INITIALLY IMMEDIATE foreign key constraints is identical to regular immediate constraints. They can be applied interchangeably to applications without impacting functionality.

The Mobile Server calls the BeforeApply function before it applies client transactions to the server and calls the AfterApply function after it applies the transactions. Using the BeforeApply function, you can set constraints to DEFFERED to delay referential integrity checks. After the transaction is applied, call the AfterApply function to set constraints to IMMEDIATE. At this point, if a client transaction violates referential integrity, it is rolled back and moved into the error queues.

To prevent foreign key constraint violations using DEFERRABLE constraints:

1. Drop all foreign key constraints and then recreate them as DEFERRABLE constraints.
2. Bind user-defined PL/SQL procedures to publications that contain tables with referential integrity constraints.

3. The PL/SQL procedure should set constraints to `DEFERRED` in the `BeforeApply` function and `IMMEDIATE` in the `AfterApply` function as in the following example featuring a table named `SAMPLE3` and a constraint named `address.14_fk`:

```
procedure BeforeApply(clientname varchar2) is
cur integer;
begin
  cur := dbms_sql.open_cursor;
  dbms_sql.parse(cur,'SET CONSTRAINT SAMPLE3.address14_fk
                 DEFERRED', dbms_sql.native);
  dbms_sql.close_cursor(cur);
end;
procedure AfterApply(clientname varchar2) is
cur integer;
begin
  cur := dbms_sql.open_cursor;
  dbms_sql.parse(cur, 'SET CONSTRAINT SAMPLE3.address14_fk
                 IMMEDIATE', dbms_sql.native);
  dbms_sql.close_cursor(cur);
end;
```

3.11.4 Unique Key Constraint

A unique key constraint enforces uniqueness of data. However, you may have multiple clients across multiple devices updating the same table. Thus, a record may be unique on a single client, but not across all clients. Enforcing uniqueness is the customer’s reponisibility and depends on the data.

How do you guarantee that the records added on separate clients are unique? You can use the sequence numbers generated on the client by Oracle Database Lite. See Section 3.4.1.8, "Creating Client-Side Sequences for the Downloaded Snapshot" for more information.

3.11.5 Not Null Constraint

When you have a not null constraint on the client or on the server, you must ensure that this constraint is set on both sides.

- On the server—Create a NOT NULL constraint on the back-end server table using the Oracle database commands.
- For the client—Set a column as NOT NULL by executing the `setPubItemColOption` method in the `ConsolidatorManager` API. Provide `Consolidator.NOT_NULL` as the input parameter for `nullType`. The constraint is then enforced on the table in the client Oracle Lite database.

3.11.6 Generating Constraints on the Mobile Client

The Primary Key, Foreign Key, Not Null and Default Value constraints can be synchronized to the Mobile client; the Unique constraints cannot be synchronized. For foreign key constraints, you decide if you want the foreign key on the Mobile client. That is, when you create a foreign key constraint on a table on the back-end server, you may or may not want this constraint to exist on the Mobile client.

- Each publication that is defined is specific to a certain usage. For example, if you have a foreign key constraint between two tables, such as department and employee, your publication may only specify that information from the employee table is downloaded. In this situation, you would not want the foreign constraint between the employee and department table to be enforced on the client.

- If you do have a master-detail relationship or other constraint relationships synchronized down to the client, then you would want to have the constraint generated on the client.

In order to generate the constraints on theMobile client, perform the following:

1. Within the process for creating or modifying an existing publication using the APIs, invoke the `assignWeights` method of the ConsolidatorManager object, which does the following tasks:
   a. Calculates a weight for each of the publication items included in the publication.
   b. Creates a script that, when invoked on the client, generates the constraints on the client. This script is automatically added to the publication.
2. On the Mobile client, perform a synchronization for the user, which brings down the snapshot and the constraint script. The script is automatically executed on the Mobile client.

Once executed on the client, all constraints on the server for this publication are also enforced on the Mobile client.

3.11.6.1 The assignWeights Method

The `assignWeights` method automatically calculates weights for all publication items belonging to a publication. If a new publication item is added or if there is a change in the referential relationships, the API should be called again.

The following defines the `assignWeights` method and its parameters:

```
public void assignWeights(java.lang.String pub, boolean createScripts)
                  throws ConsolidatorException
```

Where:

- `pub` - Publication name
- `createScripts` - If true, creates refrential constraints scripts and adds them to the publication to be propagated to subscribed clients.

3.12 Parent Tables Needed for Updateable Views

For a view to be updatable, it must have a parent table. A parent table can be any one of the view base tables in which a primary key is included in the view column list and is unique in the view row set. If you want to make a view updatable, provide the Mobile Server with the appropriate hint and the view parent table before you create a publication item on the view.

To make publication items based on a updatable view, use the following two mechanisms:

- Parent table hints
- `INSTEAD OF` triggers or DML procedure callouts

3.12.1 Creating a Parent Hint

Parent table hints define the parent table for a given view. Parent table hints are provided through the `parentHint` method of the Consolidator Manager object, as follows:

```
consolidatorManager.parentHint("SAMPLE3","ADDROLRL4P","SAMPLE3","ADDRESS");
```

See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.12.2 INSTEAD OF Triggers

`INSTEAD OF` triggers are used to execute `INSTEAD OF INSERT`, `INSTEAD OF UPDATE`, or `INSTEAD OF DELETE` commands. `INSTEAD OF` triggers also map these DML commands into operations that are performed against the view base tables. `INSTEAD OF` triggers are a function of the Oracle database. See the Oracle database documentation for details on `INSTEAD OF` triggers.

3.13 Resolving Conflict Resolution with Winning Rules

- Client wins—When the client wins, the Mobile Server automatically applies client changes to the server. And if you have a record that is set for `INSERT`, yet a record already exists, the Mobile Server automatically modifies it to be an `UPDATE`.
- Server wins—If the server wins, the client updates are not applied to the application tables. Instead, the Mobile Server automatically composes changes for the client. The client updates are placed into the error queue, just in case you still want these changes to be applied to the server—even though the winning rules state that the server wins.

The Mobile Server uses internal versioning to detect synchronization conflicts. A separate version number is maintained for each client and server record. When the client updates are applied to the server, then the Mobile Server checks the version numbers of the client against the version numbers on the server. If the version does not match, then the conflict resolves according to the defined winning rules—such as client wins or server wins, as follows:

The Mobile Server does not automatically resolve synchronization errors. Instead, the Mobile Server rolls back the corresponding transactions, and moves the transaction operations into the Mobile Server error queue. It is up to the administrator to view the error queue and determine if the correct action occurred. If not, the administrator must correct and re-execute the transaction. If it did execute correctly, then purge the transaction from the error queue.

One type of error is a synchronization conflict, which is detected in any of the following situations:

- The client and the server update the same row.
- The client deletes the same row that the server updates.
- The client updates a row at the same time that the server deletes it when the "server wins" conflict rule is specified. This is considered a synchronization error for compatibility with Oracle database advanced replication.
- Both the client and server create rows with the same primary key values.
- For systems with delayed data processing, where the client data is not directly applied to the base table—for instance, in a three-tiered architecture—a situation could occur when a client inserts a row and then updates the same row, while the row has not yet been inserted into the base table. In that case, if the `DEF_APPLY`

parameter in C$ALL_CONFIG is set to TRUE, an INSERT operation is performed, instead of the UPDATE. It is up to the application developer to resolve the resulting primary key conflict. If, however, DEF_APPLY is not set, a "NO DATA FOUND" exception is thrown.

All the other errors, including nullity violations and foreign key constraint violations are synchronization errors. See Section 3.11, "Synchronizing With Database Constraints" for more information.

All synchronization errors are placed into the error queue. For each publication item created, a separate and corresponding error queue is created. The purpose of this queue is to store transactions that fail due to unresolved conflicts. The administrator can attempt to resolve the conflicts, either by modifying the error queue data or that of the server, and then attempt to re-apply the transaction.

The administrator can change the transaction operations and re-execute or purge transactions from the error queue from either of the following:

- The Mobile Manager GUI—See Chapter 6, "Managing Synchronization" in the *Oracle Database Lite Administration and Deployment Guide* on how to update the client transaction in the error queue and re-execute the statement using the Mobile Manager GUI.
- The Consolidator Manager API and accessing the Mobile Server error queue tables directly and customize the conflict rules, as described in the following sections:
  - Section 3.13.1, "Resolving Errors and Conflicts Using the Error Queue"
  - Section 3.13.2, "Customizing Synchronization Conflict Resolution Outcomes"

3.13.1 Resolving Errors and Conflicts Using the Error Queue

The error queue stores transactions that fail due to synchronization errors or unresolved conflicts. For unresolved conflicts, only the "Server Wins" conflicts are reported. If you have set your conflict rules to "Client Wins", then these are not reported. The administrator can do one of the following:

- Attempt to correct the error by modifying the error queue data or that of the server, and re-apply the transaction through the executeTransaction method of the Consolidator Manager object.
- If a conflict was reported and resolved to your satisfaction, then you can purge the transaction from the error queue with the purgeTransaction method of the Consolidator Manager object. Otherwise, you can override the default conflict resolution by modifying the error queue data and re-apply the transaction.

View the error queue through the Mobile Manager GUI, where you can see what the conflict was. You can fix the problem and reapply the data by modifying the DML operation appropriately and then re-executing. See Section 5.9.4.3 "Viewing Transactions in the Error Queue" in the *Oracle Database Lite Administration and Deployment Guide* for directions.

3.13.2 Customizing Synchronization Conflict Resolution Outcomes

You can customize synchronization conflict resolution by performing the following:

1. Configure the winning rule to Client Wins.
2. Attach BEFORE INSERT, UPDATE, and DELETE triggers to database tables.
3. Create a custom DML procedure.

The triggers in the database compare old and new row values and resolve client changes, as you specify in the triggers.

3.14 Manipulating Application Tables

If you need to manipulate the application tables to create a secondary index or a virtual primary key, you can use `ConsolidatorManager` methods to programmatically perform these tasks in your application, as described in the following sections:

- Section 3.14.1, "Creating Secondary Indexes on Client Device"
- Section 3.14.2, "Virtual Primary Key"

3.14.1 Creating Secondary Indexes on Client Device

The first time a client synchronizes, the Mobile Server automatically enables a Mobile client to create the database objects on the client in the form of snapshots. By default, the primary key index of a table is automatically replicated from the server. You can create secondary indexes on a publication item through the Consolidator Manager APIs. See the *Oracle Database Lite API Javadoc* for specific API information. See Section 3.4.1.6, "Create Publication Item Indexes" for an example.

3.14.2 Virtual Primary Key

You can specify a virtual primary key for publication items where the base object does not have a primary key defined. This is useful if you want to create a fast refresh publication item on a table that does not have a primary key.

A virtual primary key must be unique and not null. You can create a virtual primary key for more than one column, but the API must be called separately for each column that you wish to assign a virtual primary key. The following methods create and drop a virtual primary key.

Use the `createVirtualPKColumn` method to create a virtual primary key column.

```
consolidatorManager.createVirtualPKColumn("SAMPLE1", "DEPT", "DEPT_ID");
```

Use the `dropVirtualPKColumns` method to drop a virtual primary key.

```
consolidatorManager.dropVirtualPKColumns("SAMPLE1", "DEPT");
```

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.15 Facilitating Schema Evolution

You can use schema evolution when adding or altering a column in the application tables for updatable publication items. You do not use schema evolution for read-only publication items.

If you do alter the schema, then the client receives a complete refresh on the modified publication item, but not for the entire publication.

Note: You should stop all synchronization events and MGP activity during a schema evolution.

The following types of schema modifications is supported:

- Add new columns
- Change the type of a column—You can only modify the type of a column in accordance to the Oracle Database limitations. In addition, you **CANNOT** modify a primary key or virtual primary key column
- Increase the width of a column

Note: You cannot modify the definition of any primary key or virtual primary key.

For facilitating schema evolution, perform the following:

1. If necessary, modify the table in the back-end Oracle database.
2. Modify the publication item directly on the production Mobile repository through MDW or the `alterPublicationItem` API. Modifying the SQL query of the publication item causes the schema evolution to occur.

   A schema evolution only occurs if the SQL query is modified. If the SQL query does not change, then the evolution does not occur. If your modification only touched the table, then you must modify the SQL query by adding an additional space to force the schema evolution to occur.

   Note: If you decide to republish the application to a different Mobile repository, then update the publication definition in the packaging wizard.

   ---

3. Once you alter the SQL query, then either use Mobile Manager to refresh the metadata cache or restart the Mobile Server. To refresh the metadata cache through the Mobile Server, select Data Synchronization->Administration->Reset Metadata Cache or execute the `resetCache` method of the `ConsolidatorManager` class.

   Note: Use of the high priority flag during sync will override any schema evolution, as a result, the new table definition will not come to the client.

When you modify the table in the Mobile repository, the client snapshot is no longer. Thus—by default—a complete refresh occurs the next time you synchronize, because a new snapshot must be created on the client.

3.16 Set DBA or Operational Privileges for the Mobile Server

You can set either DBA or operational privileges for the Mobile Server with the following Consolidator Manager API:

```
void setMobilePrivileges( String dba_schema, String dba_pass, int type )
       throws ConsolidatorException
```

where the input parameter are as follows:

- `dba_schema`—The DBA schema name
- `dba_pass`—The DBA password

- `type`—Define the user by setting this parameter to either `Consolidator.DBA` or `Consolidator.OPER`

If you specify Consolidator.DBA, then the privileges needed are those necessary for granting DBA privileges that are required for publish/subscribe functions of the Mobile Server.

If you specify `Consolidator.OPER` type, then the privileges needed are those necessary for executing the Mobile Server without any schema modifications. The OPER is given DML and select access to publication item base objects, version, log, and error queue tables.

The Mobile Server privileges are modified using the `C$MOBILE_PRIVILEGES` PL/SQL package, which is created for you automatically after the first time you use the `setMobilePrivileges` procedure. After the package is created, the Mobile Server privileges can be administered from SQL or from this Java API.

3.17 Create a Synonym for Remote Database Link Support For a Publication Item

Publication items can be defined for database objects existing on remote database instances outside of the Mobile Server repository. Local private synonyms of the remote objects should be created in the Oracle database. Execute the following SQL script located in the <*ORACLE_HOME*>`\Mobile\server\admin\consolidator_rmt.sql` directory on the remote schema in order to create Consolidator Manager logging objects.

The synonyms should then be published using the `createPublicationItem` method of the `ConsolidatorManager` object. If the remote object is a view that needs to be published in updatable mode and/or fast-refresh mode, the remote parent table must also be published locally. Parent hints should be provided for the synonym of the remote view similar those used for local, updatable and/or fast refreshable views.

Two additional methods have been created, `dependencyHint` and `removeDependencyHint`, to deal with non-apparent dependencies introduced by publication of remote objects.

Remote links to the Oracle database must be established prior to attempting remote linking procedures, please refer to the *Oracle SQL Reference* for this information.

Note: The performance of synchronization from remote databases is subject to network throughput and the performance of remote query processing. Because of this, remote data synchronization is best used for simple views or tables with limited amount of data.

The following sections describe how to manage remote links:

- Section 3.17.1, "Publishing Synonyms for the Remote Object Using CreatePublicationItem"
- Section 3.17.2, "Creating or Removing a Dependency Hint"

3.17.1 Publishing Synonyms for the Remote Object Using CreatePublicationItem

The `createPublicationItem` method creates a new, stand-alone publication item as a remote database object. If the URL string is used, the remote connection is established and closed automatically. If the connection is null or cannot be established, an exception is thrown. The remote connection information is used to create logging objects on the linked database and to extract metadata.

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

```
consolidatorManager.createPublicationItem(
     "jdbc:oracle:oci8:@oracle.world",
     "P_SAMPLE1",
     "SAMPLE1",
     "PAYROLL_SYN",
     "F"
     "SELECT * FROM sample1.PAYROLL_SYN"+"WHERE SALARY >:CAP", null, null);
```

Note: Within the select statement, the parameter name for the data subset must be prefixed with a colon, for example `:CAP`.

3.17.2 Creating or Removing a Dependency Hint

Use the `dependencyHint` method to create a hint for a non-apparent dependency.

```
Given remote view definition
        create payroll_view as
        select p.pid, e.name
        from payroll p, emp e
        where p.emp_id = e.emp_id;

Execute locally
        create synonym v_payroll_syn for payroll_view@<remote_link_address>;
        create synonym t_emp_syn for emp@<remote_link_address>;
```

Where `<remote_link_address>` is the link established on the Oracle database. Use `dependencyHint` to indicate that the local synonym `v_payroll_syn` depends on the local synonym `t_emp_syn`:

```
consolidatorManager.dependencyHint("SAMPLE1","V_PAYROLL_SYN","SAMPLE1","T_EMP_
SYN");
```

Use the `removeDependencyHint` method to remove a hint for a non-apparent dependency.

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.18 Using the Sync Discovery API to Retrieve Statistics

The sync discovery feature is used to request an estimate of the size of the download for a specific client, based on historical data. The following statistics are gathered to maintain the historical data:

- The total number of rows send for each publication item.
- The total data size for these rows.
- The compressed data size for these rows.

The following sections contain methods that can be used to gather statistics:

- Section 3.18.1, "getDownloadInfo Method"
- Section 3.18.2, "DownloadInfo Class Access Methods"
- Section 3.18.3, "PublicationSize Class"

3.18.1 getDownloadInfo Method

The `getDownloadInfo` method returns the `DownloadInfo` object. The `DownloadInfo` object contains a set of `PublicationSize` objects and access methods. The `PublicationSize` objects carry the size information of a publication item. The method `Iterator iterator()` can then be used to view each `PublicationSize` object in the `DownloadInfo` object.

```
DownloadInfo dl = consolidatorManager.getDownloadInfo("S11U1", true, true);
```

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.18.2 DownloadInfo Class Access Methods

The access methods provided by the `DownloadInfo` class are listed in Table 3–23:

Table 3–23 DownloadInfo Class Access Methods

Method	Definition
`iterator`	Returns an Iterator object so that the user can traverse through the all the `PublicationSize` objects that are contained inside the `DownloadInfo` object.
`getTotalSize`	Returns the size information of all `PublicationSize` objects in bytes, and by extension, the size of all publication items subscribed to by that user. If no historical information is available for those publication items, the value returned is '-1'.
`getPubSize`	Returns the size of all publication items that belong to the publication referred to by the string `pubName`. If no historical information is available for those publication items, the value returned is '-1'.
`getPubRecCount`	Returns the number of all records of all the publication items that belong to the publication referred by the string `pubName`, that will be synchronization during the next synchronization.
`getPubItemSize`	Returns the size of a particular publication item referred by `pubItemName`. It follows the following rules in order. **1.** If the publication item is empty, it will return '0'. **2.** If no historical information is available for those publication items, it will return '-1'.
`getPubItemRecCount`	Returns the number of records of the publication item referred by `pubItemName` that will be synced in the next synchronization.

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.18.3 PublicationSize Class

The access methods provided by the `PublicationSize` class are listed inTable 3–24:

Table 3–24 PublicationSize Class Access Methods

Parameter	Definition
`getPubName`	Returns the name of the publication containing the publication item.
`getPubItemName`	Returns the name of the publication item referred to by the `PublicationSize` object.
`getSize`	Returns the total size of the publication item referred to by the `PublicationSize` object.
`getNumOfRows`	Returns the number of rows of the publication item that will be synchronized in the next synchronization.

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

Sample Code

```
import   java.sql.*;
import   java.util.Iterator;
import   java.util.HashSet;

import   oracle.lite.sync.ConsolidatorManager;
import   oracle.lite.sync.DownloadInfo;
import   oracle.lite.sync.PublicationSize;

public class TestGetDownloadInfo
{

   public static void main(String argv[]) throws Throwable
   {
// Open Consolidator Manager connection
      try
      {
// Create a ConsolidatorManager object
         ConsolidatorManager cm = new ConsolidatorManager ();
// Open a Consolidator Manager connection
         cm.openConnection ("MOBILEADMIN", "MANAGER",
                      "jdbc:oracle:thin:@server:1521:orcl", System.out);
// Call getDownloadInfo
         DownloadInfo dlInfo = cm.getDownloadInfo ("S11U1", true, true);
// Call iterator for the Iterator object and then we can use that to transverse
// through the set of PublicationSize objects.
         Iterator it = dlInfo.iterator ();
// A temporary holder for the PublicationSize object.
         PublicationSize ps = null;
// A temporary holder for the name of all the Publications in a HashSet object.
         HashSet pubNames = new HashSet ();
// A temporary holder for the name of all the Publication Items in a HashSet
// object.
         HashSet pubItemNames = new HashSet ();
// Traverse through the set.
         while (it.hasNext ())
         {
```

```
// Obtain the next PublicationSize object by calling next ().
            ps = (PublicationSize)it.next ();

// Obtain the name of the Publication this PublicationSize object is associated
// with by calling getPubName ().
            pubName = ps.getPubName ();
            System.out.println ("Publication: " + pubName);

// We save pubName for later use.
            pubNames.add (pubName);

// Obtain the Publication name of it by calling getPubName ().
            pubItemName = ps.getPubItemName ();
            System.out.println ("Publication Item Name: " + pubItemName);

// We save pubItemName for later use.
            pubItemNames.add (pubItemName);

// Obtain the size of it by calling getSize ().
            size = ps.getSize ();
            System.out.println ("Size of the Publication: " + size);

// Obtain the number of rows by calling getNumOfRows ().
            numOfRows = ps.getNumOfRows ();
            System.out.println ("Number of rows in the Publication: "
                                + numOfRows);
         }

// Obtain the size of all the Publications contained in the
// DownloadInfo objects.
         long totalSize = dlInfo.getTotalSize ();
         System.out.println ("Total size of all Publications: " + totalSize);

// A temporary holder for the Publication size.
         long pubSize = 0;

// A temporary holder for the Publication number of rows.
         long pubRecCount = 0;

// A temporary holder for the name of the Publication.
         String tmpPubName = null;

// Transverse through the Publication names that we saved earlier.
         it = pubNames.iterator ();
         while (it.hasNext ())
         {
// Obtain the saved name.
            tmpPubName = (String) it.next ();

// Obtain the size of the Publication.
            pubSize = dlInfo.getPubSize (tmpPubName);
            System.out.println ("Size of " + tmpPubName + ": " + pubSize);

// Obtain the number of rows of the Publication.
            pubRecCount = dlInfo.getPubRecCount (tmpPubName);
            System.out.println ("Number of rows in " + tmpPubName + ": "
                                + pubRecCount);
         }

// A temporary holder for the Publication Item size.
```

```
         long pubItemSize = 0;

// A temporary holder for the Publication Item number of rows.
         long pubItemRecCount = 0;

// A temporary holder for the name of the Publication Item.
         String tmpPubItemName = null;

// Traverse through the Publication Item names that we saved earlier.
         it = pubItemNames.iterator ();
         while (it.hasNext ())
         {
// Obtain the saved name.
            tmpPubItemName = (String) it.next ();

// Obtain the size of the Publication Item.
            pubItemSize = dlInfo.getPubItemSize (tmpPubItemName);
            System.out.println ("Size of " + pubItemSize + ": " + pubItemSize);

// Obtain the number of rows of the Publication Item.
            pubItemRecCount = dlInfo.getPubItemRecCount (tmpPubItemName);
            System.out.println ("Number of rows in " + tmpPubItemName + ": "
                                   + pubItemRecCount);
         }
         System.out.println ();

// Close the connection
         cm.closeConnection ();
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

3.19 Customizing Replication With Your Own Queues

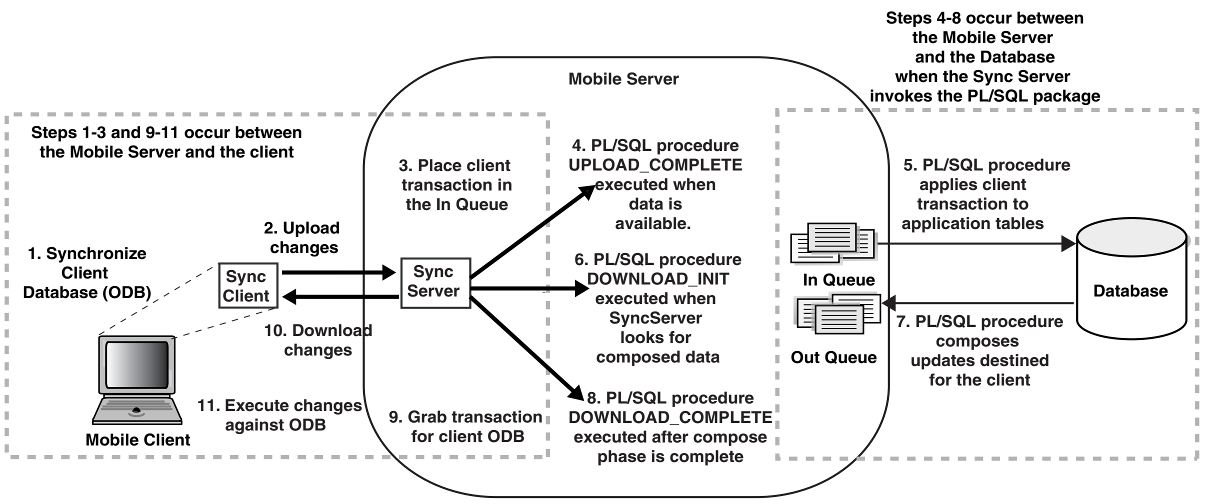
Application developers can manage the replication process programmatically by using queue-based publication items. By default on the server-side, the MGP manages both the in queues and the out queues by gathering all updates posted to the in queue, applying these updates to the relevant tables, and then composing all new updates created on the server that are destined for the client and posting it to the out queue. This is described in Section 3.1, "How Does Synchronization Work?".

However, you can bypass the MGP and provide your own solution for the apply and compose phases on the server-side for selected publication items. You may wish to bypass the MGP for the publication item if one or more of the following are true:

- If you want to facilitate synchronous data exchange, use queue-based publication items.
- If you have complex business rules for data subsetting, in how you decide what data each user receives, then use queue-based publication items. You can incorporate these business rules into generation of the client’s queue data. This is especially true if the rules are dynamically evaluated during runtime.
- If your client collects large amounts of data only for upload to the server, never receives data from the server, and it does not require conflict resolution, then use the data collection queues.

Figure 3–2 shows how the Sync Server invokes the UPLOAD_COMPLETE PL/SQL procedure when the client upload is complete. And before it downloads all composed updates to the client, the Sync Server invokes the DOWNLOAD_INIT PL/SQL procedure.

Figure 3–2 Queue-Based Synchronization Architecture


Steps 4-8 occur between the Mobile Server and the Database when the Sync Server invokes the PL/SQL package
Mobile Server
Steps 1-3 and 9-11 occur between the Mobile Server and the client
3. Place client transaction in the In Queue
4. PL/SQL procedure UPLOAD_COMPLETE executed when data is available.
5. PL/SQL procedure applies client transaction to application tables
2. Upload changes
1. Synchronize Client Database (ODB)
Sync Client
Sync Server
6. PL/SQL procedure DOWNLOAD_INIT executed when SyncServer looks for composed data
In Queue
Database
Out Queue
7. PL/SQL procedure composes updates destined for the client
10. Download changes
11. Execute changes against ODB
9. Grab transaction for client ODB
8. PL/SQL procedure DOWNLOAD_COMPLETE executed after compose phase is complete
Mobile Client

To bypass the MGP, do the following:

1. Define your publication item as queue-based or data collection. Then, the MGP is not aware of the queues associated with this publication item. You can do this when creating the publication item either through MDW or Consolidator APIs.
2. If queue-based, then create a package, either PL/SQL or Java, that implements the queue interface callback methods. This includes the following callback methods:
   - UPLOAD_COMPLETE to process the incoming updates from the client.
   - DOWNLOAD_INIT to complete the compose phase.
   - DOWNLOAD_COMPLETE if you have any processing to perform after the compose phase.
3. Create the queues. The in queue, CFM$*<publication_item_name>* is created by default for you. Create the out queue as CTM$*<publication_item_name>*.

This section describes two methods for customizing the server-side apply/compose phases, as follows:

- Section 3.19.1, "Customizing Apply/Compose Phase of Synchronization with a Queue-Based Publication Item"—You can define both the apply and compose phases using queue-based publication items.
- Section 3.19.2, "Creating Data Collection Queues for Uploading Client Collected Data"—You can only use the data collection queue for the upload phase. The queues are optimized for when a client collects data to upload to the server and never receives data from the server.
- Section 3.19.3, "Selecting How/When to Notify Clients of Composed Data"—You can notify a client that there is new data on the server ready to be downloaded to initiate a synchronization.

3.19.1 Customizing Apply/Compose Phase of Synchronization with a Queue-Based Publication Item

---

Note: The sample for queue-based publication items is located in *<OLITE_HOME>*/Mobile/Sdk/samples/Sync/win32/QBasedPI.

When you want to substitute your own logic for the apply/compose phase of the synchronization process, use a queue-based publication item. The following briefly gives an overview of how the process works internally when using a queue-based publication item:

- When data arrives from the client it is placed in the publication item in queues. The Sync Server calls UPLOAD_COMPLETE, after which the data is committed. All records in the current synchronization session are given the same transaction identifier. The Queue Control Table (C$INQ) indicates which publication item In Queues have received new transactions with the unique transaction identifier. Thus, this table shows which queues need processing.
- If you have a queue-based publication item, you must implement the compose phase, if you have one. The MGP is unaware of queue-based publication items and so will not be able to perform any action for this publication item. When you implement your own compose logic, you decide when and how the compose logic is invoked. For example, you could do the following:
  - You could have a script execute your compose logic at a certain time of the day.
  - You could schedule the compose procedure as a job in the Job Scheduler.
  - You could include the compose logic as part of the DOWNLOAD_INIT function, so that it executes before the client downloads.

  Note: If you decide to implement the compose phase independent of the DOWNLOAD_INIT function; then once the compose is finished, you may want the client to receive the data as soon as possible. In this case, invoke the EN_QUEUE_NOTIFICATION function to start an automatic synchronization from the client. For more information on this function, see Section 3.19.3, "Selecting How/When to Notify Clients of Composed Data".

  Before the Sync Server begins the download phase of the synchronization session, it calls DOWNLOAD_INIT. In this procedure, you can customize the compose or develop any pre-download logic for the client. The Sync Server finds a list of the publication items, which can be downloaded based on the client's subscription. A list of publication items and their refresh mode, ('Y' for complete refresh, 'N' for fast refresh) is inserted into a temporary table (C$PUB_LIST_Q). Items can be deleted or the refresh status can be modified in this table since the Sync Server refers to C$PUB_LIST_Q to determine the items that are downloaded to the client.

Similar to the In Queue, every record in the Out Queue should be associated with it a transaction identifier (TRANID$$). The Sync Server passes the last_tran parameter to indicate the last transaction that the client has successfully applied. New out queue records that have not been downloaded to the client are be marked with the value of curr_tran parameter. The value of curr_tran is always greater than that of last_tran, though not sequential. The Sync Server downloads records from the Out

Queues when the value of TRANID$$ is greater than last_tran. When the data is downloaded, the Sync Server calls DOWNLOAD_COMPLETE.

When you decide to use queue-based publication items, you need to do the following:

1. Create both the In and Out Queues used in the apply and compose phases.
   - You can use the default In Queue, which is named CFM$*<publication_item_name>*. Alternatively, you can create the queue of this name manually. For example, if you wanted the In Queue to be a view, then you would create the In Queue manually.
   - Create the Out Queue for the compose phase as CTM$*<publication_item_name>*.
2. Create the publication item and define it as a queue-based publication item. This can be done either through MDW or the Consolidator APIs.
3. Create the PL/SQL or Java callback methods for performing the apply and compose phases. Since the MGP has nothing to do with the queues used for these phases, when you are finished processing the data, you must manage the queues by deleting any rows that have completed the necessary processing.
4. Register the package to be used for all of the queue processing for a particular publication item.

3.19.1.1 Queue Creation

If a queue-based publication item is created, it will always use a queue by the name of CFM$*<publication_item_name>*. However, if you want to customize how the In Queue is defined—for example, by defining certain rules, making it a view or designating the location of the queue—then you can create your own In Queue. The Out Queue is never defined for you, so you must create an Out Queue named CTM$*<publication_item_name>* in the Mobile Server repository manually using SQL.

These queues are created based upon the publication item tables. For example, the following table ACTIVESTATEMENT has five columns, as follows:

```
create table ACTIVESTATEMENT(
        StatementName varchar2(50) primary key,
        TestSuiteName varchar2(50),
        TestCaseName varchar2(50),
        CurrLine varchar2(4000),
        ASOrder integer) nologging;
```

The application stores its data in these five columns. When synchronization occurs, this data must be uploaded and downloaded. However, there is also meta-information necessary for facilitating the synchronization phases. Therefore, the Out Queue that you create contains the meta-information in the CLID$$CS, TRANID$$ and DMLTYPE$$ columns, as well as the columns from the ACTIVESTATEMENT table, as follows:

```
create table CTM$AUTOTS_PUBITEM(
CLID$$CS VARCHAR2 (30),
StatementName varchar2(50) primary key,
TestSuiteName varchar2(50),
TestCaseName varchar2(50),
CurrLine varchar2(4000),
ASOrder integer,
TRANID$$ NUMBER (10),
DMLTYPE$$ CHAR (1) CHECK (DMLTYPE$$ IN ('I','U','D'))) nologging;
```

Thus, before you can create the queues, you must already know the structure of the tables for the publication item, as well as the publication item name.

The following shows the structure and creation of the queues:

- In queue
- Out queue
- Queue Control Table
- Temporary Table

In queue

All In Queues are named `CFM$<name>` where name is the publication item name. It contains the application publication item table columns, as well as the fields listed in Table 3–25:

Table 3–25 In Queue Interface Creation Parameters

Parameter	Description
`CLID$$CS`	A unique string identifying the client.
`TRANID$$`	A unique number identifying the transaction.
`SEQNO$$`	A unique number for every DML language operation per transaction in the inqueue (CFM$) only.
`DMLTYPE$$`	Checks the type of DML instruction: ■ 'I' - Insert ■ 'D' - Delete ■ 'U' - Update

The following designates the structure when creating the In Queue:

```
create table 'CFM$'+name
(
CLID$$CS   VARCHAR2 (30),
TRANID$$   NUMBER (10),
SEQNO$$    NUMBER (10),
DMLTYPE$$  CHAR (1) CHECK (DMLTYPE$$  IN ('I','U','D'),
publication item column definitions
)
```

Note: You must have the parameters in the same order as shown above for the In Queue. It is different than the ordering in the Out Queue.

Out queue

All Out Queues are named `CTM$<name>` where name is the publication item name. It contains the application publication item table columns, as well as the fields listed in Table 3–26:

Table 3–26 Out Queue Interface Creation Parameters

Parameter	Description
`CLID$$CS`	A unique string identifying the client.

Table 3–26 (Cont.) Out Queue Interface Creation Parameters

Parameter	Description
`TRANID$$`	A unique number identifying the transaction.
`DMLTYPE$$`	Checks the type of DML instruction: ▪ 'I' - Insert ▪ 'D' - Delete ▪ 'U' - Update

The following designates the structure when creating the In Queue:

```
create table 'CTM$'+name
(
CLID$$CS   VARCHAR2 (30),
publication item column definitions
TRANID$$   NUMBER (10),
DMLTYPE$$  CHAR (1) CHECK (DMLTYPE$$  IN ('I','U','D'),
)
```

Note: You must have the parameters in the same order as shown above for the Out Queue. It is different than the ordering in the In Queue.

Another example of creating an Out Queue is in the FServ example, which uses the default In Queue of `CFM$PI_FSERV_TASKS` and creates the `CTM$PI_FSERV_TASKS` Out Queue for the `PI_FSERV_TASKS` publication item, as follows:

```
create table CTM$PI_FSERV_TASKS(
                   CLID$$CS     varchar2(30),
                   ID           number,
                   EMP_ID       number,
                   CUST_ID      number,
                   STAT_ID      number,
                   NOTES        varchar2(255)
                   TRANID$$     number(10),
                   DMLTYPE$$    char(1) check(DMLTYPE$$ in ('I','U','D')),
);
```

Note: The application publication item table for the FServ example contains columns for ID, EMP_ID, CUST_ID, STAT_ID, and NOTES.

Queue Control Table

The Sync Server automatically creates a queue control table, `C$INQ`, and a temporary table, `C$PUB_LIST_Q`. You will process the information in the queue control table in the PL/SQL or Java callout methods to determine which publication items have received new transactions.

The parameters for the control table queue are listed in Table 3–27:

Table 3–27 Queue Control Table Parameters

Parameter	Description
`CLID$$CS`	A unique string identifying the client.
`TRANID$$`	A unique number identifying the transaction.

Table 3–27 (Cont.) Queue Control Table Parameters

Parameter	Description
STORE	Represents the publication item name in the queue control table.

The control table has the following structure:

```
'C$INQ'
(
CLIENTID   VARCHAR2 (30),
TRANID$$   NUMBER,
STORE      VARCHAR2 (30),

)
```

Temporary Table

The DOWNLOAD_INIT procedure uses the Temporary Table C$PUB_LIST_Q for determining what publication items to download in the compose phase.

```
'C$PUB_LIST_Q'
(
NAME   VARCHAR2 (30),
COMP_REF   CHAR(1),
CHECK(COMP_REF IN('Y','N'))
)
```

The parameters for the manually created queues are listed in Table 3–28:

Table 3–28 Queue Interface Creation Parameters

Parameter	Description
NAME	The publication item name that is to be downloaded from the repository to the Out Queue.
COMP_REF	This value is 'Y' for complete refresh.

3.19.1.2 Queue-Based PL/SQL Procedure for UPLOAD_COMPLETE and DOWNLOAD_INIT Callouts

The PL/SQL package for the queue-based publication callouts is in a package where both the UPLOAD_COMPLETE and DOWNLOAD_INIT procedures are defined. The signatures for both callout procedures are as follows:

```
CREATE OR REPLACE PACKAGE CONS_QPKG AS
/*
 * notifies that In Queue has a new transaction by providing the client
 * identifier and the transaction identifier.
*/
PROCEDURE UPLOAD_COMPLETE(
     CLIENTID      IN     VARCHAR2,
     TRAN_ID      IN     NUMBER     -- IN queue tranid
     );
/*
 * initializes client data for download. provides the compose phase for the
 * client. The input data for this procedure is the client id, the last
 * and current transaction markers and the priority.
*/
PROCEDURE DOWNLOAD_INIT(
     CLIENTID      IN     VARCHAR2,
     LAST_TRAN     IN     NUMBER,
     CURR_TRAN     IN     NUMBER,
     HIGH_PRTY     IN     VARCHAR2
     );
/*
 *  notifies when all the client's data is sent
*/
PROCEDURE DOWNLOAD_COMPLETE(
     CLIENTID     IN     VARCHAR2
     );

END CONS_QPKG;
/
```

3.19.1.2.1 In Queue Apply Phase Processing Within the `UPLOAD_COMPLETE` procedure, you should develop a method of applying all changes from the client to the correct tables in the repository. The FServ example performs the following:

1. From the Master Table `C$INQ`, locates the rows for the designated client and transaction identifiers that have been marked for update.
2. Retrieves the application publication item data and the `DMLTYPE$$` from the In Queue, based on the client and transaction identifiers.
3. Performs insert, update, or delete (determined by the value in `DMLTYPE$$`) for updates in the application tables in the repository.
4. After updates are complete, delete the rows in the `C$INQ` and the In Queue that you just processed.

```
PROCEDURE UPLOAD_COMPLETE(CLIENTID IN VARCHAR2, TRAN_ID IN NUMBER) IS
/*create cursors for execution later */
/* PI_CUR locates the rows for the client out of the master table */
CURSOR PI_CUR(C_CLIENTID VARCHAR2, C_TRAN_ID NUMBER ) IS
   SELECT STORE FROM C$INQ
       WHERE CLID$$CS = C_CLIENTID AND TRANID$$ = C_TRAN_ID FOR UPDATE;
/* TASKS_CUR retrieves the values for the client data to be updated */
/*   from the In Queue */
CURSOR TASKS_CUR(C_CLIENTID varchar2, C_TRAN_ID number ) IS
   SELECT ID, EMP_ID, STAT_ID, NOTES, DMLTYPE$$ FROM CFM$PI_FSERV_TASKS
       WHERE CLID$$CS = C_CLIENTID AND TRANID$$ = C_TRAN_ID FOR UPDATE;
/* create variables */
TASK_OBJ TASKS_CUR%ROWTYPE;
PI_OBJ PI_CUR%ROWTYPE;
INSERT_NOT_ALLOWED EXCEPTION;
DELETE_NOT_ALLOWED EXCEPTION;
UNKNOWN_DMLTYPE EXCEPTION;

BEGIN

   OPEN PI_CUR(CLIENTID, TRAN_ID);
   /* C$INQ is used to find out which publication items have received data
      from clients. The publication item name is available in the STORE column
    */
   LOOP
     FETCH PI_CUR INTO PI_OBJ;
     EXIT WHEN PI_CUR%NOTFOUND;

   /* Locate the updates for the publication item PI_FSERV_TASKS */
     IF PI_OBJ.STORE = 'PI_FSERV_TASKS' THEN
       OPEN TASKS_CUR(CLIENTID, TRAN_ID);
       LOOP
```

```
         /* Process the in queue for PI_FSERV_TASKS */
         FETCH TASKS_CUR INTO TASK_OBJ;
         EXIT WHEN TASKS_CUR%NOTFOUND;

         /* Discover the DML command requested. For this publication, only
             updates are allowed.
         IF TASK_OBJ.DMLTYPE$$ = 'I' THEN
            RAISE INSERT_NOT_ALLOWED;
         ELSIF TASK_OBJ.DMLTYPE$$ = 'U' THEN
            FSERV_TASKS.UPDATE_TASK(TASK_OBJ.ID, TASK_OBJ.EMP_ID,
                 TASK_OBJ.STAT_ID, TASK_OBJ.NOTES);
         ELSIF TASK_OBJ.DMLTYPE$$ = 'D' THEN
            RAISE DELETE_NOT_ALLOWED;
         ELSE
            RAISE UNKNOWN_DMLTYPE;
         END IF;

         /* after processing, delete the update request from the in queue */
         DELETE FROM CFM$PI_FSERV_TASKS WHERE CURRENT OF TASKS_CUR;
       END LOOP;
       close TASKS_CUR;
     END IF;

     /* after completing all updates for the client apply phase, delete from
        master queue */
     DELETE FROM C$INQ WHERE CURRENT OF PI_CUR;
   END LOOP;
END;
```

3.19.1.2.2 Out Queue Compose Phase Processing Within the `DOWNLOAD_INIT` procedure, develop a method of composing all changes from the server that are destined for the client from the publication item tables in the repository. The FServ example performs the following:

1. From the Temporary Table `C$PUB_LIST_Q`, discover the publication items that you should download data for the user using the client id, current and last transaction.
2. Retrieves the application publication item data into the Out Queue. This example always uses complete refresh.

```
PROCEDURE DOWNLOAD_INIT( CLIENTID IN VARCHAR2,
                         LAST_TRAN IN NUMBER,
                         CURR_TRAN IN NUMBER,
                         HIGH_PRTY IN VARCHAR2 ) IS
/*create cursor used later in procedure which retrieves the publication name
  from the temporary table to perform compose phase.*/
CURSOR PI_CUR IS SELECT NAME from C$PUB_LIST_Q;
/*create variables*/
PI_NAME VARCHAR2(50);
STATID_CLOSE NUMBER;

BEGIN

   OPEN PI_CUR;
   /* C$PUB_LIST_Q (the temporary table) is used to find out which pub items
      have data to download to clients through the publication item out queue.
      The publication item name is available in the NAME column
    */
   LOOP
     FETCH PI_CUR INTO PI_NAME;
```

```
      EXIT WHEN PI_CUR%NOTFOUND;

      /* Populate the out queue of pub item PI_FSERV_TASKS with all
         unclosed tasks for the employee with this CLIENTID using a complete
         refresh. COMP_REF is always reset to Y since partial refresh has
         not been implemented.
       */
      /* if the PI_FSERV_TASKS publication item has data ready for the client,
         then perform a complete refresh and place all data in the out queue */
      IF PI_NAME = 'PI_FSERV_TASKS' THEN
         UPDATE C$PUB_LIST_Q SET COMP_REF='Y' where NAME = 'PI_FSERV_TASKS';
         SELECT ID INTO STATID_CLOSE FROM MASTER.TASK_STATUS
              WHERE DESCRIPTION='CLOSED';
         INSERT INTO CTM$PI_FSERV_TASKS(CLID$$CS, ID, EMP_ID, CUST_ID,
              STAT_ID, NOTES, TRANID$$, DMLTYPE$$)
              SELECT CLIENTID, a.ID, a.EMP_ID, a.CUST_ID, a.STAT_ID, a.NOTES,
                CURR_TRAN, 'I' FROM MASTER.TASKS a, MASTER.EMPLOYEES b
                WHERE a.STAT_ID < STATID_CLOSE AND b.CLIENTID = CLIENTID
                AND a.EMP_ID = b.ID;
      END IF;
    END LOOP;
END;
```

If, however, you want to perform another type of refresh than a complete refresh, such as an incremental refresh, then do the following:

1. Read the value of COMP_REF
2. If the value is N, insert only the new data into the Out Queue.

In this situation, the LAST_TRAN parameter becomes useful.

3.19.1.3 Create a Publication Item as a Queue

You create the publication item as you would normally, with one change: define the publication item as queue-based. See Section 5.4, "Create a Publication Item" for directions on how to define the publication item as queue-based when using MDW.

If you are using the Consolidator APIs, then the createQueuePublicationItem method creates a publication item in the form of a queue. This API call registers the publication item and creates CFM$<name> table as an In Queue, if one does not exist.

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

You must provide the Consolidator Manager with the primary key, owner and name of the base table or view in order to create a queue that can be updated or refreshed with fast-refresh. If the base table or view name has no primary key, one can be specified in the primary key columns parameter. If primary key columns parameter is null, then Consolidator Manager uses the primary key of the base table.

3.19.1.4 Register the PL/SQL Package Outside the Repository

Once you finish developing the PL/SQL package, register it using the registerQueuePkg method. This method registers the package separately from the Mobile Server repository; although it refers to the in queues, out queues, queue control table and temporary table that are defined in the repository.

The following methods register or remove a procedure, or retrieve the procedure name.

- The `registerQueuePkg` method registers the string `pkg` as the current procedure. The following registers the FServ package.

> **Note:** The developer used Consolidator Manager APIs to create the subscription, so this was included in the Java application that created the subscription.

```
/* Register the queue package for this publication */
  consolidatorManager.registerQueuePkg(QPKG_NAME, PUB_FSERV);
```

- The `getQueuePkg` method returns the name of the currently registered procedure.
- The `unRegisterQueuePkg` method removes the currently registered procedure.

> **Note:** See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.19.2 Creating Data Collection Queues for Uploading Client Collected Data

If you have an application where all it does it collect data, such as taking inventory or uploading collection on any meter (for example, a parking meter), then you can use data collection queues to improve the performance of uploading the data collected to the server. Since the data only flows from the client to the server, then synchronous communication is the best method for uploading massive amounts of data.

> **Note:** If you are collecting data on the client, but still need updates from the server, you can use the default method for synchronization or create your own queues. See Section 3.19.1, "Customizing Apply/Compose Phase of Synchronization with a Queue-Based Publication Item" for more information.

The Data Collection Queue is lightweight and simple to create. You can customize whether the data is implicitly applied or not. This queue does not require the MGP to apply the changes. It does not create objects in the application schema or map data.

Data Collection Queues are also easier to implement than a Queue-Based publication item. There is no need to create a package with callback methods, as Oracle Database Lite takes care of automatically uploading any new data from the client. In addition, you configure how Oracle Database Lite handles if there is any data to be downloaded or if you want the data on the client to be erased when it is uploaded to the server.

When you create the Data Collection Queue, the following is performed for you:

- Automatically generates the in-queue when the publication item is created, which is named as follows: `CFM$<publication_item_name>`.
- Optionally, enables the developer to choose automatic removal of client data once captured to the server. This is specified when you create the publication item.
- Optionally, if you need an out-queue, then the developer can specify the out-queue or to have Oracle Database Lite automatically generate an out-queue, which would be named as follows: `CTM$<publication_item_name>`.

3.19.2.1 Creating a Data Collection Queue

When you create a data collection queue, you perform the following:

> **Note:** All `ConsolidatorManager` methods are fully documented in the Oracle Database Lite API Javadoc. This section provides context of the order in which to execute these methods.

1. Create the table(s) for the data that the queue updates on the back-end Oracle database.
2. Create the data collection queue and its publication item using the `ConsolidatorManager` `createDataCollectionQueue` method, where the input parameters are as follows:
   - `name`—A character string specifying a new publication item name.
   - `owner`—A string specifying the base schema object owner.
   - `store`—A string specifying the table name that it is based on.
   - `inq_cols`—A string specifigying columns in the order in which to replicate them. If null, then defaults to `*`, which makes the SQL statement, `select * from <table>`.
   - `pk_columns`—A string specifying the primary keys.
   - `client_data`—If true, removes client data from the Mobile device when uploaded to the server.
   - `isOutView`—If true, then creates out queue as an empty view, otherwise creates out queue as a table.

   The following creates the `PI_CUSTOMERS` data collection queue:

   ```
   cm.createDataCollectionQueue( "PI_CUSTOMERS", /* Publication Item name */
          MYSCHEMA,                           /* Schema owner */
         "CUSTOMERS",                         /* store */
          null,                               /* inqueue_columns
          null,                               /* null selects all pk_columns
          true,                               /* removes old data after sync
          true );                             /* isOutView */
   ```

3. Create the publication that is to be used by the data collection queue. Use the `ConsolidatorManager` `createPublication` method. The following creates the `PUB_CUSTOMERS` publication that is used by the `PI_CUSTOMERS` data collection queue:

   ```
   cm.createPublication("PUB_CUSTOMERS",0, "sales.%s", null);
   ```

4. Add the publication item created within step 1 within this publication with the `ConsolidatorManager` `addPublicationItem` method. The following adds a publication item to the publication:

   ```
   cm.addPublicationItem("PUB_CUSTOMERS", "PI_CUSTOMERS", null, null,
                   "S", null, null);
   ```

5. If you want to have data download from the server to the Mobile client, create an Out Queue with a name that consists of `CTM$<publication_item_name>`. The following replaces the default out queue view for `CUSTOMER` with a view that selects all customers assigned to the `EMP_ID` associated with current sync session.

```
stmt.executeUpdate(
    "CREATE OR REPLACE VIEW CTM$"+pubIs[0]+" ( CLID$$CS, TRANID$$,
          DMLTYPE$$,"+" CUST_ID, CNAME, CCOMPANY, CPHONE, CCONTACT_DATE )"+"\n
         AS SELECT CONS_EXT.GET_CURR_CLIENT, 999999999, 'I',cust.*
        FROM CUSTOMERS cust "+"\n
        WHERE cust.CUST_ID IN (SELECT CUST_ID
        FROM CUSTOMER_ASSIGNMENT WHERE EMP_ID IN "+"\n
        (SELECT EMP_ID FROM SESSION_EMP
             WHERE SESSION_ID = DBMS_SESSION.UNIQUE_SESSION_ID))"
);
```

Note: See the Oracle Database Lite samples page for the full data collection queue example from which these snippets were taken. The example demonstrates both a regular queue and a data collection queue.

3.19.3 Selecting How/When to Notify Clients of Composed Data

If you have created your own compose logic, such as in the queue-based publications, then you may want the server to notify the client that there is data to be downloaded. You can take control of starting an automatic synchronization from the server using the enqueue notification APIs.

There are other situations where you may want to control how and when clients are notified of compose data from the synchronization process. For example, if you have so many clients that to notify all of them of the data waiting for them would overload your system, you may want to control the process by notifying clients in batches.

In the normal synchronization process, when the compose phase is completed, all clients that have data in the out queue are notified to download the data. If, for example, you have 2000 clients, having all 2000 clients request a download at the same time could overrun your server and cause a performance issue. In this scenario, you could take control of the notification process and notify 100 clients at a time over the span of a couple of hours. This way, all of the clients receive the data in a timely fashion and your server is not overrun.

You can use the enqueue notification functionality, as follows:

- If you implement queue-based publications for the compose phase, you can notify the clients with the `EN_QUEUE_NOTIFICATION` function within the Queue-based `DOWNLOAD_INIT` function.
- If you write your own compose function, use the `enQueueNotification` method to notify the client that there is data to download.

This starts an automatic synchronization process for the intended client.

The enqueue notification APIs enable the server to tell the client that there is data to be downloaded and what type of data is waiting. Notifying the client of what type of data is waiting enables the client to evaluate whether it conforms to any automatic synchronization rules. For example, if the server has 10 records of low priority data, but the client has set the Server MGP Compose rule to only start an automatic synchronization if 20 records of low priority data exist, then the automatic synchronization is not started. So, the notification API input parameters include parameters that enable the server to describe the data that exists on the server.

A notification API is provided for you in both PL/SQL and Java, as follows:

- **Java**: the `ConsolidatorManager enQueueNotification` method

```
public long enQueueNotification(java.lang.String clientid,
                                java.lang.String publication,
                                java.lang.String pubItems,
                                int recordCount,
                                int dataSize,
                                int priority)
                         throws ConsolidatorException
```

- **PL/SQL**: the `EN_QUEUE_NOTIFICATION` function

```
FUNCTION EN_QUEUE_NOTIFICATION(
  CLIENTID        IN VARCHAR2,
  PUBLICATION     IN VARCHAR2,
  PUB_ITEMS       IN VARCHAR2,
  RECORD_COUNT    IN NUMBER,
  DATA_SIZE       IN NUMBER,
  PRIORITY        IN NUMBER)
RETURN NUMBER;
```

Where the parameters for the above are as follows:

Table 3–29 Enqueue Notification Parameters

Parameters	Description
`clientid`	Consolidator client id, which is normally the username on the client device. This identifies the client to be notified. If the client does not have any automatic synchronization rules, this is the only required paramter for an automatic synchronization to start.
`publication`	Name of the publication for which you want notification control. This tells the client for which publication the data is destined.
`pubItems`	One or more publication items for which you want notification. Separate multiple publication items with a comma. This notifies the clients for which publication items the data applies.
`recordCount`	This notifies the client how many records exist on the server for the download.
`dataSize`	Reserved for future expansion.
`priority`	This notifies the client of the priority of the data that exists on the server. The value is 0 for high and 1 for low.

The enqueue notification API returns a unique notification ID, which can be used to query notification status in the `isNotificationSent` method, which is as follows:

- JAVA

```
public boolean isNotificationSent(long notificationId)
  throws ConsolidatorException
```

- PL/SQL

```
FUNCTION NOTIFICATION_SENT(
   NOTIFICATION_ID IN NUMBER)
RETURN BOOLEAN;
```

If the notification has been sent, a boolean value of TRUE is returned.

3.20 Deleting a Client Device

If you want to delete a device, use the `delete` method from the `Device` class. To retrieve the `Device` object, use either the `getDevice` or `getDeviceByName` methods, as demonstrated below.

If the device id is available, the following can be directly used:

```
if (oracle.lite.resource.ResourceManager.getInstance() == null)
oracle.lite.resource.ResourceManager.initialize(JDBC_URL, USER, PASSWORD);

oracle.lite.resource.Device d =
 oracle.lite.resource.ResourceManager.getInstance().getDevice(deviceId);

d.delete();
```

If the device id is not available, then you can provide the device name, which is shown on the Mobile Manager UI in the `oracle.lite.resource.User.getDeviceByName(deviceName)` method. Once retrieved, use the delete method of the Device object as demonstrated above.

3.21 Synchronization Performance

There are certain optimizations you can do to increase performance. See Section 1.2 "Increasing Synchronization Performance" in the *Oracle Database Lite Troubleshooting and Tuning Guide* for a full description.

3.22 Troubleshooting Synchronization Errors

The following section can assist you in troubleshooting any synchronization errors:

- Section 3.22.1, "Foreign Key Constraints in Updatable Publication Items"

3.22.1 Foreign Key Constraints in Updatable Publication Items

Replicating tables between Oracle database and clients in updatable mode can result in foreign key constraint violations if the tables have referential integrity constraints. When a foreign key constraint violation occurs, the server rejects the client transaction.

- Section 3.22.1.1, "Foreign Key Constraint Violation Example"
- Section 3.22.1.2, "Avoiding Constraint Violations with Table Weights"
- Section 3.22.1.3, "Avoiding Constraint Violations with BeforeApply and After Apply"

3.22.1.1 Foreign Key Constraint Violation Example

For example, two tables `EMP` and `DEPT` have referential integrity constraints. The `DeptNo` (department number) attribute in the `DEPT` table is a foreign key in the `EMP` table. The `DeptNo` value for each employee in the `EMP` table must be a valid `DeptNo` value in the `DEPT` table.

A Mobile Server user adds a new department to the `DEPT` table, and then adds a new employee to this department in the `EMP` table. The transaction first updates `DEPT` and then updates the `EMP` table. However, the database application does not store the sequence in which these operations were executed.

When the user replicates with the Mobile Server, the Mobile Server updates the `EMP` table first. In doing so, it attempts to create a new record in `EMP` with an invalid foreign key value for DeptNo. Oracle database detects a referential integrity violation. The Mobile Server rolls back the transaction and places the transaction data in the Mobile Server error queue. In this case, the foreign key constraint violation occurred because the operations within the transaction are performed out of their original sequence.

Avoid this violation by setting table weights to each of the tables in the master-detail relationship. See Section 3.22.1.2, "Avoiding Constraint Violations with Table Weights" for more information.

3.22.1.2 Avoiding Constraint Violations with Table Weights

Mobile Server uses table weight to determine in which order to apply client operations to master tables. Table weight is expressed as an integer and are implemented as follows:

1. Client INSERT operations are executed first, from lowest to highest table weight order.
2. Client DELETE operations are executed next, from highest to lowest table weight order.
3. Client UPDATE operations are executed last, from lowest to highest table weight order.

In the example listed in Section 3.22.1.1, "Foreign Key Constraint Violation Example", a constraint violation error could be resolved by assigning DEPT a lower table weight than EMP. For example:

```
(DEPT weight=1, EMP weight=2)
```

You define the order weight for tables when you add a publication item to the publication. For more information on setting table weights in the publication item, see Section 3.4.1.7.2, "Using Table Weight".

3.22.1.3 Avoiding Constraint Violations with BeforeApply and After Apply

You can use a PL/SQL procedure avoid foreign key constraint violations based on out-of-sequence operations by using DEFERRABLE constraints in conjunction with the BeforeApply and AfterApply functions. See Section 3.11.3.2, "Defer Constraint Checking Until After All Transactions Are Applied" for more information.

3.23 Datatype Conversion Between the Oracle Server and Client Oracle Lite Database

Before you publish your application, you create the tables for your applications in the Oracle database. Thus, when the first synchronization occurs, Oracle Database Lite takes the Oracle database datatypes and converts them to corresponding allowed datatypes in the Oracle Lite database on the client. Table 3–30 lists the Oracle database datatypes in the left column and displays how the datatype can be mapped to the Oracle Lite database datatypes across the top row.

Note: For Oracle Database Lite Datatypes, see Appendix D, "Oracle Database Lite Datatypes" in the *Oracle Database Lite SQL Reference*.

Table 3–30 Conversion of Oracle Database Datatypes to Oracle Database Lite Datatypes

Oracle Database Lite Datatypes	1 B	2 B	4 B	Float	Double	Number	Date Time	Long Var Binary	Varchar	Char	BLOB	CLOB
INTEGER			X									
VARCHAR2									X			
VARCHAR									X			
CHAR										X		
SMALLINT		X										
FLOAT				X								
DOUBLE PRECISION					X							
NUMBER		X	X			X						
DATE							X					
LONG RAW								X				
LONG									X			
BLOB											X	
CLOB												X

Note: Oracle Database Lite does not support creating publication items for synchronization on a table with object type columns, even if the publication item query does not include any of the object type columns. However, it is possible to define a view which selects only columns of supported data types and then create a publication item using the view definition.

"X" indicates that the datatype can be mapped to this Oracle Lite database datatype. To save on space, signed 1 byte represents **TINYINT**, signed 2 byte represents **SMALLINT**, and signed 4 byte represents **INTEGER**.

For conversion of the **NUMBER** datatype, if the precision is less than 5, then the number maps to a signed 2 byte (`SMALLINT`) datatype. If the precision is less than 10, then it maps to a signed 4 bytes (`INTEGER`) datatype. Even though the numbers are not equivalent on the client and the server, we still guarantee that valid numbers from the server will transfer to the client, and invalid numbers from the client are rejected by the server.

While the `TIMESTAMP` data type is supported; the `TIMESTAMP WITH TIME ZONE` is not supported for publication items.

4

Invoking Synchronization APIs from Applications

The following sections describe the APIs available to start synchronization programmatically within your application, whether the application is C, C++, or Java:

- Section 4.1, "Synchronization APIs For C or C++ Applications"
- Section 4.2, "Synchronization API for Java Applications"
- Section 4.3, "msync/OCAPIs/mSyncCom"

4.1 Synchronization APIs For C or C++ Applications

You can initiate and monitor synchronization from a C or C++ client application. The synchronization methods for the C/C++ interface are contained in `ocapi.h` and `ocapi.dll`, which are located in the <*ORACLE_HOME*>`\Mobile\bin` directory.

A C++ example is provided in the <*ORACLE_HOME*>`\Mobile\Sdk\Samples\sync\msync\src` directory. The source code is contained in `SimpleSync.cpp`. The executable—`SimpleSync.exe`—is in the <*ORACLE_HOME*>`\Mobile\Sdk\Samples\sync\msync\bin` directory.

The functions available for setting up and initiating the synchronization are as follows:

1. Section 4.1.1, "Overview of Synchronization API"
2. Section 4.1.2, "Initializing the Environment With ocSessionInit"
3. Section 4.1.3, "Managing the C/C++ Data Structures"
4. Section 4.1.4, "Retrieving Publication Information With ocGetPublication"
5. Section 4.1.5, "Managing User Settings With ocSaveUserInfo"
6. Section 4.1.6, "Manage What Tables Are Synchronized With ocSetTableSyncFlag"
7. Section 4.1.7, "Configure Proxy Information"
8. Section 4.1.8, "Start the Synchronization With the ocDoSynchronize Method"
9. Section 4.1.9, "Clear the Synchronization Environment Using ocSessionTerm"
10. Section 4.1.10, "Retrieve Synchronization Error Message with ocGetLastError"

4.1.1 Overview of Synchronization API

The Synchronization API does not run under the `eshell.exe`. For starting synchronization, the application performs the following:

1. Create, memset, and initialize the `ocEnv` structure.
2. Set any optional fields in the `ocEnv` structure before you execute the ocSessionInit method. Additionally, you can set proxy information in the `ocSetSyncOption` method, or optionally specify synchronization type for each table with the `ocSetTableSyncFlag` function.
3. Invoke the `ocSessionInit()` method.
4. If you want to update the `ocEnv` structure after the ocSessionInit, you can perform the `ocSaveUserInfo` method.
5. Invoke the `ocDoSynchronize()` method, which returns after the synchronization completes, an error occurs, or the user interrupts the process. While executing, the `ocDoSynchronize` function invokes any callback function set in the `ocEnv.fnProgress` field. The callback function must not call any blocking functions, as this process is not reentrant or threaded.
6. Once synchronization completes, then invoke the `ocSessionTerm()` method to clear the `ocEnv` data structure.
7. If synchronization failed, then use the `ocGetLastError` function to retrieve the error message.

For an example, see the `msync.cpp` sample code.

4.1.2 Initializing the Environment With ocSessionInit

The `ocSessionInit` function initializes the synchronization environment—which is contained in the `ocEnv` structure or was created with `ocSaveUserInfo`. For more information, see Section 4.1.5, "Managing User Settings With ocSaveUserInfo".

Note: Every time you invoke the `ocSessionInit` function, you must also clean up with `ocSessionTerm`. These functions should always be called in pairs. See Section 4.1.9, "Clear the Synchronization Environment Using ocSessionTerm" for more information.

Syntax

```
int ocSessionInit( ocEnv env );
```

Table 4–1 lists the `ocSessioninit` parameter and its description.

Table 4–1 ocSessionInit Parameters

Name	Description
`env`	An `ocEnv` class, which contains the synchronization environment.

This call initializes the `ocEnv` structure—which holds context information for the synchronization engine—and restores any user settings that were saved in the last `ocSaveUserInfo` call, such as username and password (See Section 4.1.5, "Managing User Settings With ocSaveUserInfo"). An `ocEnv` structure is passed as the input parameter. Perform the following to prepare the `ocEnv` variable:

1. Create the `ocEnv` by allocating a variable the size of `ocEnv`.
2. Memset the `ocEnv` variable before invoking the `ocSessionInit` function. If you do not perform a `memset` on the `ocEnv` variable, then the `ocSessionInit` function will not perform correctly.

3. Set all required fields in the ocEnv structure before passing it to ocSessionInit. If the caller wants to overwrite user preference information after the ocSessionInit() call, it can be done by calling ocSaveUserInfo.

For a full description of ocEnv, see Section 4.1.3.1, "ocEnv Data Structure".

The following example allocates a new ocEnv, which is then passed into the ocSessionInit call.

```
env = new ocEnv;
// Reset ocenv
memset( env, 0, sizeof(ocEnv) );

// init OCAPI
ocError rc = ocSessionInit(env);
```

4.1.3 Managing the C/C++ Data Structures

Two data structures—ocEnv Data Structure and ocTransportEnv Data Structure—are used for certain functions in the Mobile Sync API.

4.1.3.1 ocEnv Data Structure

The ocEnv data structure holds internal memory buffers and state information. Before using this structure, the application initializes it by passing it to the ocSessionInit method.

Table 4–2 lists the field name, type, usage, and corresponding description of the ocEnv structure parameters.

- Required—If the usage is required, then you either set before calling the ocSessionInit function or you have saved these parameters previously with the ocSaveUserInfo function.
- Optional—If the usage is optional, then optionally set after calling the ocSessionInit function and before the ocDoSynchronize function.
- Read Only.

Table 4–2 ocEnv Structure Field Parameters

Field	Type	Usage	Description
username	char[32]	Required.	Name of the user to authenticate. This name is limited to 28 characters, because of other parts of the product.
password	char[32]	Required.	User password (clear text). This name is limited to 28 characters, because of other parts of the product.
trType	Enum	Required.	If set to 0 (OC_BUILDIN_HTTP), then use HTTP built-in transport driver. This is the default. If set to OC_USER_METHOD, then use user provided transport functions.
newPassword	char[32]	Optional.	If first character of this string is not null—in otherwords (char) 0—this string is sent to the server to change the user password; the password change is effective on the next synchronization session.

Table 4–2 (Cont.) ocEnv Structure Field Parameters

Field	Type	Usage	Description
savePassword	Short	Optional.	If set to 1, the password is saved locally and is loaded the next time `ocSessionInit` is called.
appRoot	`char[32]`	Optional.	Directory to where the application will be copied. If first character is null, then it uses the default directory.
priority	Short	Optional.	0= OFF (default) 1= ON; Only high priority table or rows are synchronized when turned on.
secure	Short	Optional.	If set to 0, then AES is used on the transport. If set to `OC_SSL_ENCRYPTION`, use SSL synchronization (SSL-enabled device only).
syncDirection	Enum	Optional.	If set to 0 (`OC_SENDRECEIVE`), then sync is bi-directional (default). If set to `OC_SENDONLY`, then push changes only to the server. This stops the sync after the local changes are collected and sent. User must write own transport method (like floppy bases) when using this method. If set to `OC_RECEIVEONLY`, then send no changes and only receive update from server. This only performs the receive and allow changes function to local database stages.
exError	ocError	Read-only.	Extended error code - either OS or OKAPI error code.
transportEnv	ocTransportEnv		Transport buffer. See Section 4.1.3.2, "ocTransportEnv Data Structure".
progressProc	fnProgress	Optional.	If not null, points to the callback for progress listening. See Section 4.1.8.1, "See Progress of Synchronization with Progress Listening".
totalSendDataLen	Long		Reserved
totalRecieveDataLen	Long		Reserved
userContext	Void*	Optional.	Can be set to anything by the caller for context information (such as progress dialog handle, renderer object pointer, and so on.
ocContext	Void*		Reserved.
logged	Short		Reserved.
bufferSize	Long		Reserved (for Wireless/Nettech only).
pushOnly	Short	Optional.	If set to 1, then only push changes to the server.
syncApps	Short	Optional.	Set to 1 (by default), performs application deployment. If set to 0, then no applications will be received from the server.

Table 4–2 (Cont.) ocEnv Structure Field Parameters

Field	Type	Usage	Description
syncNewPublications	Short	Optional.	If set to 1 (default), receives any new publication created from the server since last synchronization. If set to 0, only synchronizes existing publications (useful for slow transports like wireless).
clientDbMode	Enum	Optional.	If set to `OC_DBMODE_EMBEDDED` (default), it uses local Oracle Database Lite ODBC driver. If set to `OC_DBMODE_CLIENT`, it uses the Branch Office driver.
syncTimeLog	Short	Optional.	If set to 1, log sync start time is recorded in the `conscli.odb` file.
updateLog	Short	Optional.	Debug only. If set to 1, logs server-side insert and update row information to the publication odb.
options	Short	Optional.	Debug only. A bitset of the following flags: ■ `OCAPI_OPT_SENDMETADATA` Sends meta-info to the server. ■ or `OCAPI_OPT_DEBUG` Enables debugging messages. ■ `OCAPI_OPT_DEBUG_F` Saves all bytes sent and received for debugging. ■ `OCAPI_OPT_NOCOMP` Disables compression. ■ `OCAPI_OPT_ABORT` If set, OCAPI will try to abort the current sync session. ■ `OCAPI_OPT_FULLREFRESH` Forces OCAPI to purge all existing data and do a full refresh.
cancel	Short		Caller can set to 1 on next operation. `ocDoSyncrhonize` returns with -9032.

The environment structure contains fields that the caller can update to change the way Mobile Sync module works. The following example demonstrates how to set the fields within the `ocEnv` structure.

```
typedef struct ocEnv_s {
 // User info
char username[MAX_USERNAME];    // Mobile Sync Client id, limited to 28 characters
char password[MAX_USERNAME];    // Mobile Sync Client password for
                                // authentication during sync, limited to 28 chars
char newPassword[MAX_USERNAME]; // resetting Mobile Sync Client password
                                   // on server side if this field is not blank
short savePassword;           // if set to 1, save password
char appRoot[MAX_PATHNAME];     // dir path on client device for deploying files
short priority;              // High priority table only or not
short secure;            // if set to 1, data encrypted over the wire
enum {
OC_SENDRECEIVE = 0,     // full step of synchronize
OC_SENDONLY,     // send phase only
OC_RECEIVEONLY,     // receive phase only
OC_SENDTOFILE,     // send into local file | pdb
OC_RECEIVEFROMFILE     // receive from local file | pdb
}syncDirection;     // synchronize direction

enum {
OC_BUILDIN_HTTP = 0,     // Use build-in HTTP transport method
OC_USER_METHOD     // Use user defined transport method
}trType;           // type of transport

ocError exError;     // extra error code

ocTransportEnv transportEnv;     // transport control information

                       // GUI related function entry
progressProc fnProgress;     // callback to track progress; this is optional

                 // Values used for Progress Bar. If 0, progress bar won't show.
long totalSendDataLen; // set by Mobile Sync API informing transport total number
                     // of bytes to send; set before the first fnSend() is called
long totalReceiveDataLen;     // to be set by transport informing Mobile Sync API
                       // total number of bytes to receive;
                       // should be set at first fnReceive() call.
void* userContext;     // user defined context
void* ocContext;       // internal use only
short logged;          // internal use only
long bufferSize;       // send/receive buffer size, default is 0
short pushOnly;        // Push only flag
short syncApps;        // Application deployment flag
short cancel;          // cancel
} ocEnv;
```

4.1.3.2 ocTransportEnv Data Structure

You can configure the HTTP URL, proxy, proxy port number and other HTTP-specific transport definitions in the `ocTrHttp` structure. This structure is an HTTP public structure defined in `octrhttp.h`.

You access the `ocTrHttp` structure from within the `ocTransportEnv` data structure, which is provided as part of the `ocEnv` data structure. The following demonstrates the fields within the `ocTransportEnv` structure:

```
typedef struct ocTransportEnv_s {
void* ocTrInfo;            // transport internal context
```

The `ocTrInfo` is a pointer that points to the HTTP parameters in the `ocTrHttp` structure. The following code example retrieves the `ocTrInfo` pointer to the HTTP parameters and then modifies the URL, proxy, and proxy port number to the input arguments:

```
ocTrHttp* http_params = (ocTrHttp*)(env->transportEnv.ocTrInfo);
// set server_name
strcpy(http_params->url, argv[3]);
// set proxy
strcpy(http_params->proxy, argv[4]);
// set proxy port
http_params->proxyPort = atoi(argv[5])
```

4.1.4 Retrieving Publication Information With ocGetPublication

This function gets the publication name on the client from the Web-to-Go application name. The Web-to-Go user knows only the application name, which happens when the Packaging Wizard is used to package an application before publishing it. If the Web-to-Go application needs the publication name in order to interact with the database, then this function is used to retrieve that name, given the application name.

Syntax

```
ocError ocGetPublication(ocEnv* env, const char* application_name,
 char* buf, int buf_len);
```

The parameters for the `ocGetPublication` function are listed in Table 4–3 below.

Table 4–3 ocGetPublication Parameters

Name	Description
`ocEnv* env`	Pointer to an `ocEnv` structure buffer to hold the return synchronization environment.
`const char* application_name(in)`	The name of the application.
`char* buf(out)`	The buffer where the publication name is returned.
`int buf_len(in)`	The buffer length, which must be at least 32 bytes.

Return value of 0 indicates that the function has been executed successfully. Any other value is an error code.

The following code example demonstrates how to get the publication name.

```
void sync()
{
         ocEnv env;
         int rc;

         // Clean up ocenv
         memset(&env 0, sizeof(env) );

         // init OCAPI
         rc = ocSessionInit(&env);

         strcpy(env.username, "john");
         strcpy(env.password, "john");

         // We use transportEnv as HTTP paramters
         ocTrHttp* http_params = (ocTrHttp*)(env.transportEnv.ocTrInfo);
         strcpy(http_params->url, "your_host");

         // Do not sync webtogo applicaton "Sample3"
         char buf[32];
         rc = ocGetPublication(&env, "Sample3", buf, sizeof(buf));
         rc = ocSetTableSyncFlag(&env, buf, NULL, 0);

         // call sync
         rc = ocDoSynchronize(&env);
         if (rc < 0)
                fprintf(stderr, "ocDoSynchronize failed with %d:%d\n",
                    rc, env.exError);
         else
```

```
                printf("Sync compeleted\n");

        // close OCAPI session
        rc = ocSessionTerm(&env);
        return 0;
}
```

4.1.5 Managing User Settings With ocSaveUserInfo

Saves user settings for the `ocEnv` structure. These settings can be used for the current session or used by the `ocSessionInit` function to initialize the environment when next invoked.

Syntax

```
int ocSaveUserInfo( ocEnv *env );
```

Table 4–4 lists the `ocSaveUserInfo` parameter and its description.

Table 4–4 ocSaveUserInfo Parameters

Name	Description
`env`	Pointer to the synchronization environment.

This saves or overwrites the user settings into a file or database on the client side. The following information provided in the environment structure is saved:

Note: See Section 4.1.3.1, "ocEnv Data Structure" or Section 4.1.3.2, "ocTransportEnv Data Structure" for more information.

- `username`
- `password`
- `savePassword`
- `newPassword`
- `priority`
- `secure`
- `pushOnly`
- `syncApps`
- `syncNewPublications`

If you use the HTTP default transport set in the `ocTransportEnv` structure, then the following is also saved:

- `url`
- `useProxy`
- `proxy`
- `proxyPort`

For more information on how to use these fields, see Section 4.1.3, "Managing the C/C++ Data Structures".

4.1.6 Manage What Tables Are Synchronized With ocSetTableSyncFlag

Update the table flags for selective sync. Call this for each table to specify whether it should be synchronized(1) or not (0) for the next session. Selective sync only works if you have first performed at least one synchronization for the client. Then, set the flag so that on the next synchronize—that is, before the next invocation of the `ocDoSynchronize` method—a selective sync occurs.

Note: Automatic synchronization is based on a different model than manual synchronization. Automatic synchronization operates on a transactional basis. Thus, the selective sync option is not supported when you use automatic synchronization for a publication, since we are no longer concerned with synchronization of only a subset of data.

The default `sync_flag` setting for `ocSetTableSyncFlag` is TRUE (1) for all the tables; that is, all tables are flagged to be synchronized. If you want to selectively synchronize specific tables, you must first disable the default setting for all tables and then enable the synchronization for only the specific tables that you want to synchronize.

Syntax

```
ocSetTableSyncFlag(ocEnv *env, const char* publication_name,
               const char* table_name, short sync_flag)
```

Table 4–5 lists the name and description of parameters for the `ocSetTableSyncFlag` function.

Table 4–5 ocSetTableSyncFlag Parameters

Name	Description
`env`	Pointer to the synchronization environment.
`publication_name`	The name of the publication which is being synchronized. If the value for the `publication_name` is `NULL`, it means all publications in the database. This string is the same as the `client_name_template` parameter of the Consolidator Manager `CreatePublication` method. In most cases, you will use `NULL` for this parameter. For more information, see Section 3.4, "Creating Publications Using Oracle Database Lite APIs".
`table_name`	This is the name of the snapshot. It is the same as the name of the `store`, the third parameter of `CreatePublicationItem()`. For more information, see Section 3.4, "Creating Publications Using Oracle Database Lite APIs".
`sync_flag`	If the `sync_flag` is set to 1, you must synchronize the publication. If the `sync_flag` is set to 0, then do not synchronize. The value for the `sync_flag` is not stored persistently. Each time before `ocDoSynchronize()`, you must call `ocSetTableSyncFlag()`.

This function allows client applications to select the way specific tables are synchronized.

Set `sync_flag` for each table or each publication. If `sync_flag` = 0, the table is not synchronized.

To synchronize specific tables only, you must perform the following steps:

1. Disable the default setting, which is set to 1 (TRUE) for all the tables.

Example:

```
ocSetTableSyncFlag(&env, <publication_name>,null,0)
```

Where `<publication_name>` must be replaced by the actual name of your publication, and where the value `null` is specified to mean **all** the tables for that publication without exception.

2. Enable the selective sync for specific tables.

   Example:

   ```
   ocSetTableSyncFlag(&env, <publication_name>,<table_name>,1)
   ```

4.1.7 Configure Proxy Information

If you are using a firewall and need to configure proxy information, perform the following before you execute the `ocDoSynchronize` method:

1. Configure the proxy URL, IP address and/or port number through the `ocSaveUserInfo` function. See Section 4.1.5, "Managing User Settings With ocSaveUserInfo" for more information.
2. If required, configure the proxy username and password. To configure the proxy username and password, use the `ocSetSyncOption` and provide the following:

   ```
   ocSetSyncOption( env, "HTTPUSER=<username>;HTTPPASS=<password>");
   ```

   Note: The username and password are limited to 28 characters.

   Where the `ocSetSyncOption` syntax is as follows:

   ```
   int ocSetSyncOption(ocEnv *env, const char *str);
   ```

You can set one or more name/value pairs searated by a semi-colon in the string. The previous example shows the `HTTPUSER` and `HTTPPASS` name/value pairs. You can also set the URL string as follows: `URL=www.myhost.com`.

4.1.8 Start the Synchronization With the ocDoSynchronize Method

Starts the synchronization process.

Syntax

```
int ocDoSynchronize( ocEnv *env );
```

Table 4–6 lists the name and description of the `ocDoSynchronize` parameter.

Table 4–6 ocDoSynchronize Parameters

Name	Description
`env`	Pointer to the synchronization environment.

This starts the synchronization cycle. A round trip synchronization is activated if `syncDirection` is `OC_SENDRECEIVE` (default). If `syncDirection` is `OC_SENDONLY` or `OC_RECEIVEONLY`, then the developer must implement a custom transport. If the developer wishes to upload only changes, then set `pushonly=1`. You cannot only download changes under the existing synchronization architecture.

This method returns when the synchronize completes. A return value of 0 indicates that the function has been executed successfully. If an error occurred, local errors are returned by `ocDoSynchronize`, which are defined in `ocerror.h`. For errors returned by the server, see the `ol_sync.log` error log file, which is written into the working directory of the application. Each line in the error file has the following format:

```
<type>, <code>, <date>, <message>
```

Where:

- `<type>`: The type of the message, which can either be set to `ERROR` or `SUCCESS`.
- `<code>`: Error code of the last operation of the synchronization.
- `<date>`: Date and timestamp for when the synchronization completes. This is in the format of `dd/mm/yyyy hh:mm:ss`.
- `<message>`: A readable message text.

4.1.8.1 See Progress of Synchronization with Progress Listening

If you create and set the progress callback function, then Oracle Database Lite invokes this callback function at different times while the `ocDoSynchronize` method is executing. Create the callback function, as follows:

```
void myProgressProc ( void *env, int stage, int present);
```

When the `ocDoSynchronize` invokes your `myProgressProc` function, it provides the following information as input to your function:

- `env`—A pointer to the environment (`ocEnv` structure) for the synchronization session. This provides the function to retrieve the `userContext` pointer.
- `stage`—A number that denotes the stage in the synchronization process, which is one of the following values, where these values are defined in `ocapi.h`:

Table 4–7 Description of the Stage Values

Stage Value	Description
`OC_PREPARE_START`	Start of the prepare stage, which collects all internal data from the database and prepares to send the data to the server.
`OC_PREPARING`	Progress in the prepare stage.
`OC_PREPARE_FINISH`	Prepare stage is completed.
`OC_SEND_START`	Starting to send the data to the server.
`OC_SENDING`	Sending the data.
`OC_SEND_FINISH`	Completed sending the data.
`OC_RECEIVE_START`	Starting to receive data.
`OC_RECEIVING`	Receiving data from the server.
`OC_RECEIVE_FINISH`	Completed receiving data from the server.
`OC_PROCESS_START`	Starting to process received data.
`OC_PROCESSING`	Processing received data.
`OC_PROCESS_FINISH`	Completed processing. Synchronization is finished.

- `present`—The percentage completed in the particular stage that synchronization is in from 0 to 100.

If the function is a member of a class, then it must be defined as static.

After you create the callback function, set the function pointer in the `ocEnv.fnProgress` (Table 4–2) to the address of your callback function. Save this with the `ocSaveUserInfo` or `ocSessionInit` methods.

4.1.9 Clear the Synchronization Environment Using ocSessionTerm

Clears and performs a cleanup of the synchronization environment and buffers. This function must be invoked for every `ocSessionInit`, even if the `ocDoSynchronize` function is not performed.

Syntax

```
int ocSessionTerm( ocEnv *env );
```

Table 4–8 lists the `ocSessionTerm` parameter and its description.

Table 4–8 ocSessionTerm Parameters

Name	Description
`env`	Pointer to the environment structure returned by `ocSessionInit`.

De-initializes all the structures and memory created by the `ocSessionInit()` call. Users must ensure that they are always called in pairs.

4.1.10 Retrieve Synchronization Error Message with ocGetLastError

Retrieves the synchronization error message and code.

Syntax

```
int ocGetLastError( ocEnv *env, char *buf, int buf_size);
```

Table 4–9 lists the `ocGetLastError` parameters.

Table 4–9 ocGet Parameters

Name	Description
`env`	Pointer to the environment structure returned by `ocSessionInit`.
`buf`	A string with the error message.
`buf_size`	The size of the error message string.

4.2 Synchronization API for Java Applications

The following sections describe how you can use Java on a WinCE device to build your own client synchronization initiation:

- Section 4.2.1, "Overview"
- Section 4.2.2, "Sync Class"
- Section 4.2.3, "SyncException Class"

- Section 4.2.4, "SyncOption Class"
- Section 4.2.5, "Java Interface SyncParam Settings"
- Section 4.2.6, "Java Interface TransportParam Parameters"
- Section 4.2.7, "Manage What Tables Are Synchronized With Selective Sync"
- Section 4.2.8, "SyncProgress Listener Service"

4.2.1 Overview

Using the Java interface for Mobile Sync client-side synchronization tasks, programs written in Java can use the functionality provided by the OCAPI library. The Java interface resides in the `oracle.lite.msync` package.

The Java interface provides for the following functions:

- Setting client side user profiles containing data such as user name, password, and server
- Starting the synchronization process
- Tracking the progress of the synchronization process

The Java interface consists of two files, `olite40.jar` and `msync_java.dll`. To use the Java interface, the `olite40.jar` file must be included in the `CLASSPATH`. The `olite40.jar` file is located in the following directory.

<*ORACLE_HOME*>`\Mobile\classes`

The `msync_java.dll` file is located in the following directory.

<*ORACLE_HOME*>`\Mobile\bin`

There are four parts to the Java interface. They are:

- `Sync` Class
- `SyncException` Class
- `SyncOption` Class
- `SyncProgressListener` Interface

The following sections describe the Java interface.

4.2.2 Sync Class

This class initiates synchronization by using the provided synchronization options. The parameters for the constructor are listed in Table 4–10.

Constructors

`Sync(SyncOption` *`option`*`)`

Table 4–10 Sync Class Constructor

Parameter	Description
`option`	Instance of the `SyncOption` Class. This contains all the parameters needed to perform synchronization.

Public Methods

To monitor the progress of the synchronization process, the public method `SyncProgressListener` adds a progress listener to the object.

`SyncProgressListener add(ProgressListener listener)`

The parameters for the `SyncProgressListener` method are described in Table 4–11.

Table 4–11 Sync Class Public Method

Parameter	Description
`listener`	An object that implements the `ProgressListener` interface. The synchronization object calls the `progress()` function of this object to notify it of the synchronization progress.
`void doSync ()`	Starts a synchronization session and blocks that thread until synchronization is complete.
`void abort ()`	Aborts the synchronization session.

The following code demonstrates how to start a session using the default settings.

```
try
{
  Sync mySync = new Sync( new SyncOption());
  mySync.doSync();
}
catch ( SyncException e)
{
  System.err.println( "Sync Error:"+e.getMessage());
}
```

4.2.3 SyncException Class

This class signals a non-recoverable error during the synchronization process. The `SyncException()` class constructs a `clear` object. The parameters for the constructor are listed inTable 4–12:

Constructors

`SyncException()`

`SyncException(int errorCode, string errorMessage)`

Table 4–12 syncException Constructor Parameter Description

Parameter	Description
`errorCode`	The error. Refer the *Oracle Database Lite Message Reference*.
`errorMessage`	A readable text message that provides extra information.

Public Methods

The methods for the `SyncException` are listed in Table 4–13.

Table 4–13 SyncExceptionClass Public Methods

Parameters	Description
`int getErrorCode()`	Gets the error code.
`String getErrorMessage`	Gets the error message.

4.2.4 SyncOption Class

The `SyncOption` class is used to define the parameters for the synchronization process. It can either be constructed manually, or can save or load data from the user profile.

Constructors

```
SyncOption()

SyncOption
   ( String user,
     String password,
     String syncParam,
     String transportDriver,
     String transportParam)
```

The parameters for the `SyncOption` constructor are listed in Table 4–14:

Table 4–14 SyncOption Constructors

Parameter	Description
`user`	A string containing the name used for authentication by the Mobile Server.
`password`	A string containing the user password.
`syncParam`	A string which defines an optional list of parameters for the synchronization session. See Section 4.2.5, "Java Interface SyncParam Settings" for more information.
`transportDriver`	A string containing the name of the transport driver. Currently, only "HTTP" is supported.
`transportParam`	A string containing all the parameters needed for the specified driver to operate. See Section 4.2.6, "Java Interface TransportParam Parameters" for more information.
`priority`	A boolean value which limits synchronization to server tables flagged as high priority, otherwise all tables are synchronized.
`pushOnly`	A boolean value which makes synchronization push only.

Public Methods

These methods load and save the user profile. The parameters of the public methods are listed in Table 4–15:

Table 4–15 Sync Option Public Method Parameters

Parameter	Description
`void load(String username)`	This loads the profile for the specified user name. If the user name is left null, the profile is loaded for the last user to synchronize.
`void save()`	This saves the settings to the profile for the active user.
`void setUser(String username)` `String getuser()`	This is used to set and get the current user.
`void setPassword(String password)` `String getPassword()`	This is used to set and get the password.

Table 4–15 (Cont.) Sync Option Public Method Parameters

Parameter	Description
`void setSyncParam(String syncParam)`	This is used to set and get the synchronization parameters.
`string getSyncParam()`	
`void setTransportDriver(String driverName)`	This is used to set and get the driver name. Release 5.0.2 supports the "HTTP" driver.
`String getTransportDriver()`	
`void setTransportParam(String transportParam)`	Set and get the transport parameters.
`String getTransportParam()`	

Example 1

The following code example demonstrates how to start a synchronization session using the default settings:

```
SyncOption opt = new SyncOption

("sam","lion","pushonly","HTTP","server=server1;proxy=www-proxy.us.oracle.com;prox
yPort=80");

opt.save();
```

Example 2

The following example is of a client that creates the `SyncOption` class and then performs the synchronization with the `doSync` method.

```
import oracle.lite.mSync.*;

public class JavaSyncClient{
    String user = "SALES1";
    String password = "MANAGER";
    //Set the Sync params
    //Set syncParam to fullrefresh
    String syncParam = "";//fullrefresh;
    // Set the Transport params
    String transportDriver = "HTTP";
    String trasportParam = "server=localhost";

    /**
     * Constructor
     */
    public JavaSyncClient() throws Exception{
        //Create the SyncOption class
        SyncOption syncOpt = new SyncOption(user, password,
            syncParam, transportDriver, trasportParam);
        syncOpt.setSyncFlag("MYORDERS", "", (short) 0);
        //Save the options before the sync
        syncOpt.save();
        //Create the Sync class
        Sync mySync = new Sync(syncOpt);
        //Perform the synchronization
        mySync.doSync();
    }

    /**
```

```
     * main
     */
    public static void main(String[] args)  throws Exception {
        JavaSyncClient JavaSyncClient = new JavaSyncClient();
    }
}
```

4.2.5 Java Interface SyncParam Settings

The `syncParam` is a string that can be passed when creating the `SyncOption` object. It allows support parameters to be specified to the synchronization session. The string is constructed of name-and-value pairs. For example:

```
"name=value;name2=value2;name3=value3, ...;"
```

The names are not case sensitive, but the values are. The field names which can be used are listed in Table 4–16.

Table 4–16 Java Interface SyncParamSettings

Name	Value/Options	Description
`"reset"`	N/A	Clear all entries in the environment before applying any remaining settings.
`"security"`	`SSL` or `AES`	Use the appropriate selection to choose either SSL or AES stream encryption.
`"push only"`	N/A	Use this setting to upload changes from the client to the server only, do not download. This is useful when data transfer is one way, client to server.
`"noapps"`	N/A	Do not download any new or updated applications. This is useful when synchronizing over slow connection or on a slow network.
`"syncDirection"`	`"sendonly"` `"receiveonly"`	"SendOnly" is the same as "pushonly". "ReceiveOnly" allows no changes to be posted to the server.
"`noNewPubs"`	N/A	This setting prevents any new publications created since the last synchronization from being sent, and only synchronizes data from the current publications.
`"tableFlag"`	`"enable"`	The "enable" setting allows [Publication.Item] to be synchronized, "disable" prevents synchronization.
`[Publication.Item]`	`"disable"`	
`"fullrefresh"`	N/A	Forces a complete refresh.
`"clientDBMode"`	`"EMBEDDED"` or `"CLIENT"`	If set to "EMBEDDED", access to the database is by conventional ODBC, if set to "CLIENT" access is by multi-client ODBC.

Example 1

The first example enables SSL security and disables application deployment for the current synchronization session:

```
"security=SSL; noapps;"
```

Example 2

The second example resets all previous settings, activates upload for the "Dept" table only:

```
"reset;pushOnly;tableFlag[TestApp.Emp]=disable;tableFlag[TestApp.Dept]=enable;"
```

4.2.6 Java Interface TransportParam Parameters

The format of the `TransportParam` string is used to set specific parameters using a string of name-and-value pairs, for example:

```
"name=value;name2=value2;name3=value3, ...;"
```

The names are not case sensitive, but the values are. The field names which can be used are listed in Table 4–17.

Table 4–17 TransportParam Parameters

Name	Value	Description
`"reset"`	N/A	Clear all entries in the environment before applying the rest of the settings.
`"server"`	server hostname	The hostname or IP address of the Mobile Server.
`"proxy"`	proxy server hostname	The hostname or IP address of the proxy server.
`"proxyPort"`	port number	The port number of the proxy server.
`"cookie"`	cookie string	The cookie to be used for transport.

Example

The example directs the Mobile Sync engine to use the server at "`test.oracle.com`" through the proxy "`proxy.oracle.com`" at port 8080:

```
"server=test.oracle.com;proxy=proxy.oracle.com;proxyPort=8080;"
```

4.2.7 Manage What Tables Are Synchronized With Selective Sync

Update the table flags for selective sync. Call this for each table to specify whether it should be synchronized(1) or not (0) for the next session. Selective sync only works if you have first performed at least one synchronization for the client. Then, set the flag so that on the next synchronize—that is, before the next invocation of the `doSynchronize` method—a selective sync occurs.

The default setting is TRUE (1) for all the tables; that is, all tables are flagged to be synchronized. If you want to selectively synchronize specific tables, you must first disable the default setting for all tables and then enable the synchronization for only the specific tables that you want to synchronize.

Note: Automatic synchronization is based on a different model than manual synchronization. Automatic synchronization operates on a transactional basis. Thus, the selective sync option is not supported when you use automatic synchronization for a publication, since we are no longer concerned with synchronization of only a subset of data.

Syntax

```
public void setSyncFlag(java.lang.String publication_name,
        java.lang.String table_name,
        short sync_flag) throws SyncException
```

Table 4–5 lists the name and description of parameters for the `setSyncFlag` function.

Table 4–18 setSyncFlag Parameters

Name	Description
`publication_name`	The name of the publication which is being synchronized. If the value for the `publication_name` is `NULL`, it means all publications in the database. This string is the same as the `client_name_template` parameter of the Consolidator Manager `createPublication` method. In most cases, you will use `NULL` for this parameter. For more information, see Section 3.4, "Creating Publications Using Oracle Database Lite APIs".
`table_name`	This is the name of the snapshot. It is the same as the name of the `store`, the third parameter of `createPublicationItem()`. For more information, see Section 3.4, "Creating Publications Using Oracle Database Lite APIs".
`sync_flag`	If the `sync_flag` is set to 1, you must synchronize the publication. If the `sync_flag` is set to 0, then do not synchronize. The value for the `sync_flag` is not stored persistently. Each time before `doSynchronize()`, you must call `setSyncFlag()`.

This function allows client applications to select the way specific tables are synchronized.

Set `sync_flag` for each table or each publication. If `sync_flag` = 0, the table is not synchronized. To synchronize specific tables only, you must perform the following steps:

1. Disable the default setting, which is set to 1 (TRUE) for all the tables.

   Example:

   ```
   setSyncFlag(<publication_name>,null,0)
   ```

   Where `<publication_name>` must be replaced by the actual name of your publication, and where the value `null` is specified to mean **all** the tables for that publication without exception.

2. Enable the selective sync for specific tables.

   Example:

   ```
   setSyncFlag(<publication_name>,<table_name>,1)
   ```

Alternatively, see the following code snippet on how to enable the selective sync flag for EVERY table EXCEPT the `OrdersODB.TEST` table.

```
SyncOption op = new SyncOption(user, passwd,
                     "noNewPubs","HTTP",server.toString());
op.setSyncFlag("","",(short)1); //turn on sync flag for all the tables
op.setSyncFlag("","OrdersODB.TEST",(short)0);
                      //turn off sync flag for OrdersODB.TEST
```

4.2.8 SyncProgress Listener Service

The SyncProgressListener is an interface that allows progress updates to be trapped during synchronization.

This class initiates synchronization by using the provided synchronization options. The parameters for the method are listed in Table 4–19:

Method

```
void progress

   (int progressType,
    int completed);
```

Table 4–19 SyncProgressListener Abstract Method

Parameter	Description
`progressType`	This is set to one of the constants listed in Table 4–20.
`completed`	This is the percentage of completion for specific `progressType`.

The names of the constants which report the synchronization progress are listed in Table 4–20.

Table 4–20 SyncProgressListener Interface Constants

Constant Name	Progress Type
`PT_INT`	States that the synchronization engine is in the initializing stage. The current and total counts are set to 0.
`PT_PREPARE_SEND`	States that the synchronization engine is preparing local data to be sent to the server. This includes getting locally modified data. For streaming implementations this takes a shorter amount of time.
`PT_SEND`	States that the synchronization engine is sending data to the network. The total count equals the number of bytes to be sent, and the current count equals the byte count being sent currently.
`PT_RECV`	States that the synchronization engine is receiving data from the server. The total count equals the number of bytes to be received, and the current count equals the byte count being received currently.
`PT_PROCESS_RECV`	States that the synchronization engine is applying the newly received data from the server to the local data stores.
`PT_COMPLETE`	States that the synchronization engine has completed the synchronization process.

Example

This simple class implements the `SyncProgressListener`.

```
class myProgressTracker implements SyncProgress Listener;

{
  public void progress
      (int progressType,
      int completed)
     {
       System.out.println( "Status: "+progressType+"="+ completed+"%" );
      } //progress
 }
```

4.3 msync/OCAPIs/mSyncCom

For more information, refer to the *Oracle Database Lite API Specification*.

5 Using Mobile Database Workbench to Create Publications

The following sections describe how to use the Mobile Database Workbench (MDW) to create publications. When using MDW, you first create a project and then create the other objects contained within a publication.

5.1 Use MDW to Create Publications

The Mobile Database Workbench (MDW) is a new tool that enables you to iteratively create and test publications—testing each object as you add it to a publication. Publications are stored within a project, which can be saved and restored from your file system, so that you can continue to add and modify any of the contained objects within it.

All work is created within a project, which can be saved to the file system and retrieved for further modifications later. Once you create the project, start creating the publication items, sequences, scripts and resources that are to be associated with the publication. You can create the publication and associated objects in any order, but you always associate an existing object with the publication. Thus, it saves time to start with creating the objects first and associating it with the publication afterwards.

The following describes how to launch MDW:

- Section 5.1.1, "Set Access Privileges to SYSTEM Tables for Your Application Schema"
- Section 5.1.2, "Launch MDW"

5.1.1 Set Access Privileges to SYSTEM Tables for Your Application Schema

Before you start up MDW, ensure that the application schema has the correct privileges to the following SYSTEM tables:

- `all_views`
- `all_objects`
- `all_synonyms`
- `all_tables`
- `all_constraints`
- `all_dependencies`

When you create a SQL statement in a publication item using MDW, then MDW checks the dependencies using the SYSTEM tables. So, if you have not set the privileges for the application schema to the SYSTEM tables, you may receive the ORA-1031 "Insufficient privileges" error message.

5.1.2 Launch MDW

To launch MDW, execute `oramdw`, which is located in *$ORACLE_HOME*`\Mobile\Sdk\bin`.

5.2 Create a Project

Create a new project with the Project Wizard. The project is the vehicle that contains your iterative approach to defining publications, publication items, sequences, scripts and resources. The project can be saved and restored from your file system, so that you can continue to modify any of the contained objects within it.

You cannot perform any action on developing your publications without first creating the project.

You must have access to the back-end database with the Oracle Mobile Repository and already defined the tables and schema that you are going to be using in your publication items before entering the project wizard.

Perform the following to create the project:

1. Click **File->New->Project** to start the Project Wizard.
2. An Introductory screen appears. If you do not want this introductory screen to display each time you start a new project, check the "Skip This Page Next Time" box.
3. Define the project name. Enter the project name and location for your new project, as follows:
   - Project name: This name can be any valid Java identifier. The name cannot contain any spaces. For example, your project name may be something like `MY_NEW_PROJECT`.

- Project location: Enter a valid location in the directory on your machine that has write permission to store the project. On a Windows machine, you could enter the location as `c:\myprojects`. To browse for a directory, click **Browse**.

Click **Next** to move to the next step in the wizard.

4. Provide the Mobile Repository access information. Because you are interacting with the repository to create and manipulate synchronization objects, including the SQL scripts for the publication items, you need access to the Mobile Repository. Enter the following access information:

   - User name and password: Specify the Mobile Server repository user name and password (with administrator privilege). This is the same user name and password with which the repository was originally created. For example, the default Mobile Server repository user name and password is `mobileadmin/manager`.

     The administrator username and password are used to connect to the back-end database, create the schema and assign database privileges for the Mobile Server. In order to perform these actions, the administrator user must have the following privileges:

     – The following privileges are required with the Admin option:

       ALTER ANY TABLE, ALTER SESSION, ALTER SYSTEM, CREATE SESSION, CREATE ANY SEQUENCE, CREATE ANY VIEW, CREATE ANY TRIGGER, CREATE ANY INDEX, CREATE ANY TABLE, CREATE ANY SYNONYM, CREATE ANY PROCEDURE, CREATE PROCEDURE, CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE, CREATE VIEW, CREATE INDEXTYPE, DELETE ANY TABLE, DROP ANY SEQUENCE, DROP ANY PROCEDURE, DROP ANY VIEW, INSERT ANY TABLE, DROP ANY SYNONYM, DROP ANY TRIGGER, DROP ANY INDEX, DROP ANY TABLE, SELECT ANY TABLE , SELECT ANY DICTIONARY, UPDATE ANY TABLE

       Lastly, the `SELECT_CATALOG_ROLE` role is required with the Admin option.

   - JDBC driver type: Select the JDBC driver that is used to connect to the Oracle database that hosts the Mobile Server repository. At this time, the only driver that you can choose is the Oracle Thin driver.

   - Host name or IP address, port number, and SID: Enter the location, port, and SID of the database that contains the Mobile Server Repository. For the location, you can either enter the host name or the IP address. The port and SID are configured within the definition of the Oracle database and are used to access the database. For example, the host, port and SID could be `my-pc1`, 1521, and `orcl`.

   Click **Next** to move to the next step in the wizard. Once you click Next, the wizard verifies that the database connection information is correct. If incorrect, the wizard prompts you to re-enter the information. You can only advance if you enter the correct information where the Mobile Server Repository is located.

5. Specify schema username and password, each of which are limited to 28 characters. Enter the user and password of the schema owner for the schema that you are using for the Mobile application. The Mobile application schema contains all database tables, views, synonyms used to build the snapshots for the application.

Note: All schema objects for an application exist in the same back-end repository, which is why the Oracle database host, port and SID are only read-only on this screen.

Click **Next** to move on to the last screen in the Project Wizard. As you click **Next**, MDW verifies that the username and password that you entered are valid for connecting to the application schema in the back-end database. If these are not valid, you cannot advance until you supply a valid username and password.

6. A summary page appears. Once the creation of the project is completed, this page displays all of the information about your new project.
   - Click **Back** to modify any of the information supplied.
   - Click **Finish** to complete the project creation.
   - Click **Cancel** to abort creation of this project.

At this point, you can create your publication within this project.

5.3 Use the Quick Wizard to Create Your Publication

The Quick Start Wizard enables you to create a simple publication in just a few steps. It generates the publication items within your publication by assuming that you want the default settings. In addition, the snapshot defaults to select all items within the table. For example, if the table selected is `EMP`, then the select statement defaults to `select * from emp`.

You can associate a publication item in a publication, which is then associated in an application. The publication item is the vehicle that defines the SQL to retrieve data from the database for the application users. When you execute the quick wizard, it creates a publication item for each table you wish to include in the publication. In addition, the wizard defaults the SQL statement used to define the data subset for each table as `select * from <table_name>`.

Note: Since this tool is a quick wizard, it associates a single publication item for each table you include in the publication. In order to create a more complex snapshot—such as one that enables automatic synchronization, creates multiple publication items based on the same table or a more complex SQL statement—see Section 5.4, "Create a Publication Item".

The publication item name defaults to the following: `<table_name>_PI<number>` where `<number>` is sequential between 1 and 9. For example, the first publication item created on table `EMP` would be named `EMP_PI1`. If, in a separate publication, you have already defined a publication item for `EMP_PI1`, then the next time you execute the wizard for the table `EMP`, it will be named `EMP_PI2`.

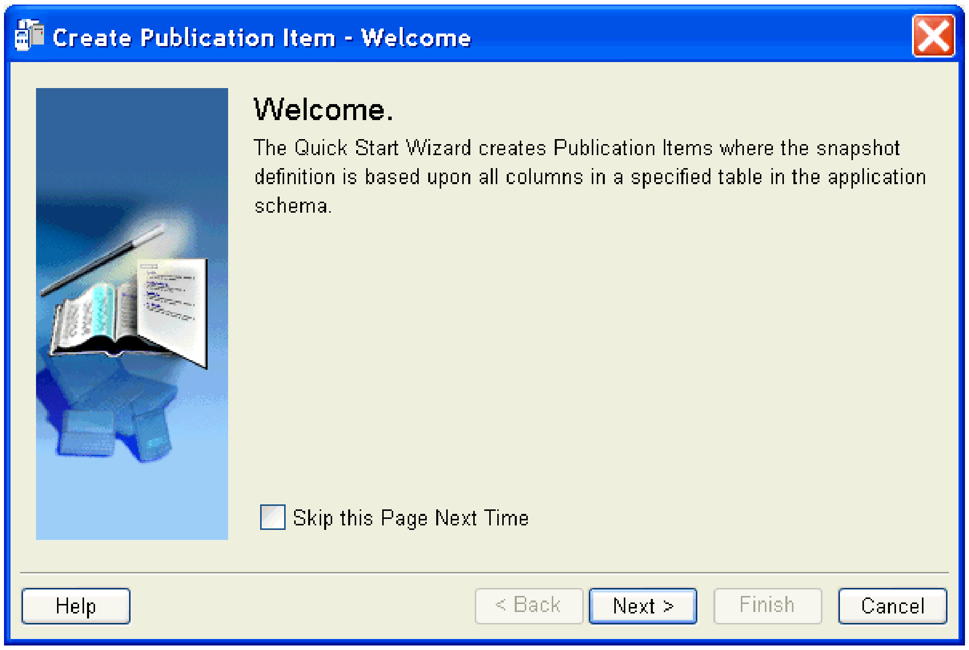
After creating this publication item, this wizard enables you to test it immediately. When the wizard completes, you can always return to the main menu and modify any of the default settings or specify a more specific data subset with your own SQL statement.

For each of the screens in the wizard, click **Next** to advance to the next screen.

1. To start the quick wizard, select the **Quick Wizard** button.

2. An introductory screen appears. If you do not want this introductory screen to display each time you start a new project, check the "Skip This Page Next Time" box.

Figure 5–1 Welcome Screen for Quick Wizard to Create a Publication


Create Publication Item - Welcome

Welcome.

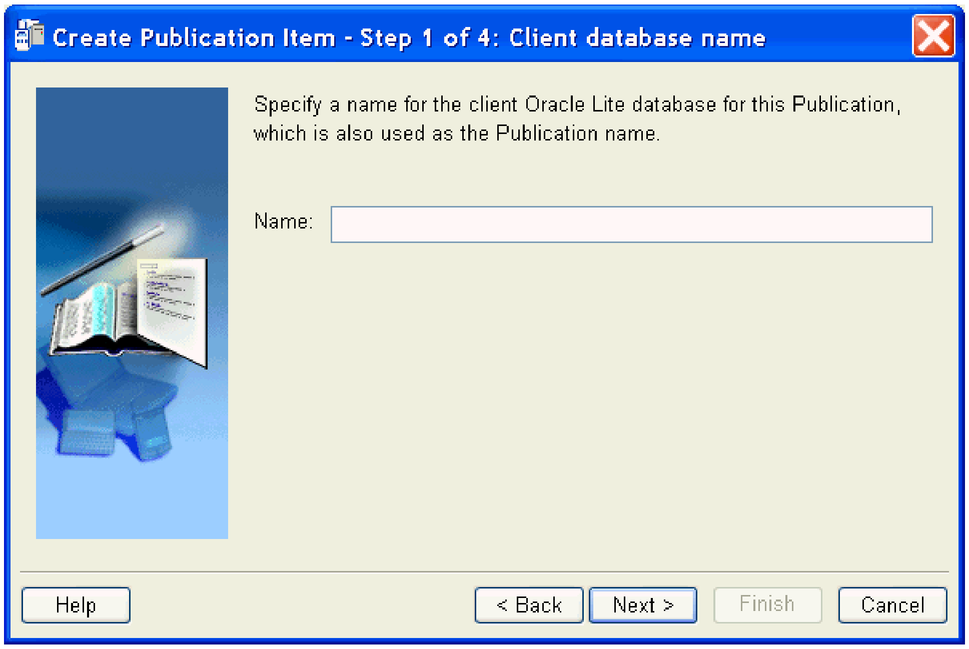
The Quick Start Wizard creates Publication Items where the snapshot definition is based upon all columns in a specified table in the application schema.

☐ Skip this Page Next Time

3. Provide a name for the client Oracle Lite database. This is the database that exists on the device to contain the downloaded snapshot information. The name that you choose will also be used as the name of the publication.

   When this publication is finished, a client database is created on your device and the first synchronization to retrieve the snapshot from the back-end Oracle database is initiated.

Figure 5–2 Define Client Oracle Lite Database Name



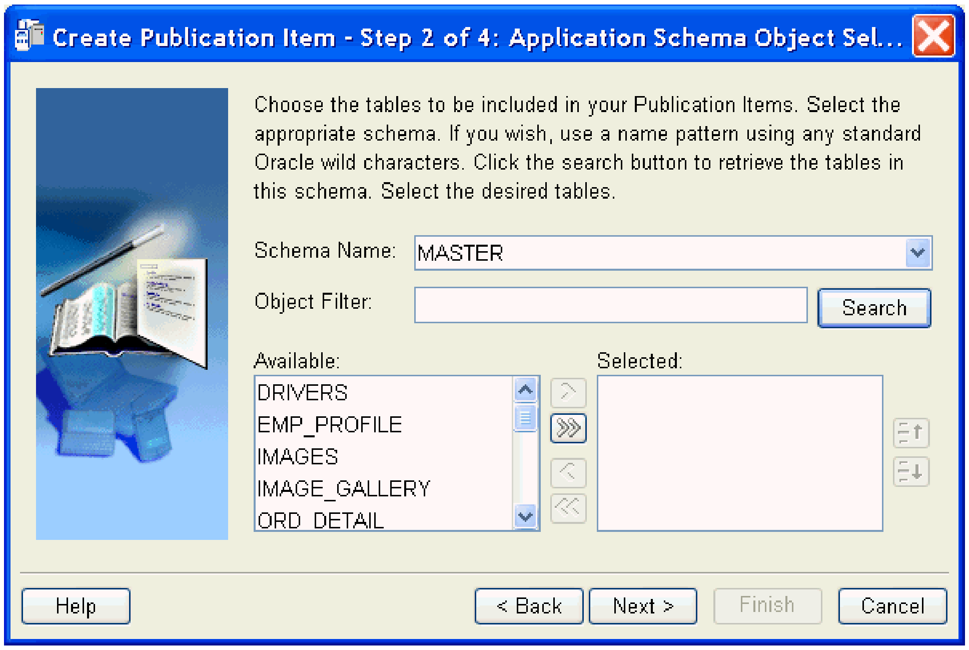
Create Publication Item - Step 1 of 4: Client database name

Specify a name for the client Oracle Lite database for this Publication, which is also used as the Publication name.

Name:

4. Select the table(s) to be included in the publication item, as follows:

Figure 5–3 Define the Tables to Include in the Publication



Create Publication Item - Step 2 of 4: Application Schema Object Sel...

Choose the tables to be included in your Publication Items. Select the appropriate schema. If you wish, use a name pattern using any standard Oracle wild characters. Click the search button to retrieve the tables in this schema. Select the desired tables.

Schema Name: MASTER

Object Filter:

Available: DRIVERS, EMP_PROFILE, IMAGES, IMAGE_GALLERY, ORD_DETAIL

Selected:

- Choose the application schema to associate with this publication item: The application schema is the schema from which the publication item retrieves data. All available schemas in the database are listed in the pull-down list. You must have created the schema prior to starting this wizard.

Note: If the schema you want is not in this list, cancel the wizard, create the schema in the back-end database, and then re-start this wizard.

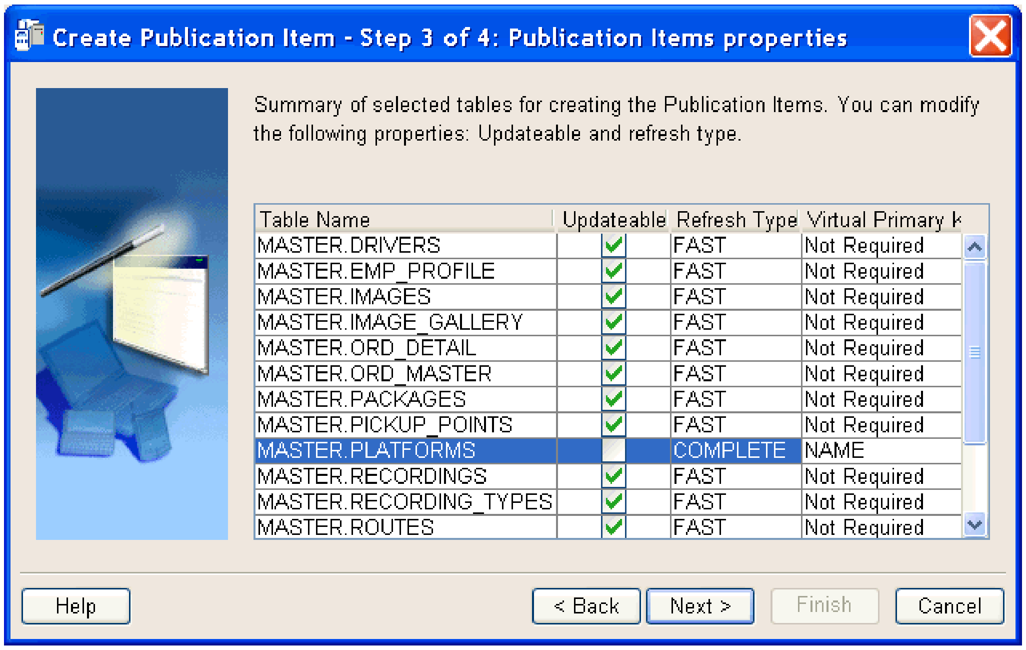
- Click **Search** to display all tables within this schema in the Available column. To search for a specific table or tables, enter the name or partial name with wild charaters in the Object Filter field and then click **Search**. You can use any of the standard Oracle wild card characters.
- Select the table(s) that you want in the publication item and click the arrow buttons to move one or all tables into the Selected column. You can move these tables back and forth using the arrow buttons.

Note: If you do not see the object that you expect to see, verify that you created the table in this schema in the back-end Oracle database.

- When you are satisfied with the list of the tables in this publication item, then click **Next**.

5. Once the creation of the publication item is completed, a Summary page displays the defaults used for each table included in this publication item, as follows:

Figure 5–4 Modify the Table Properties for Synchronization



Create Publication Item - Step 3 of 4: Publication Items properties

Summary of selected tables for creating the Publication Items. You can modify the following properties: Updateable and refresh type.

Table Name	Updateable	Refresh Type	Virtual Primary k
MASTER.DRIVERS	✓	FAST	Not Required
MASTER.EMP_PROFILE	✓	FAST	Not Required
MASTER.IMAGES	✓	FAST	Not Required
MASTER.IMAGE_GALLERY	✓	FAST	Not Required
MASTER.ORD_DETAIL	✓	FAST	Not Required
MASTER.ORD_MASTER	✓	FAST	Not Required
MASTER.PACKAGES	✓	FAST	Not Required
MASTER.PICKUP_POINTS	✓	FAST	Not Required
MASTER.PLATFORMS		COMPLETE	NAME
MASTER.RECORDINGS	✓	FAST	Not Required
MASTER.RECORDING_TYPES	✓	FAST	Not Required
MASTER.ROUTES	✓	FAST	Not Required

- **Table name**: Displays the schema and name of the table included in this publication item.
- **Updatable**: This is checked if the table is listed as updatable. You can toggle this item to read-only by double-clicking on the field. However, if it is unchecked, you should only enable it if the table has a virtual primary key.

  For more information on Read-Only or Updatable options, see Section 3.3.1.1, "Manage Snapshots".

- **Refresh Type**: By default, all tables use fast refresh. If the table does not have a primary key, then the table uses complete refresh. Double-click on this field to change the refresh type.

  For more information on Fast or Complete refresh types, see Section 3.9, "Understanding Your Refresh Options".

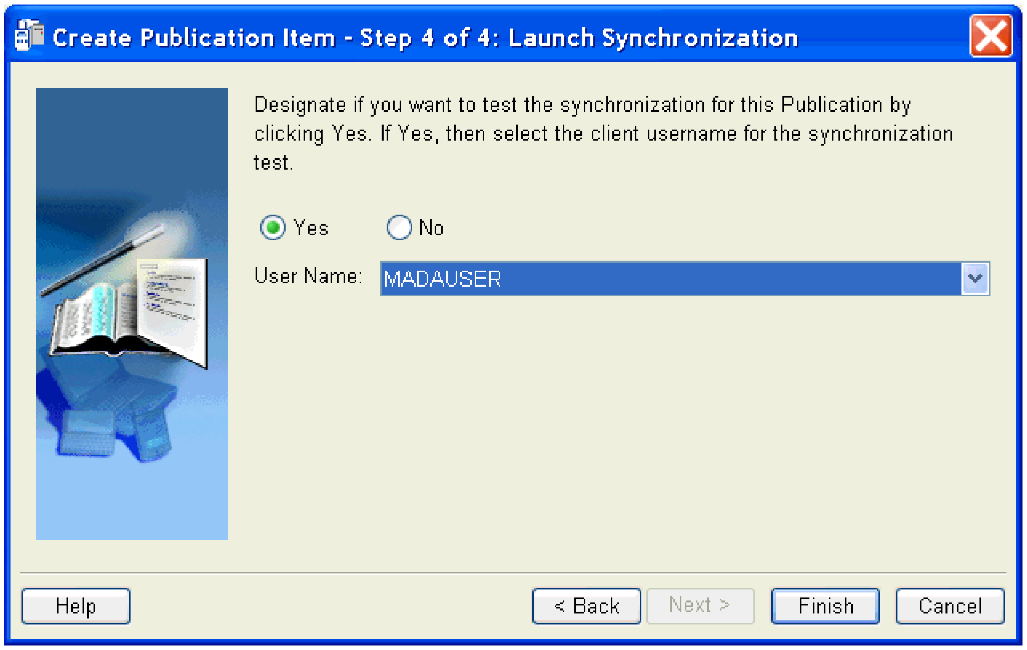
- **Virtual Primary Key**: This field displays the virtual primary key for the table. If you want to have the table be updatable or use the fast refresh type, then the table must have a virtual primary key. If the table does not have a primary key, but it does contain a field with UNIQUE constraints, then you can specify this field as the virtual primary key to be able to use fast refresh or updatable.

  Note: Any virtual primary key added must be unique and not null.

  To specify a column in the table as your virtual primary key, double-click on the Virtual Primary Key field to list all of the UNIQUE fields. If you select one of them to be the virtual primary key, then you can use the Updatable or fast refresh options for this table.

6. Decide if you want to test this publication.

Figure 5–5 Decide to Test the Publication


Create Publication Item - Step 4 of 4: Launch Synchronization

Designate if you want to test the synchronization for this Publication by clicking Yes. If Yes, then select the client username for the synchronization test.

[✓ Yes] No

User Name: MADAUSER

You can specify that you want to test this publication as soon as the wizard exits. By default, **Yes** is selected. This provides a test of the publication against the back-end Oracle database.

In order to perform this test, a valid client username must be provided. From the drop-down list, select the client username that you would like to use. You will be prompted for the password during synchronization.

7. You can end the wizard by performing one of the following:

   - Click **Back** to modify any of the information supplied.

- Click **Finish** to complete the project creation.
- Click **Cancel** to abort creation of this project.

8. If you clicked **Yes** for testing the publication, then the Test Publication screen is brought up. Click the **Synchronize** button to start the test.

This creates a basic publication, which you can now view in the project in the MDW main screen. You can modify this publication in any way.

5.4 Create a Publication Item

The Publication Item Wizard steps you through the process of creating a publication item in the project. A publication item encapsulates a snapshot definition. It can be based on a table, view or synonym in the Master Application schema in the back-end database. If you use a synonym for a remote object to build a publication item, then you are required to provide the JDBC connection information to the remote database where the remote object resides.

After you create the publication items in the project, then you can associate multiple publication items with a publication, which is then associated with an application. See Section 5.9.2, "Publication Item Tab Associates Publication Items With the Publication" for details.

You can create a publication item through the publication item wizard by clicking **File->New->Publication Item**.

1. The publication item wizard introduction appears. If you do not want this introductory screen to display each time you start a new project, check the "Skip This Page Next Time" box.

   Click **Next** to advance to the next screen.

2. Define name, refresh type, and automatic synchronization, as follows:

   - Publication item name: This name can be any valid Java identifier. The name cannot contain any spaces. Publication item names are limited to twenty-six characters and must be unique across all publications. For example, your publication item name may be something like `MY_PUBLICATION_ITEM`.
   - Refresh type: The refresh mode of the publication item is specified during creation to be either fast or complete refresh. See the Section 3.9, "Understanding Your Refresh Options" for more information.

     From the drop-down list choose one of the following refresh types:

     – Complete: All data is refreshed with current data. Everytime you synchronize, all data in the snapshot is retrieved. This can be performance intensive.

     – Fast: This is the recommended mode. Only incremental changes are synchronized. Thus, you are not downloading the complete data snapshot each time a synchronization is requested. The advantages of fast refresh are reduced overhead and increased speed when replicating data stores with large amounts of data where there are limited changes between synchronization sessions.

     – Queue-based: You can create your own queue. Mobile Server will upload and download changes from the user. You perform the activity of the MGP and apply/compose the modifications to the back-end database. See the Section 3.19, "Customizing Replication With Your Own Queues" for more information.

Once you create the publication item with a particular refresh type, the only way to modify the publication item to have a different refresh type is to delete is and recreate it with the desired refresh type.

- Enable Automatic Synchronization checkbox: This defines the publication item to use Automatic Synchronization, where synchronization for this publication item occurs automatically and in the background. That is, you do not have to manually press the Sync button as it occurs automatically. You can have several publication items in a single publication where some use automatic synchronization and others require the user to manually request synchronization.

  If you have multiple publication items within a publication, you can specify which are to be automatically synchronized within each publication item.

  Step 7 shows you how to specify—with a SQL statement—which users receive the automatic synchronization. Section 5.5, "Define the Rules Under Which the Automatic Synchronization Starts" shows you how to define automatic synchronization rules that will apply to this publication item.

3. Designate the publication item object with the appropriate schema, as follows:

   - Choose the application schema to associate with this publication item: The application schema is the schema from which the publication item retrieves data. All available schemas in the database are listed in the pull-down list. You must have created the schema prior to creating the publication item.

     If the schema you want is not in this list, cancel this publication item, create the schema in the back-end database, and then associate the schema with the publication item.

   - Designate the object type as a table, view, or synonym.
   - To choose the table, view or synonym from within the schema that you wish to base this publication item on, click **Search** on the Object Filter. This brings up several items in the Object List. Select the object that you are interested in and click **Next**.

   Note: To search only for objects that match a condition, designate the condition in the Filter box and click **Search**. You can use the same pattern matching characters in a valid `SQL WHERE` clause. The filter is case-sensitive; use upper-case characters.

4. Choose columns to add to the publication item. When you are adding columns to the publication item, you should first verify what data types are supported and how others are modified when brought down to the Oracle Lite database. For example, the `TIMESTAMP` data type is supported, but the `TIMESTAMP WITH TIME ZONE` data type is not. For details, see Section 3.23, "Datatype Conversion Between the Oracle Server and Client Oracle Lite Database".

   There are two tabs to enable you to structure your publication item, as follows:

   - Column Selection: Choose the columns from within the object that you will use to retrieve information for the application. To choose the appropriate columns, select the column name and click the left or right arrow buttons to move between the Available and Selected windows. To move all columns, use the double arrows.

Note: The primary key defaults to being in the Selected window, as it is required if you are using Fast Refresh and Updateable option. Since most publication items use these options, MDW places the primary key as Selected. You can move it back to the Available window.

- Structure: If you are not sure what columns you want, you can see the entire table structure by clicking this tab.

Note: Oracle Database Lite does not support creating publication items for a table with object type columns, even if the publication item query does not include any of the object type columns. However, it is possible to define a view which selects only columns of supported data types and then create a publication item using the view definition.

If you have specified a fast refresh, you must provide a primary key. If you have specified a table or view that does not have a primary key, exit out of this wizard and create a virtual primary key specifying one of the columns in the table or view. If you do not create a virtual primary key before specifying this publication item, then any future synchronization of this primary key will fail.

Note: Any virtual primary key must be unique and not null.

---

5. Modify the SQL statement for the publication item. From the columns that you selected in the previous screen, this simple SQL statement is available as a template for you to modify. You can add any qualifiers and complexity to this base statement. To view the structure of the schema object, select the Structure tab.

   - Perform Iterative Modifications

     See Section 5.4.1, "Create SQL Statement for Publication Item" for directions on how to edit and execute the SQL statement for this publication item.

   - Apply/Compose Callbacks

     When creating publication items, the user can specify a customizable package to be called during the Apply and Compose phase of the MGP background process. Client data is accumulated in the in queue prior to being processed by the MGP. Once processed by the MGP, data is accumulated in the out queue before being pulled to the client by Mobile Sync. See Section 3.7.2, "Customize What Occurs Before and After Compose/Apply Phases for a Single Publication Item" for more information on how to create the callbacks.

     Provide the schema and package names for any apply/compose callbacks.

   - Dependency Hint

     Click **Add** to add a dependency hint. All existing dependency hints are shown in the window. See Section 5.4.2, "Create a Dependency Hint" for more information.

6. If you specified a view, then you may need to define parent table and primary key hints. See Section 5.4.3, "Specify Parent Table and Primary Key Hints" for directions on how to define these hints.

7. If you specified automatic synchronization for this publication item, then the Automatic Synchronization Query page is shown, which includes the following:

   - By default, all users are included in the Compose, which means that all users receive the data that is being retrieved from the server and brought down to the Mobile clients. If you want to specify that only certain users that subscribe to this application receive the application data, then check this box.
   - If you clicked the checkbox, then specify which users are to receive the Compose data with a SQL query.

     For example, if you only want `MADAUSER` within the `mobileadmin` schema to receive the Compose data, then type in `select name from mobileadmin.users where name ='MADAUSER'`. Click **Run** to verify that the SQL query returns what you want; click **Next** to advance to the next page.

8. Summary page provides an overview of the publication item that you just created. To view any dependency hints, click **Advanced**. If you have used a view as the base for your publication item and you created Parent Table or Primary Key hints, click Hint to view these hints. The Parent Table and Primary Key hints are only valid on views.

   To modify any part of the publication item, click **Back** to return to the previous screens.

   Click **Finish** if you are satisfied with the publication item. Click **Cancel** to eliminate all work in creating this publication item.

5.4.1 Create SQL Statement for Publication Item

You can compose your SQL statement through iterative steps to ensure that you are creating a valid statement. You have the following options:

- Modify the query by clicking **Edit**.
- Execute the statement against the schema in the back-end database by clicking **Run**. You will be notified directly if the statement fails or view the data from the schema object is retrieved for you to view.

  When you click **Run**, then the query of the publication item is validated by executing the query against the back-end database. A dialog is displayed with the returned snapshot that this publication item would generate. If there is more than one page, then click **Previous** and **Next** to move between the pages. Click **Close** to return to the publication item screen.

  If the publication item SQL statement requireds an input value, then a dialog appears for you to input the value for the SQL query. You can modify this value or the SQL query until you receive the results that you desire.
- To return the query to the original statement, click **Reset**.
- When finished, click **Next**.

5.4.2 Create a Dependency Hint

If the updates to this publication item effects another table, use the dependency hint to notify the synchronization engine to update the other table. For example, if the publication item updates the employee table, and these updates should also apply to the department table, add a dependency hint notifying the synchronization engine of the relationship with the department table.

For your dependency hint, specify whether the hint is based upon a table or synonym. Then, use the pulldown lists to select the schema and table/synonym names. Click **OK** to save the hint or **Cancel** to return to the Advanced screen.

For more information on dependency hints, see Section 3.17.2, "Creating or Removing a Dependency Hint".

5.4.3 Specify Parent Table and Primary Key Hints

When you use a view, you may need to specify a parent table and primary key hints. A view can be composed of one or more tables joined together. If you have specified fast refresh, then you must specify which table is the parent table and which column is the primary key.

- To create a parent table hint, select the base table from the Base Table(s) drop-down list and check the Parent Table Hint checkbox.

  Note: If you do not check the Parent Table Hint checkbox, then the hint is not created when you click **Next**.

- To create a primary key hint, click **Add**. Identify the primary key hint for this view, as follows:

  1. From the View Columns drop-down list, select the view column that you want to be the primary key. This column name may be an alias of the actual `table.column` name.
  2. From the Base Tables drop-down list, select the base table where the actual column exists that is to be the primary key.
  3. From the Primary Key Columns drop-down list, all primary key columns from the base table are shown, select the appropriate column for the primary key. If you have a composite primary key, iteratively add each column within the composite primary key.

  Note: If you do not provide accurate details in regards to the view, base table, and primary key to which the view column maps, the publication item may save, but the execution of the publication item will fail. For example, if you choose a view column which does not map to the primary key column, MDW will allow the action, but any execution of the publication item will result in failure.

  Click **OK** to accept this primary key hint.

5.5 Define the Rules Under Which the Automatic Synchronization Starts

Once you have enabled a publication item to use automatic synchronization, you must define the rules under which the automatic synchronization executes. The circumstances under which an automatic synchronization occurs is defined within the synchronization rules. There are two types of automatic synchronization rules: events and conditions. If an event is true, it starts a synchronization; however, the synchronization cannot occur unless all conditions are true, as well. This evaluates as follows:

```
when EVENT and if (CONDITIONS) then sync;
```

If an event is true, then a synchronization can start—but only if all conditions are true.

Thus, event and condition rules are as follows:

- Events—An event is variable, as follows:
  - Data events: For example, you can specify that a synchronization occurs when there are a certain number of modified records in the client database.
  - System events: For example, you can specify that if the battery drops below a predefined minimum, you want to synchronize before the battery is depleted.
- Conditions—A condition is an aspect of the client that needs to be present for a synchronization to occur. This includes conditions such as battery life or network availability.

For example, if the event for new data inserted and the condition specified is that the network must be available, then a synchronization only occurs when the network is available and there is new data.

When you define the rules for the synchronization, you can define them in two places:

- Publication level: You specify the rules under which the synchronization occurs at the publication level for all publication items in that publication.
- Platform level: Some of the rules are very specific to the platform of the client, such as battery life, network bandwidth, and so on.

Note: This section describes how to do this through MDW; see Section 3.2.3, "Define the Rules Under Which the Automatic Synchronization Starts" for directions on how to perform this programmatically.

The following sections detail all of the rules you can configure for automatic synchronization:

- Section 5.5.1, "Configure Publication-Level Automatic Synchronization Rules"
- Section 5.5.2, "Configure Platform-Level Automatic Synchronization Rules"

5.5.1 Configure Publication-Level Automatic Synchronization Rules

When you are creating the publication, you can define data events that will cause an automatic synchronization. Although these are defined at the publication level, they apply only to the publication items within this publication that have automatic synchronization enabled.

For full details of how to configure these data events, see Section 5.9.6, "Event Tab Configures Automatic Synchronization Rules for this Publication".

Table 5–1 describes the publication level data events that trigger automatic synchronization. The lowest value you can specify is 1.

Table 5–1 Automatic Events for the Publication

Events	Description
Client commit	Upon commit to the Oracle Lite database, the Mobile client detects the total number of record changes in the automatic synchronization log. If the number of modifications is equal to or greater than your pre-defined number, automatic synchronization occurs.

Table 5–1 (Cont.) Automatic Events for the Publication

Events	Description
Server MGP compose	If after the MGP compose cycle, the number of modified records for a user is equal to or greater than your pre-defined number, then an automatic synchronization occurs. Thus, if there are a certain number of records contained in an Out Queue destined for a client on the server, these modifications are synchronized to the client.

Note: If you want to modify the publication-level automatic synchronization rules after you publish the appliation, you can do so through the Mobile Manager, as follows:

1. Click **Data Synchronization**.
2. Click **Repository**.
3. Click **Publications**.
4. Select the publication and click **Automatic Synchronization Rules**.

5.5.2 Configure Platform-Level Automatic Synchronization Rules

The platform-level synchronization rules apply to a selected client platform and all publications that exist on that platform. You can specify both platform events and conditions using either MDW or the Mobile Manager. This section describes MDW; see Section 5.4.1, "Specifying Platform Rules for Automatic Synchronization" in the *Oracle Database Lite Administration and Deployment Guide* for directions on how to define these rules using Mobile Manager.

To assign platform-level automatic synchronization rules, perform the following in MDW:

1. Click **Platform**.
2. Select either the Win32 or WINCE platform, which brings up a page with two tabs: Events and Conditions. These rules will apply to all publications for this platform.
3. You can modify the following for each platform:
   - Event Rules—See Section 5.5.2.1, "Define System Event Rules for the Platform".
   - Conditions—See Section 5.5.2.2, "Define Automatic Synchronization Conditions for the Platform".

Note: You can only modify the network settings for the platform using Mobile Manager, see Section 5.4.1, "Specifying Platform Rules for Automatic Synchronization" in the *Oracle Database Lite Administration and Deployment Guide* for more information.

5.5.2.1 Define System Event Rules for the Platform

When you choose the Event tab, select the checkbox for each event that you want to enable. If the event requires a value, enter the value you desire. This initiates the automatic synchronization the first time the event occurs. For example, if the battery runs below the percentage you specified, the automatic synchronization occurs. As the battery continues to deplete, you will not trigger another synchronization.

The following system events will trigger an automatic synchronization if true.

- Network Bandwidth: Synchronize when the network bandwidth is greater than `<number>` bits/second. Where `<number>` is an integer that indicates the bandwidth bits/seconds. When the bandwidth is at this value, the synchronization occurs.
- Battery Life: Synchronize when the battery level drops to `<number>`%, where `<number>` is a percentage. Often you may wish to synchronize before you lose battery power. Set this to the percentage of battery left, when you want the synchronization to automatically occur.
- AC Power: Synchronize when the AC power is detected. Select this checkbox if you want the synchronization to occur when the device is plugged in.

5.5.2.2 Define Automatic Synchronization Conditions for the Platform

When you choose the Condition tab, you can set under what conditions the automatic synchronization is allowed or disallowed, as follows:

- Battery Level: Specify the minimum battery level required in order for an automatic synchronization to start. The battery level is specified as a percentage.
- Network Availability: Network quality can be specified using several properties. For example, if you have a very low network bandwidth and a high ping delay, you may only want to synchronize your high priority data. To add network quality condition for a specified data priority, click the **Add** button, which brings up a screen where you can specify a minimum value for the following network properties:
  - Data Priority: You could have defined records in the snapshot with a data priority number. Use this condition to specify under what conditions the different data priority records are synchronized. Data priority can be either one or zero, where zero is high priority. By default, all records are entered with a value of NULL, which is the lowest priority.
  - Minimum Network Bandwidth (bits/sec): Configure the minimum bandwidth (bits/second) in which the automatic synchronization can occur for records with this data priority.
  - Maximum Ping Delay (ms): Configure the maximum ping delay (milliseconds) in which the automatic synchronization can occur for records with this data priority.
  - Include Dial-up Networks?: The always-on network is used if available. However, if this network is not available, select **YES** if you want to use any of the dial-up networks for this data priority.

5.6 Create a Sequence

A sequence is a database schema object that generates sequential numbers for new records into a table. After creating a sequence, you can use it to generate unique sequence numbers for transaction processing. These unique integers can include primary key values. If a transaction generates a sequence number, the sequence is incremented immediately whether you commit or roll back the transaction.

Create a sequence by clicking **File->New->Sequence**.

However, when you have Oracle Database Lite, you must consider how you allow the generation of sequence numbers. If you want the client only to use the sequence, you have a native sequence, which does not involve the server-side. However, if you want the server and client to share a sequence, you have to specify a server-side sequence and allow for both the client and server to be using numbers within the same range.

For example, if you have a single back-end database and a Mobile client, where both are allowed to add records to the application tables, then you have to check the checkbox to generate the server-side sequence and allow at least two as the increment value, since the client will use one and the server will use the other.

If you have more than a single client, you want to assign who gets which sequence numbers, so that when you synchronize, none of the records have duplicate sequence numbers. Thus, if you have multiple clients, then specify a specific range of numbers for each client, so that they are not using the same numbers.

- Specify a range of values for each client. In our example, client A would be assigned sequence numbers 1 through 100, client B would be assigned sequence numbers 101 to 200, and client C would be assigned sequence numbers 201 through 300. If they ran out of sequence numbers, they are assigned another 100, which is the defined window size in our example, during the next synchronization. Since none of the clients checked to generate server-side sequence, the database, in order to never collide with the sequence numbers, starts its sequence number at -1.
- You could specify that all clients are allowed to have only odd numbers and the database has all even numbers. That is, you could start the client at 1 and increment by 2 for all of its sequence numbers. This enables you to avoid having negative numbers for your sequence numbers. The clients still have a window size, which in this example is 100, but they start with an odd number within that window and always increment by 2 to avoid any positive numbers. Thus, client A would still have the window of 1 to 100, but the sequence numbers would be 1, 3, 5, and so on up to 99.

To create a sequence, provide the following information:

- Name: This name must be a valid Java identifier.
- Starts With: Enter the number with which you want this sequence to start.
- Increment: Specify the increment from the starting value for the next value in the sequence.

Note: If you have checked the **Generate server-side sequence** checkbox and set the increment value to 1, then this value is ignored and is set to 2. When you specify the server-side sequence, then both the client and the server use every other number in the sequence. Thus, you cannot increment by 1 on the client.

- Window Size: The range of numbers given to the client when the threshold is reached.
- Threshold: Defining the amount of sequence numbers necessary to be assigned in order for operations to continue smoothly.
- Description: A description of the sequence.
- Generate server-side sequence: If you want the client and the server-sides to share a sequence, where one side has all even numbers and the other has the odd numbers, check this box. If unchecked, then the sequence is created solely for the client.

Once you create a sequence in the project, you can associate it with a publication. See Section 5.9.3, "Sequence Tab Associates Existing Sequences With the Publication" for details.

See the Section 3.4.1.8, "Creating Client-Side Sequences for the Downloaded Snapshot" for more information on sequences.

5.7 Create and Load a Script Into The Project

You can add a script to this project. Create the script on your file system and then upload it to MDW. Before you add the script to the project, you can use MDW to test the script. See the following sections for more information:

- Section 5.7.1, "Writing SQL Scripts"
- Section 5.7.2, "Test SQL Scripts"
- Section 5.7.3, "Load the Script Into the Project"

5.7.1 Writing SQL Scripts

When you write and upload a SQL script to the project, each script is executed independently by Oracle Database Lite in no specified order. Therefore, if you have dependencies and need the DDL statements to be executed in a certain order, include all statements in the correct order in a single script, where each DDL statement is separated by a semicolon (" ; ").

Alternatively, you can specify the weight for the script when loading them to specify the order in which each script is executed on the client. See Section 5.7.3, "Load the Script Into the Project" for more details.

If a SQL script fails upon execution, Oracle Database Lite will execute it once more, in case the failure was due to a dependency of a later script. However, if you have a script with a dependency on another script, you could effect your performance while Oracle Database Lite re-executes all of the scripts to resolve dependencies.

Note: If you upload scripts using one of the Consolidator APIs, you must also ensure that the order of execution for these scripts does not matter. Include all dependent DDL statements in a single script and in the order necessary for clean resolution.

5.7.2 Test SQL Scripts

You can test a SQL script that resides on your file system by selecting **Tool->SQL Window**. Through the SQL Wizard, perform the following:

1. Connect to the correct database—whether it is an Oracle database or a device Oracle Lite database.
2. Load the SQL script from your file system.
3. Execute the script. You can execute the current script or re-execute scripts that are on the history page. You can choose to have the results displayed on the screen or spooled to a file.

The following sections describe how to accomplish these tasks:

- Section 5.7.2.1, "Connect to the Database"
- Section 5.7.2.2, "Load and Execute SQL Scripts"

5.7.2.1 Connect to the Database

1. Select type of database—Select the radio button next to the type of database to which you are connecting—an Oracle database or the client Oracle Lite database. If you selected the client Oracle Lite database, you must also specify the Mobile client platform and protocol with the drop-down lists. Only currently installed platforms and protocols are displayed in these drop-down lists.
2. Specify database authentication and destination connection information—Part of the information necessary for completing the database connection is the authentication user name and password and the database destination information, which can include either the DSN of the Mobile client Oracle Lite database or the host, port, and SID for the Oracle database.
3. Review database connection values—The final screen displays a summary of all of the configured values for the database connection. Verify that the information is correct and then click **Finish**. You can return to any page to modify the information by clicking **Back**.

5.7.2.2 Load and Execute SQL Scripts

You can test any SQL scripts against the database defined in the previous portion of this wizard.

- Click **Load Script** to browse your file system for the script that you want to test. Define if you want this script to be spooled or auto-committed.
- Click **Execute** to run the script on the destination database. The results show up in the bottom results screen.

  If you want, you can spool the results to a file by checking the spool checkbox before you click **Execute**. When you check Spool, a dialog appears for you to define the location and name of the file to receive the output from the script. Check the Overwrite checkbox if you want this file overwritten each time that the script is executed.

  Check the Autocommit checkbox if you want the SQL committed automatically after the script completes.

The Results and History tabs show the current results and all past results respectively. Clear the Results screen by either clicking **Clear Results** or by checking the Auto Clear checkbox. Clear the historical information by clicking the **Clear History** button on the History page.

Note: Any SQL on the History page can be executed by selecting the corresponding row and clicking **Execute**.

5.7.3 Load the Script Into the Project

Define the script on your machine. Once defined, perform the following:

1. Bring the script into the project by clicking **File->New->Script**.
2. Provide a user-defined name to identify the script and browse for the script in your file system.
3. Specify the weight, if necessary. You can specify the weight for the script when loading them to specify the order in which each script is executed on the client. For example, when creating a master detail table on the client, you must create first the master table and then the detail table. The client does not know which script

should be executed first, unless you specify a weight to let the client know the order in which to execute the scripts.

4. Click **OK** to accept the definition and **Cancel** to return to the previous screen.

Once you include a script in the project, you can associate it with a publication. See Section 5.9.4, "Script Tab Associates Existing Scripts With the Publication" for more information.

5.8 Load a Resource Into the Project

You can load a JAR file that contains Java class files as a resource in a project. Once loaded as a resource, the JAR file is downloaded to the client on the first synchronization. In addition, if this resource is modified, it will be sent down on the next synchronization.

Note: You can only load JAR files as resources, not individual class files. Once you load a JAR file, the only way you can replace it is by dropping the JAR and then loading the new JAR file.

To specify an existing resource, click **File->New->Resource**.

Provide the JAR file on your machine. Specify a user-defined name to identify the resource and browse for the JAR file in your file system. Click **OK** to accept the definition and **Cancel** to return to the previous screen.

Once you include a resource in the project, you can associate it with a publication. See Section 5.9.5, "Resource Tab Associates Existing Resources With the Publication" for more information.

5.9 Create a Publication

Create a publication by clicking **File->New->Publication**. You can create the publication at any time. This starts the dialog for creating a publication.

There are six tabs included for configuring information about the new publication. On configures general information about the publication, one defines event rules for automatic synchronization, and the others enable you to associate different objects with the publication.

If you click **OK**, then you can associate the objects by selecting the publication name and then selecting the appropriate tab.

When you are finished creating the publication, click **File->Save** to save the publication.

- Section 5.9.1, "General Tab Configures Publication Name"
- Section 5.9.2, "Publication Item Tab Associates Publication Items With the Publication"
- Section 5.9.3, "Sequence Tab Associates Existing Sequences With the Publication"
- Section 5.9.4, "Script Tab Associates Existing Scripts With the Publication"
- Section 5.9.5, "Resource Tab Associates Existing Resources With the Publication"
- Section 5.9.6, "Event Tab Configures Automatic Synchronization Rules for this Publication"

5.9.1 General Tab Configures Publication Name

The General tab provides the following information about your new publication within your project:

- Publication name: Enter a valid Java identifier for the publication name. The name cannot contain any spaces or special characters.
- Optional description: You can add a description to remind you of the content of this publication.
- Client database name: This defaults to the same name as the publication name. However, you can modify it. The purpose of this name is to specify the name of the client Mobile database, which is created during the first synchronization.

5.9.2 Publication Item Tab Associates Publication Items With the Publication

Selecting the Publication Item tab from within the publication enables you to associate any existing publication item to this publication.

Manage Publication Items In This Publication

- To add an existing publication item to this publication, Click **Add**.
- To remove a publication item from this publication, select the desire publication item from the list and click **Remove**.
- To edit the details of the association for the publication item, select the desired publication item and click **Edit**.

To accept the current changes, click **OK**.

5.9.2.1 Associating a Publication Item to this Publication

To associate any publication item to this publication, the publication item must first exist. Thus, all of the information requested on this screen is about existing publication items.

Provide the following information to identify the publication item to associate to this publication:

Identify Existing Publication Item

From the Name drop-down list, select the name of the publication item.

Updatable or Read-Only Snapshot

Select if the snapshot is updatable or read-only. See Section 3.3.1.1, "Manage Snapshots" for more details.

- Read-only snapshots are used for querying purposes. Changes made to the master table are replicated to the snapshot by the Mobile client.
- Updatable snapshots provide updatable copies of a master table. You can define updatable snapshots to contain a full copy of a master table or a subset of rows in the master table that satisfy a value-based selection criteria. You can make changes to the snapshot which the Mobile Sync propagates back to the master table.

  A snapshot can only be updated when all the base tables that the snapshot is based on have a primary key. If the base tables do not have a primary key, a snapshot cannot be updated and becomes read-only.

Conflict Resolution

When adding a publication item to a publication, the user can specify winning rules to resolve synchronization conflicts in favor of either the client or the server. A Mobile Server synchronization conflict is detected under any of the following situations:

- The same row was updated on the client and on the server.
- Both the client and server created rows with equal primary keys.
- The client deleted a row and the server updated the same row.
- The client updated a row and the server deleted the same row. This is considered a synchronization error for compatibility with Oracle database advanced replication.
- For systems with delayed data processing, where a client's data is not directly applied to the base table (for instance in a three tier architecture) a situation could occur when first a client inserts a row and then updates the same row, while the row has not yet been inserted into the base table. In that case, if the `DEF_APPLY` parameter in `C$ALL_CONFIG` is set to `TRUE`, an `INSERT` operation is performed, instead of the `UPDATE`. It is up to the application developer to resolve the resulting primary key conflict. If, however, `DEF_APPLY` is not set, a "`NO DATA FOUND`" exception is thrown (see below for the synchronization error handling).
- All the other errors including nullity violations and foreign key constraint violations are synchronization errors.
- If synchronization errors are not automatically resolved, the corresponding transactions are rolled back and the transaction operations are moved into Mobile Server error queue in C$EQ, while the data is stored in CEQ$. Mobile Server database administrators can change these transaction operations and re-execute or purge transactions from the error queue.

Choose the type of conflict resolution you want for this publication item, as follows:

- Client wins—When the client wins, the Mobile Server automatically applies client changes to the server. And if you have a record that is set for `INSERT`, yet a record already exists, the Mobile Server automatically modifies it to be an `UPDATE`.
- Server wins—If the server wins, the client updates are not applied to the application tables. Instead, the Mobile Server automatically composes changes for the client. The client updates are placed into the error queue, just in case you still want these changes to be applied to the server—even though the winning rules state that the server wins.
- Custom—You have created your own callbacks to resolve the conflict resolution.

All synchronization errors are placed into the error queue. For each publication item created, a separate and corresponding error queue is created. The purpose of this queue is to store transactions that fail due to unresolved conflicts. The administrator can attempt to resolve the conflicts, either by modifying the error queue data or that of the server, and then attempt to re-apply the transaction.

See Section 3.13, "Resolving Conflict Resolution with Winning Rules" for more information.

DML Callback

A user can use Java to specify a customized PL/SQL procedure which is stored in the Mobile Server repository to be called in place of all DML operations for this publication item. There can be only one mobile DML procedure for each publication item. See Section 3.4.1.13, "Callback Customization for DML Operations" for more information on how to specify a DML Callback.

Enter a string for the schema and package of the DML callback, such as `schema.package_name`.

Grouping Function

If you know that two tables should share a map, but Oracle Database Lite would not normally associate these tables, provide a grouping function that denotes the shared publication item data between the tables.

Note: The Mobile Server schema owner needs to be granted execute privilege on the defined grouping function.

The grouping function is a PL/SQL function with the following signature.

```
(
CLIENT in VARCHAR2,
PUBLICATION in VARCHAR2,
ITEM in VARCHAR2
) return VARCHAR2.
```

The returned value must uniquely identify the client's group.

In this field, provide the PL/SQL grouping function fully-qualified, either with `schema.package.function_name` or `schema.function_name`.

See the Section 1.2.4 "Shared Maps" in the *Oracle Database Lite Troubleshooting and Tuning Guide* for more information.

Priority Condition

Provide a string that is to be added to the publication item query statement to limit what is returned based on priority. For example,

For example, if you have a snapshot with the following statement:

```
select * from projects where prio_level in (1,2,3,4)
```

The projects table has a column named `prio_level`, where the values can be 1 to 4. If you wanted to limit what data was returned, you could limit the statement specific to this publication item in this publication by adding a string that would be added to the SQL statement preceded by an AND.

For example, to restrict the snapshot from the projects table, you could define the priority condition string to be `prio_level = 1`. This would generate the following statement:

```
SELECT * FROM projects where prio_level in (1,2,3,4) AND prio_level = 1;
```

In this case, only projects with level =1 are replicated to the client. You can make this statement to be anything you wish to restrict what is returned to the client.

See Section 1.2.5 "Priority-Based Replication" in the *Oracle Database Lite Troubleshooting and Tuning Guide* for more information.

MyCompose Class

Provide a string with the full path and classname of the location and name of the `MyCompose` Class. See Section 3.6, "Customize the Compose Phase Using MyCompose" for more information on this class.

Weight

You can rate the order in which each publication item in this publication is executed by specifying the weight. This should be a number. Each publication item must have a unique number in ascending order. The first publication item executed is the one with the weight of one.

5.9.3 Sequence Tab Associates Existing Sequences With the Publication

You can only associate an existing sequence with the publication on this screen. To add an existing sequence, click **Add**.

> **Note:** You can create a sequence through the **File->New->Sequence** screen.

Click on the drop-down list and select one of the existing sequences to add to the publication. Click **OK** to add the sequence; click **Cancel** to go back to the previous screen.

5.9.4 Script Tab Associates Existing Scripts With the Publication

You can only associate an existing script with the publication on this screen. To add an existing script, click **Add**.

> **Note:** You can import a script through the **File->New->Script** screen.

Click on the drop-down list and select one of the existing scripts to add to the publication. Click **OK** to add the script; click **Cancel** to go back to the previous screen.

It is important that all scripts follow the instructions listed in Section 5.7.1, "Writing SQL Scripts".

5.9.5 Resource Tab Associates Existing Resources With the Publication

You can only associate an existing resource with the publication on this screen. To add an existing resource, click **Add**.

> **Note:** You can import a resource through the **File->New->Resource** screen.

Click on the drop-down list and select one of the existing resources to add to the publication. Click **OK** to add the resource; click **Cancel** to go back to the previous screen.

5.9.6 Event Tab Configures Automatic Synchronization Rules for this Publication

When you select the Event Tab, you can configure data event rules for this publication, which apply to all automatic synchronization enabled publication items associated in this publication.

Data events define when an automatic synchronization is triggered.

- Client Data Events—Synchronize if the client database contains more than `<number>` modified records, where you specify the `<number>` of modifed records in the client database to trigger an automatic synchronization.
- Server Data Events—Synchronize if the out queue contains more than `<number>` modified records, where you specify the `<number>` of modifed records in the client database to trigger an automatic synchronization.

The lowest value that can be provided in these fields is 1. Specify a high value if you want the synchronization to occur based upon other rules. Click **Apply** when finished.

5.10 Import Existing Publications and Objects from Repository

You can import existing publications, publication items, sequences, scripts or resources that already exist within the repository by choosing the **Project->Add From Repository** option, as described in the following sections:

- Section 5.10.1, "Import Existing Publication from Repository"
- Section 5.10.2, "Import Existing Publication Item From the Repository"
- Section 5.10.3, "Import Existing Sequence From the Repository"
- Section 5.10.4, "Import Existing Resource From the Repository"
- Section 5.10.5, "Import an Existing Script From the Repository"

5.10.1 Import Existing Publication from Repository

You can add an existing publication that already exists in the repository to this project by selecting **Project->Add From Repository->Publication**. All associated objects—publication items, sequences, scripts, resources—are also pulled into the project with the publication.

To view all publications in the repository, click **Search**. All publications are shown in the left-hand screen. To limit the displayed publications to only those with a certain string as part of the name, provide this string in the Filter and then click **Search**. Only those publications that match the filter are shown.

Note: In the Search Filter, you can use the same pattern matching characters in a valid `SQL WHERE` clause. The filter is case-sensitive; use upper-case characters.

Select the desired publications and either double-click or select the right arrow to move them to the right window. Once all desired publications are in the right window, click **OK** to move these publications into the project.

5.10.2 Import Existing Publication Item From the Repository

You can add an existing publication item that already exists in the repository to this project by selecting **Project->Add From Repository->Publication Item**.

To view all publication items in the repository, click **Search**. All publication items are shown in the left-hand screen. To limit the displayed publication items to only those with a certain string as part of the name, provide this string in the Filter and then click **Search**. Only those publication items that match the filter are shown.

Note: To search only for objects that match a condition, designate the condition in the Filter box and click **Search**. You can use the same pattern matching characters in a valid SQL WHERE clause. The filter is case-sensitive; use upper-case characters.

Select the desired publication items and either double-click or select the right arrow to move them to the right window. Once all desired publication items are in the right window, click **OK** to move these publication items into the project.

Once added into the project, you still must associate them with the publication if you want to test the synchronization of the publication item. See Section 5.9.2, "Publication Item Tab Associates Publication Items With the Publication" for more information.

5.10.3 Import Existing Sequence From the Repository

You can add an existing sequence that already exists in the repository to this project by selecting **Project->Add From Repository->Sequence**.

To view all sequences in the repository, click **Search**. All sequences are shown in the left-hand screen. To limit the displayed sequences to only those with a certain string as part of the name, provide this string in the Filter and then click **Search**. Only those sequences that match the filter are shown.

Note: To search only for objects that match a condition, designate the condition in the Filter box and click **Search**. You can use the same pattern matching characters in a valid `SQL WHERE` clause. The filter is case-sensitive; use upper-case characters.

Select the desired sequences and either double-click or select the right arrow to move them to the right window. Once all desired sequences are in the right window, click **OK** to move these sequences into the project.

Once added into the project, you still must associate them with a publication if you want to test it with a synchronization. See Section 5.9.3, "Sequence Tab Associates Existing Sequences With the Publication" for more information.

5.10.4 Import Existing Resource From the Repository

You can add an existing resource that already exists in the repository to this project by selecting **Project->Add From Repository->**.

To view all resources in the repository, click **Search**. All resources are shown in the left-hand screen. To limit the displayed resources to only those with a certain string as part of the name, provide this string in the Filter and then click **Search**. Only those resources that match the filter are shown.

Note: To search only for objects that match a condition, designate the condition in the Filter box and click **Search**. You can use the same pattern matching characters in a valid SQL WHERE clause. The filter is case-sensitive; use upper-case characters.

Select the desired resources and either double-click or select the right arrow to move them to the right window. Once all desired resources are in the right window, click **OK** to move these resources into the project.

Once added into the project, you still must associate them with a publication if you want to test it with a synchronization. See Section 5.9.5, "Resource Tab Associates Existing Resources With the Publication" for more information.

5.10.5 Import an Existing Script From the Repository

You can add an existing script that already exists in the repository to this project by selecting **Project->Add From Repository->Script**.

To view all scripts in the repository, click **Search**. All scripts are shown in the left-hand screen. To limit the displayed scripts to only those with a certain string as part of the name, provide this string in the Filter and then click **Search**. Only those scripts that match the filter are shown.

Note: To search only for objects that match a condition, designate the condition in the Filter box and click **Search**. You can use the same pattern matching characters in a valid SQL WHERE clause. The filter is case-sensitive; use upper-case characters.

Select the desired scripts and either double-click or select the right arrow to move them to the right window. Once all desired scripts are in the right window, click **OK** to move these scripts into the project.

Note: All scripts added to the project must follow the guidelines as described in Section 5.7.1, "Writing SQL Scripts".

Once added into the project, you still must associate them with a publication if you want to test it with a synchronization. See Section 5.9.4, "Script Tab Associates Existing Scripts With the Publication" for more information.

5.11 Create a Virtual Primary Key

For fast refresh, you must have a primary key. If the table, view, or synonym does not currently have a primary key, you can designate one of the columns as the virtual primary key through this screen, as follows:

Note: Any virtual primary key must be unique and not null.

1. Using the drop-down lists, choose the following:
   - Schema name
   - Object type: table, view or synonym type
   - Any string that exists within the object name, if desired
2. Click **Search**, which brings up a list of available objects.
3. From the object list, choose the appropriate table, view, or synonym. Once chosen, the available columns are listed.

4. Select the column(s) that you wish to be the primary key and click **OK**.

If you have a composite primary key, iteratively add each column within the composite primary key.

5.12 Test a Publication by Performing a Synchronization

You can create a test to perform a synchronization of the designated publication. Click **Project->Test Publication**. When you create the test, MDW automatically creates the subscription for the user.

1. Click **Create** to design the test and provide the following information:

   - Name: If the test is remote, then the user name is populated with the registered owner of the remote target device. If the test is local, then the user name should be a valid Mobile user in the repository.
   - Publication: From the drop-down list, select one of the available publications in this project for this test.
   - Client type: Designate if the client is local or remote. Default is local. If Active Sync is not installed, the remote option is not available.
   - Specify a user that is defined in Mobile Manager.

   Click **OK** to save the test; click **Cancel** to revert back to the previous screen.

   Note: To remove any tests, select the test and click **Remove**.

2. Once created, click **Synchronize** to perform a synchronization for the designated publication. On the pop-up dialog, provide the password for the given username and the URL of the Mobile Server. The URL for the Mobile Server should be the `hostname/webtogo`.

   Click **Option** to specify priority of the publication items, as follows:

   - High Priority: Limits synchronization to server tables flagged as high priority, otherwise all tables are synchronized.
   - Push Only: Upload changes from the client to the server only, do not download. This is useful when data transfer is one way, client to server.
   - Complete Refresh: All data is refreshed from the server to the client.
   - Debug: Turn on debugging when synchronizing.
   - Selective Synchronization: Determine which publication and publication items are allowed to synchronize. When you click this option, move the publication items that you want to synchronize from the left window to the right window using the arrow buttons. For details on how selective synchronization performs, see Section 4.1.6, "Manage What Tables Are Synchronized With ocSetTableSyncFlag" and Section 4.2.7, "Manage What Tables Are Synchronized With Selective Sync".

   Click **OK** to save the synchronization options or **Cancel** to return to the previous screen.

5.13 Deploy the Publications in the Project to the Repository

You can deploy one or more of the publications in the current project from the development/test Mobile Server repository to a target production Mobile Server repository by clicking **File->Deploy**. You should adequately test all publications before deploying to the production Mobile Server repository.

All available publications are displayed in the project publications section. To limit the displayed publications to only those with a certain string as part of the name, provide this string in the Filter and then click **Search**. Only those publications that match the filter are shown.

Note: In the Search Filter, you can use the same pattern matching characters in a valid `SQL WHERE` clause. The filter is case-sensitive; use upper-case characters.

Select the desired publications and click **OK** to deploy these publications into the repository. A dialog appears where you specify the remote database connection information, as follows:

- User name and password for database connection authentication.
- JDBC Driver type: Based on the type of the JDBC driver, different information is required. At this time, you can only use the JDBC Thin driver. Provide the host name, port, and SID for the remote database.

Click **OK** to accept the input values for the remote database; click **Cancel** to return to the previous screen.

6 Developing Mobile Web-to-Go Applications

The following sections describe how to develop and test Web-to-Go applications:

- Section 6.1, "Choose the Type of Web-to-Go Mobile Client to Use"
- Section 6.2, "Developing and Testing the Application"

6.1 Choose the Type of Web-to-Go Mobile Client to Use

To install and set up the Mobile Client, see Section 17.5.1, "Install the Mobile Client for Web-to-Go".

There are two types of Web-to-Go applications:

- The original Oracle Database Lite Web-to-Go application that uses an Oracle Database Lite Servlet stack. You can still use this type of application, but the Oracle Database Lite Server stack is not J2EE 1.3 compatible.
- A Web-to-Go application built upon the OracleAS OC4J stack. Since the OC4J product is continually updated, then building your Web-to-Go application using the J2EE standards is better if you want to use future J2EE standards. This application is known as the OC4J Web-to-Go application.

  To build the OC4J Web-to-Go application, follow the J2EE standards specified by Sun Microsystems and then create the snapshot with MDW and publish the application with the EAR or WAR file within the Packaging Wizard.

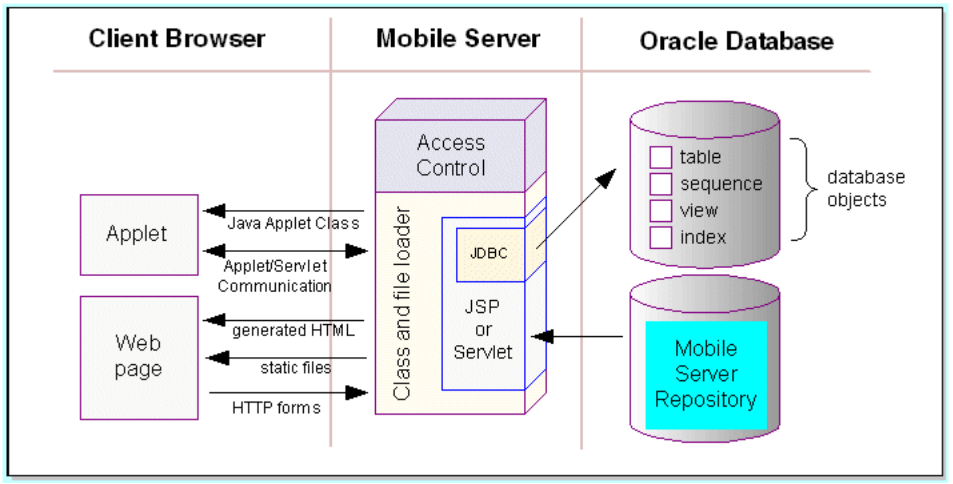
6.2 Developing and Testing the Application

Web-to-Go provides a high level Java API that provides easy-to-use functionality to developers of Mobile applications. Using this API, developers no longer need to write code for such functions as replication of database tables, database connections, security, directory locations, or deployment of applications to client devices.

In addition, the Mobile Development Kit allows developers to develop and debug Web-to-Go applications that contain Java applets, Java servlets, and JavaServer Pages (JSP).

Figure 6–1 displays the development architecture of the Mobile Server and the Oracle database.

Figure 6–1 Development Architecture



The following sections provide a discussion on how to develop Mobile applications for Web-to-Go. Topics include:

- Section 6.2.1, "Building Web-to-Go Applications"
- Section 6.2.2, "Application Roles"
- Section 6.2.3, "Developing JavaServer Pages"
- Section 6.2.4, "Developing Java Servlets for Web-to-Go"
- Section 6.2.5, "Using Web-to-Go Applets"
- Section 6.2.6, "Developing Applet JDBC Communication"
- Section 6.2.7, "Developing Applet Servlet Communication"
- Section 6.2.8, "Debugging Web-to-Go Applications"
- Section 6.2.9, "Customizing the Workspace Application"
- Section 6.2.10, "Using the Mobile Server Admin API"

6.2.1 Building Web-to-Go Applications

Web-to-Go applications adhere to Web standards and use browsers to display user interface elements in a graphical user interface. Generally, Web-to-Go applications access and manipulate data stored in databases. These applications contain static, dynamic, and database components. You can create static and dynamic components using development tools and use the Packaging Wizard to store them in the Mobile Server Repository. You can create and store the application's database components in an object relational database (Oracle Database Lite or Oracle). The following table provides examples of each component type.

Table 6–1 provides examples of each database component type.

Table 6–1 Database Component Types

Component Type	Example
static	HTML files, image files (such as GIF and JPG), HTML templates
dynamic	Java servlets, Java applets and JavaServer pages
database	tables, snapshots, and sequences

6.2.1.1 Static Components

Static components are HTML files that do not change, such as graphical elements (GIF files and JPG files), and textual elements (HTML files and templates).

6.2.1.2 Dynamic Components

Java Applets, Java Servlets, and JavaServer Pages (JSP) are dynamic components that create dynamic Web pages. Java applets, create a rich graphical user interface, while Java servlets and JSPs extend server side functionality.

6.2.1.3 Database Components

Snapshots and sequences are the two database components that Web-to-Go supports. On the Mobile Server, the snapshot definition incorporates information about the table whose snapshot was taken. Web-to-Go also executes custom DDLs (Data Definition Language) statements, enabling the creation of such database objects as views and indexes.

Note: DDLs are only supported on Windows32 and WindowsCE platforms.

6.2.1.4 Database Connections

Database connections are both application based and session based. For a given session, Web-to-Go maintains a separate connection for each application. If an application runs multiple servlets simultaneously, they use the same connection object. This may occur if the application uses multiple frames or if a user accesses the application with two separate browser windows.

6.2.2 Application Roles

It is common for applications to display different functionality depending on the type of user who is running the application. For example, an application may show different menu items depending on whether manufacturing managers or shipping clerks are running the application.

You can accomplish this in Web-to-Go by defining application roles. The application behavior then changes depending on whether or not a user has a specific role.

In the above example, you can define the application role `MANAGER`. In your application code, where you generate the menu, you must check if the user has the role `MANAGER`, and display the correct menu items.

You will use the Packaging Wizard to define application roles in Web-to-Go. You can assign roles to users and groups through the Mobile Manager. However, it is up to the application developer to determine and implement application behavior, if the user has a specific role.

You can query the Web-to-Go user context to retrieve a list of roles that are created for users.

6.2.3 Developing JavaServer Pages

Web-to-Go handles HTTP requests for JavaServer Pages (JSP) using the Mobile client Web Server, Mobile Server, and Mobile Client for Web-to-Go.

6.2.3.1 Mobile Server or Mobile Development Kit Web Server

After the Mobile Server receives an HTTP request for a JSP, it checks if the JSP source file and corresponding class file exist. If the class file exists and is newer than the JSP source file, the Mobile Server loads the Java class and executes the servlet.

If the class file does not exist, or is older than the JSP source file, the Mobile Server automatically converts the JSP source file into a Java source file and compiles it into a Java class under the `APP_HOME/_pages`. After the JSP has been converted and compiled, the Mobile Server (or the Mobile Development Kit Web Server) loads the Java class and executes the servlet.

6.2.3.2 Mobile Client for Web-to-Go

After the Mobile Client for Web-to-Go receives the HTTP request for a JavaServer page, the corresponding Java class is loaded from the `APP_HOME/_pages` directory and is executed. Since the Mobile Client for Web-to-Go assumes that the corresponding class file exists, you must convert the JSP source file into a class file. While deploying the application using the Packaging Wizard, you must include both the JSP source file and the corresponding class file. You can create the class files using the Packaging Wizard tool or manually, using the Oracle JSP (OJSP) command line translator.

List your JSP files in the Files panel of the Packaging Wizard and click **Compile** under the Files tab. The Packaging Wizard automatically locates all the JSP files that you have listed and automatically compiles all of them. The Packaging Wizard adds the `compile` class to the application package.

6.2.4 Developing Java Servlets for Web-to-Go

You develop Web-to-Go Java servlets with the Mobile Development Kit. The Mobile Development Kit for Web-to-Go simplifies the process of writing Mobile Server servlets. Before using the Mobile Development Kit for Web-to-Go, you must first install it on the development client. The Mobile Development Kit for Web-to-Go contains a Web server called the Mobile client Web Server that executes Java servlets. You can use the Mobile client Web Server to run and debug Java servlets.

- Section 6.2.4.1, "Limitations"
- Section 6.2.4.2, "Accessing Applications on the Mobile Development Kit for Web-to-Go"
- Section 6.2.4.3, "Creating a Servlet"
- Section 6.2.4.4, "Running a Servlet"
- Section 6.2.4.5, "Accessing the Schema Directly in Oracle Database Lite"

6.2.4.1 Limitations

The Mobile Development Kit for Web-to-Go Web server is a scaled down version of the Mobile Server and has the following limitations.

- It contains no application repository. As a result, the Mobile Development Kit for Web-to-Go Web server loads all files and classes directly from the file system.
- Security and access control are disabled.
- Clients that connect to the Mobile Development Kit for Web-to-Go Web server cannot go off-line.

- It provides connection management only to Oracle Database Lite. It connects the user to the schema `SYSTEM` in the Oracle Database Lite named `webtogo`.

6.2.4.2 Accessing Applications on the Mobile Development Kit for Web-to-Go

You can access applications on the Mobile Development Kit for Web-to-Go Web server by performing the following steps.

1. To launch the Mobile Development Kit for Web-to-Go Web server, start the Command Prompt and enter the following.

   ```
   cd <ORACLE_HOME>\mobile\sdk\bin
   wtgdebug.exe
   ```

2. Use your browser to connect to the Mobile Development Kit for Web-to-Go Web server using the following URL.

   ```
   http://machine_name:7070/
   ```

   The Mobile Development Kit for Web-to-Go page displays icons that represent an application in the Mobile client Web Server. Note that port 7070 is the default port for debugging Web-to-Go. For more information, see the file `webtogo.ora` under the following location.

   ```
   <ORACLE_HOME>\mobile\sdk\bin\webtogo.ora
   ```

3. Click the icon of the application that you want to access.

6.2.4.3 Creating a Servlet

Web-to-Go uses servlets to handle HTTP client requests. Servlets handle HTTP client requests by performing one of the following tasks.

- Creating dynamic HTML content and returning it to the browser.
- Processing and submitting HTML forms using an `HTTP POST` request.

Servlets must extend the `HttpServlet` abstract class defined in the Java Servlet API. The following is a servlet example.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HelloWorld extends HttpServlet
{
   /**
   * Process the HTTP POST method
   */

   public void doPost (HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
   {
     writeOutput("doPost", request, response);
   }

   /**
   * Process the HTTP GET method
   */
   public void doGet (HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
   {
      writeOutput("doGet", request, response);
```

```
   }

   /**
   * Write the actual output
   */

   public void writeOutput (String method, HttpServletRequest  request,
                            HttpServletResponse response)
   throws ServletException, IOException
   {
      PrintWriter out;

      // set content type
      response.setContentType("text/html");

      // Write the response
      out = response.getWriter();

      out.println("<HTML><HEAD><TITLE>");
      out.println("Hello World");
      out.println("</TITLE></HEAD><BODY>");
      out.println("<P>This is output from HelloWorld "+method+"().");
      out.println("</BODY></HTML>");
      out.close();
   }
}
```

6.2.4.3.1 Packages Web-to-Go provides the following Java package:

```
oracle.lite.web.applet
```

This package contains the classes to be used with Web-to-Go applets. It contains the `AppletProxy` class which is used as a proxy for Web-to-Go applets requiring JDBC connections or communicating with a servlet on the Mobile Server. It also contains a few more classes which are used by the `AppletProxy` class to communicate with the Mobile Server. For more information, see the `oracle.lite.web.applet` package as documented in the *Oracle Database Lite API Specification*.

6.2.4.3.2 Web-to-Go User Context Web-to-Go creates a user context (or user profile) for every user who logs in to Web-to-Go. Web-to-Go applications always run within the user's specific context. Servlets, which are always part of an application, can use the user context (in which it is running) to access the services provided by Web-to-Go. The user context can then be used to obtain the following information.

- Name of the user
- Application that a user is accessing
- The database connection
- Roles that the user has for this application
- Name or value pairs stored in the registry for the user

Servlets can access the user profile through the standard named `java.security.Principal` obtained through the `getUserPrincipal` method of the `javax.servlet.http.HttpServletRequest` class.

This object can also be obtained from the `HttpSession` object. For example,

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
```

```
{

   // Retrieve the database connection from the User Profile,
   // which can be accessed from the HttpRequest
      HttpSession session = request.getSession(true);
      OraUserProfile profile =
(OraUserProfile)session.getAttribute("x-mobileserver-user");
   .
   .
   .
}
```

6.2.4.3.3 Database Connectivity in Java Code Servlets can obtain a connection to the Oracle database, using the following statement.

```
HttpSession sess = request.getSession();
WTGUser user = (WTGUser)sess.getAttribute("x-mobileserver-user");
Connection conn = user.getConnection() ;
```

6.2.4.3.4 Accessing the Mobile Server Repository Servlets can open or create a new file in the application repository. Access to the Mobile Server Repository is provided through the servlet context, which can be obtained by calling the `getServletContext()` from within the servlet. For example:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
   // Retrieve the servlet context
   ServletContext ctxt = getServletContext();

   // Open an input stream to the file input.html in the Mobile Server Repository
   // All file names are relative to the application's repository directory
   InputStream in = ctx.getResourceAsStream("input.html");

  // Open an output stream to the file output.html in the Mobile Server Repository
  // All file names are relative to the application's repository directory
  URL           url = ctxt.getResource ("output.html");
  URLConnection  conn = url.openConnection();
  OutputStream   out  = conn.getOutputStream();
  ...
}
```

6.2.4.4 Running a Servlet

Before you can execute the servlet, perform the following:

- Section 6.2.4.4.1, "Registering Servlets Using runwtgpack.bat"
- Section 6.2.4.4.2, "The webtogo.ora File"
- Section 6.2.4.4.3, "Using wtgdebug.exe"

6.2.4.4.1 Registering Servlets Using runwtgpack.bat Before you can access servlets from the browser, you need to register them with the Mobile client Web Server. To register servlets, you must first register the application and then add the servlets to it. As Web-to-Go enables you to register multiple applications, it displays a list of all registered applications.

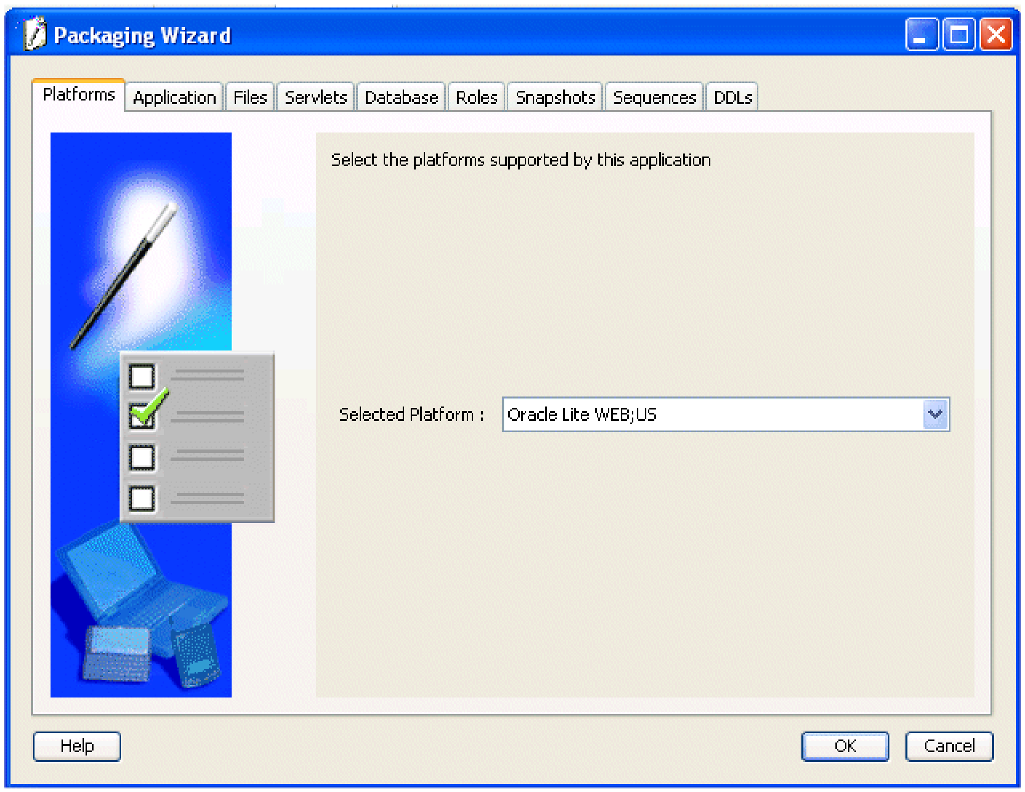
The Mobile Development Kit for Web-to-Go includes the Packaging Wizard, a tool for registering applications and servlets. You can invoke the Packaging Wizard by entering the following at the command line.

```
C:\> runwtgpack -d
```

Initially, you select whether to create a new application or to continue work on an existing application.

Figure 6–2 displays the Make a Selection dialog.

Figure 6–2 Make a Selection Panel



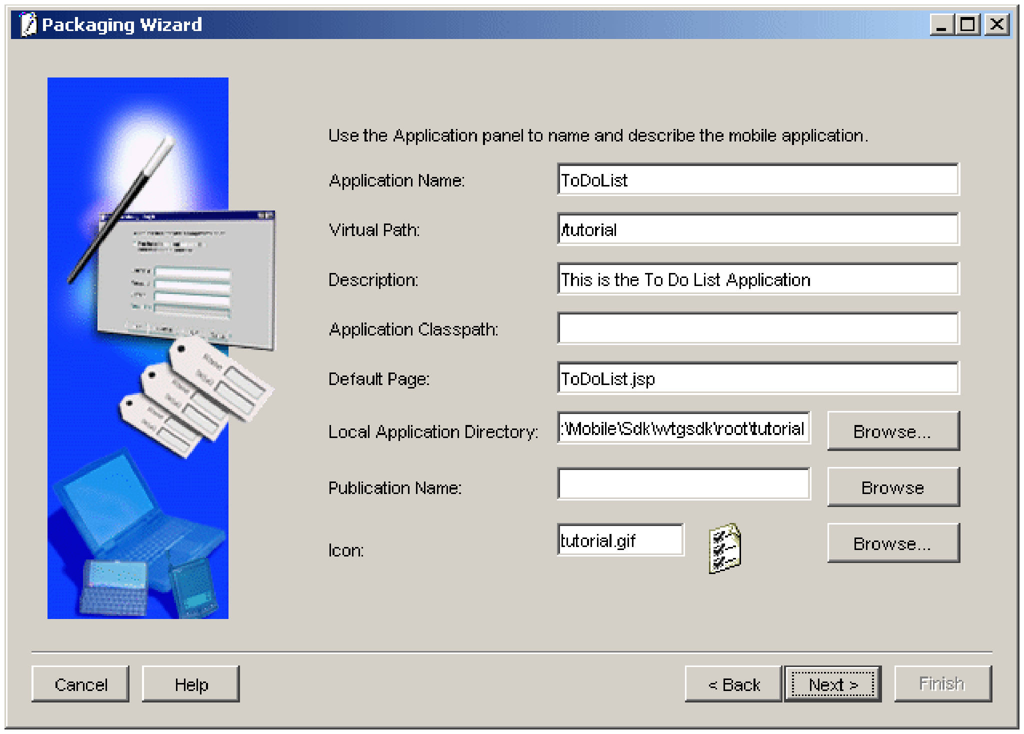
Select the platforms supported by this application

Selected Platform : Oracle Lite WEB;US

After you make your selection and click **OK**, the Applications dialog appears.

Figure 6–3 displays the Applications dialog.

Figure 6–3 Applications Panel


Packaging Wizard

Use the Application panel to name and describe the mobile application.

Application Name: ToDoList

Virtual Path: /tutorial

Description: This is the To Do List Application

Application Classpath:

Default Page: ToDoList.jsp

Local Application Directory: :\Mobile\Sdk\wtgsdk\root\tutorial

Publication Name:

Icon: tutorial.gif

For detailed instructions on how to use the Packaging Wizard, see Chapter 17, "Tutorial for Building Mobile Web-to-Go Applications".

6.2.4.4.2 The webtogo.ora File The configuration information for the Web server and the Packaging Wizard is stored in the `webtogo.ora` file.

Table 6–2 describes `webtogo.ora` parameters.

Table 6–2 Webtogo.ora Parameters

Parameter Name	Description
`ROOT_DIR`	The Mobile Server expands all file paths that are relative to its root directory. You can change the root directory by modifying the value of the parameter named `ROOT_DIR` in the `webtogo.ora` file. The default parameter value is given below. `<ORACLE_HOME>\mobile\sdk\wtgsdk\root`
`PORT`	The port on which the Web server listens. The default value is `80`. The default value for the Mobile client Web Server is `7070`.
`XMLFILE`	The XML file that contains the application information. The Packaging Wizard creates and maintains the XML file. You can modify the XML file using the Packaging Wizard.

For more information, refer the discussion of initialization parameters in the *Oracle Database Lite Administration and Deployment Guide*.

6.2.4.4.3 Using wtgdebug.exe Perform the following to debug your application with the `wtgdebug` executable:

1. Using the Command Prompt, enter `wtgdebug.exe`.

2. Use a browser to connect to the Mobile client Web Server located at the following URL.

   ```
   http://machine_name:port
   ```

   This Mobile client Web Server displays the list of applications that are currently known to the Mobile client Web Server. The Mobile client Web Server retrieves this list from the XML file. By default, this list includes the sample applications `Servlet Runner` and `Sample`.

3. Select the application to debug. This action launches a new browser window which you can use to step through the application.

> **Note:** If you change and recompile your servlet, you need to restart the Web server. You can stop the Web server by pressing Control+C.

6.2.4.5 Accessing the Schema Directly in Oracle Database Lite

The Mobile Development Kit for Web-to-Go automatically creates a database connection to Oracle Database Lite. This database connection connects to the database schema `SYSTEM`. Within your servlet code, you can obtain this connection from the HTTP request. You can also connect to Oracle Database Lite directly using ODBC. Connecting to Oracle Database Lite directly by using ODBC is helpful for performing the following tasks.

- Creating schema objects such as tables, view and sequences
- Manually checking the contents table

To connect to Oracle Database Lite, launch `msql` using the Command Prompt.

```
msql system/manager@jdbc:polite:webtogo
```

6.2.5 Using Web-to-Go Applets

Web-to-Go supports Java applets. For security reasons, Web-to-Go applets must communicate with the Mobile Server or the Oracle database by using a proxy class. The `AppletProxy` class acts as a proxy for Web-to-Go applets and provides the applet with the required methods for communicating with the Web-to-Go servlet or for making a JDBC connection. An instance of the `AppletProxy` should be created while instantiating the applet. Once the instance of the `AppletProxy` class is created, the `AppletProxy` object communicates with the Mobile Server and derives all the requisite information to connect to the server or to make a JDBC connection to the Oracle database.

6.2.5.1 Creating the Web-to-Go Applet

The Web-to-Go applet extends the `java.applet.Applet`. When the `init()` method initializes the Web-to-Go applet, it creates an instance of the `AppletProxy` class by passing the Applet reference as the parameter. Once you create an instance of the `AppletProxy` class, you can use different methods of the `AppletProxy` class for communicating with the servlet or for establishing a JDBC connection with the Oracle database. For example,

```
import oracle.lite.web.applet.*;
public class AppApplet extends Applet
{

   public void init()
```

```
   {
      ..
      ..
      // Create Instance and pass Reference of applet as parameter
      proxy = new AppletProxy(this);
   }
   AppletProxy proxy;
}
```

The applet can use the following methods to communicate with the servlet. Each method requires an instance of the `AppletProxy` class.

- `getResultObject()`
- `setSessionId()`
- `showDocument()`

The applet can use the `getConnection()` method to establish a JDBC connection with the database.

6.2.5.2 Creating the HTML Page for the Applet

The Web-to-Go applet is launched from an HTML page that contains the following tags.

```
<html>
<body>
<applet ARCHIVE="/webtogo/wtgapplet.jar" CODE="MyApplet.class" WIDTH=200
HEIGHT=100>
<PARAM NAME="ORACLE_LITE_WEB_SESSION_ID"  VALUE="123">
</applet>
</body>
</html>
```

The `AppletProxy` class uses the value of the `ORACLE_LITE_WEB_SESSION_ID` parameter to obtain the `SessionID` from the Mobile Server. The `SessionID` is subsequently added to every request an applet makes to a servlet. You can write the HTML code in a static HTML page or you can generate it from a servlet.

6.2.5.2.1 Static HTML Page Web-to-Go can automatically add the parameter to any static page containing the `APPLET` tag. For this option, you must change the HTML page's extension to `.ahtml` as demonstrated in the following syntax.

page_name.ahtml

When the client accesses the HTML page, a Web-to-Go system servlet adds the required `<PARAM>` tag for the `ORACLE_LITE_WEB_SESSION_ID` parameter, to the HTML output. For example,

```
<PARAM NAME="ORACLE_LITE_WEB_SESSION_ID" VALUE="123">
```

The Web-to-Go system servlet sets the `VALUE` attribute to your Web-to-Go `SessionID`.

6.2.5.2.2 HTML Page Generated from a Servlet You can also dynamically generate the HTML page that contains the `<APPLET>` tag. When you generate the HTML page dynamically, you must add the `SessionID` parameter manually. You can retrieve the `SessionID` information from the `oraUserProfile` as follows.

```
import oracle.lite.web.html.*;
import oracle.lite.web.servlet.*;
```

```
public class AppServlet extends HttpServlet
{
   public void doGet(HttpServletRequest req, HttpServletResponse resp)
   {
      PrintWriter out = new PrintWriter(resp.getOutputStream());
      out.println("<HTML>");
      out.println("<BODY>");
      out.println("<APPLET ARCHIVE="/webtogo/wtgapplet.jar"
                  CODE='MyApplet.class' WIDTH=200 HEIGHT=100>");
      // Add these lines to add one more PARAM tag in html page
      // This code should be added in-between  <APPLET> and  </APPLET> tag
      OraHttpServletRequest ora_request    = (OraHttpServletRequest) req;
      OraUserProfile        oraUserProfile = ora_request.getUserProfile();
      out.println(" <PARAM NAME=\"ORACLE_LITE_WEB_SESSION_ID\" VALUE=\""
                  +oraUserProfile.getAppletSessionId(req)+"\"> ");
      out.println("</APPLET>");
      out.println("</BODY>");
      out.println("</HTML>");
      out.close();
   }
}
```

6.2.6 Developing Applet JDBC Communication

You can develop Java applets that access the database using a JDBC connection. Once you create an instance of the `AppletProxy` class, you must use the `getConnection` method of the `AppletProxy` class to obtain a JDBC connection. The `getConnection` method returns the `JDBCConnection` object.

Note: The `AppletProxy` class is described in Section 6.2.5.1, "Creating the Web-to-Go Applet".

6.2.6.1 getConnection()

You can use the `getConnection` method to obtain a `JDBCConnection`. The `getConnection` method determines whether the connection is connected or disconnected and provides access to the Oracle database, if connected, or to the Oracle Database Lite, if disconnected, to the user.

Example

```
import oracle.lite.web.applet.*;
public class AppApplet extends Applet
{
   public void init()
   {
      ...
      // Create Instance and pass Reference of applet as parameter
      proxy = new AppletProxy(this);
   }
   public java.sql.Connection getDataBaseConnection()
   {
      java.sql.Connection  dBConnection = proxy.getConnection();
      return dBConnection;
   }
   AppletProxy proxy;
}
```

6.2.6.2 Design Issue

The Web-to-Go applet holds the database connection even after the user exits Web-to-Go. The applet maintains the connection even if the user types a new URL in the browser or clicks the **Back** button. Web-to-Go application designers must ensure that their applications explicitly close the database connection when the user exits Web-to-Go.

Example

You can close the connection by calling the following statement.

```
dBConnection.close()
```

6.2.7 Developing Applet Servlet Communication

You can develop Java applets that communicate with Java servlets in the Web-to-Go environment. When a client first connects to the Mobile Server, the server generates a `SessionID` and sends it back to the client. Each subsequent client request to the server contains this `SessionID`. The Mobile Server authenticates the `SessionID` before executing the client's request. When applets communicate with Web-to-Go servlets, each applet request must also contain this `SessionID`. The `setSessionId` method in the `AppletProxy` class can be used to add the `SessionID` to each applet request. The `AppletProxy` class also contains other methods that provide communication between applets and servlets.

Note: The `getResultObject` and `showDocument` methods can be used to communicate with the Java servlet. Use the `setSessionID` method if you want to create your own URL connection object.

6.2.7.1 Creating the Web-to-Go Servlet

Servlets must extend the `HttpServlet` abstract class defined in the Java Servlet API. The following example creates a servlet called `HelloWorld` that extends the `HttpServlet` class. The servlet sends the `Hello World` string to the applet that calls it as an object.

Example

```
public class HelloWorld extends HttpServlet
{
   public void doGet (HttpServletRequest request, HttpServletResponse response)
   {
      ObjectOutputStream out = new ObjectOutputStream (resp.getOutputStream());
      Object obj = (Object) "Hello World" ;
      out.writeObject(obj);
      out.close();
   }
}
```

6.2.7.1.1 getResultObject() The Web-to-Go applet uses the `getResultObject()` method to communicate with the Web-to-Go servlet by passing the servlet URL and the `ServletParameter` object as parameters. The servlet responds to the applet request with a text string. The `ServletParameter` object can be either an object that can be serialized or a string containing `name/value` pairs. If the servlet accepts parameters, you can call the `getResultObject` method and pass the servlet parameters as one of the arguments.

Example

```
public Object getResult()
{
   java.net.URL url = new URL("http://www.foo.com/EmpServlet");
   String ServletParameter = "empname=John";
   Object resultObject = proxy.getResultObject(url, ServletParameter);
   return resultObject;
}
```

6.2.7.1.2 setSessionID() You can use the `setSessionID` method for adding a `SessionID` to an existing `URLConnection` object. When you write the applet-servlet communication mechanism, call `setSessionID` (URLConnection) at the end of the method. The method adds a `SessionID` to the passed `URLConnection` object and then returns the `URLConnection` object.

Example

```
public void YourMethod()
{
   java.net.URL url = new URL("http://www.foo.com/MyServlet");
   java.net.URLConnection con = url.URLConnection();
   ...
   // pass the URLConnection to the method setSessionId
   con = proxy.setSessionID(con);
   // Do whatever you want to do with this URLConnection object
   ObjectOutputStream out = new ObjectOutputStream(con.getOutputStream());
   out.writeObject(obj);
   out.flush();
   out.close();
}
```

6.2.7.1.3 showDocument() The `showDocument` method displays any static document including those with a suffix of `.html`, `.doc`, `.xls`, or any other one defined by the user. The `showDocument` method retrieves these documents from the Mobile Server and displays them in the client browser. To display documents, a user must have access permissions for the document and must have the correct `MIME` type set in the Mobile Server. The `showDocument` (String `relativeDocUrl`, String `winName`) method displays the document in a different browser window identified by a window name that is passed in the `winName` parameter. The following method launches the help file from the server in a browser window named `'helpwin'`.

Example

```
public void showHelp()
{
   String relativeDocUrl = "Help/HelpIndex.html";
   proxy.showDocument (url, helpWin);
}
```

To show the document in the same browser window as your applet, use call `showDocument(url)` as given below.

```
public void showHelp()
{
   String relativeDocUrl = "Help/HelpIndex.html";
   proxy.showDocument (url);
}
```

6.2.8 Debugging Web-to-Go Applications

You can run Web-to-Go applications inside a Java debugger if you have already installed the Mobile Development Kit for Web-to-Go and a Java debugger, such as the Oracle9*i* JDeveloper, Borland's JBuilder, or Visual J++. The example in this section assumes you are using Oracle9*i* JDeveloper. However, most of the information provided is also relevant to other debuggers.

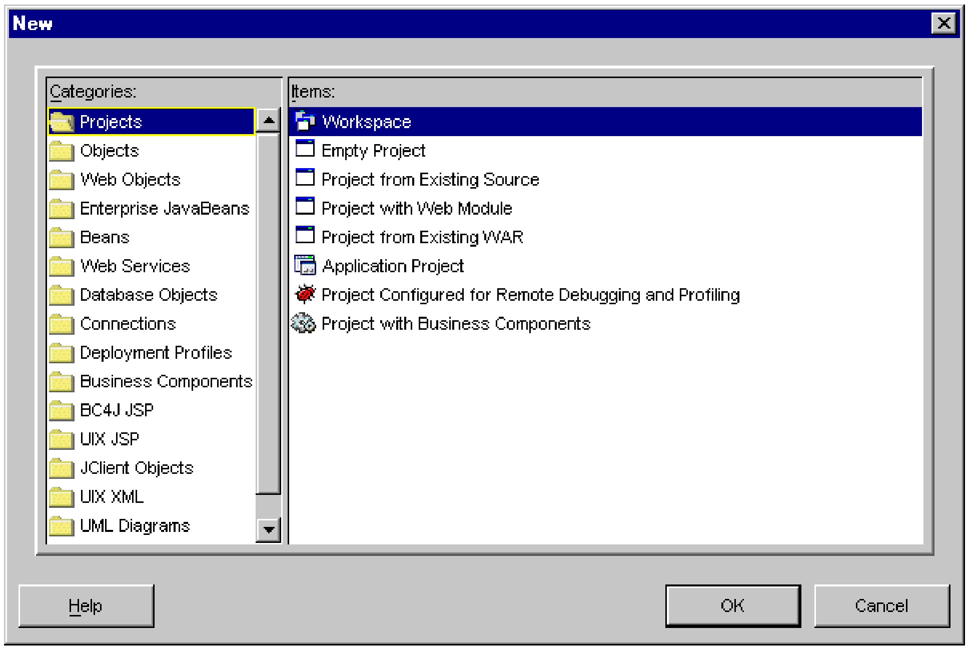
6.2.8.1 Running Sample 1 Using Oracle9*i* JDeveloper

This section discusses how to configure the Oracle9*i* JDeveloper to run the Sample 1 application that is bundled with the Mobile Development Kit for Web-to-Go. For detailed information and full documentation on how to use Oracle9*i* JDeveloper, consult the online help in Oracle9*i* JDeveloper and Oracle9*i* JDeveloper's documentation.

6.2.8.1.1 Creating a Debug Project To create a new debug project in Oracle9*i* JDeveloper, perform the following steps.

1. Start JDeveloper.
2. To create a new project in JDeveloper, click **File**, then click **New** (assuming you have defined a workspace in JDeveloper).
3. From the **Directories** menu in the left panel, select **Projects**, as displayed in Figure 6–4, then select **Empty Project**.

Figure 6–4 Creating a New Project

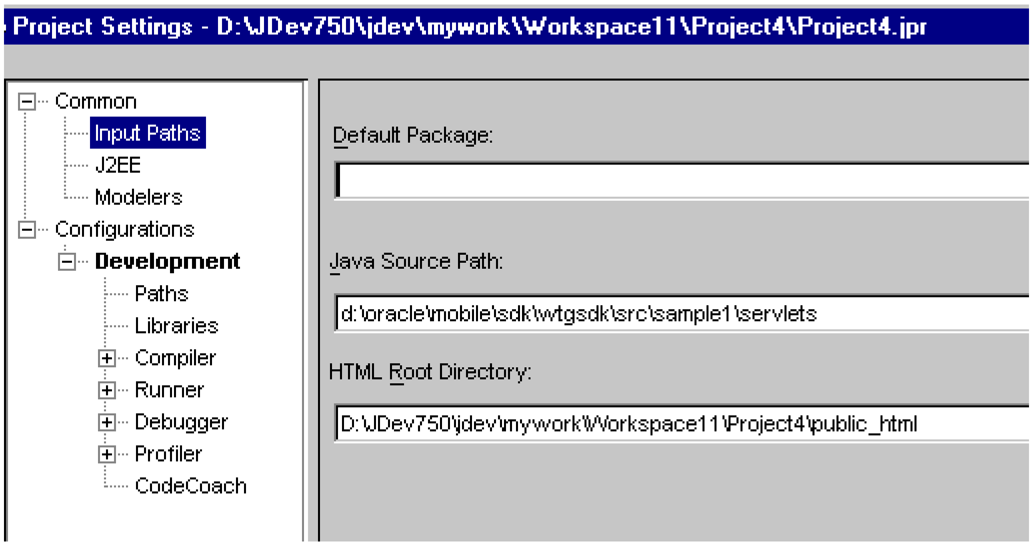


4. Set the Project Settings for your new project. Right click on **Project** to retrieve Project Settings. In the Project Settings dialog, expand Common in the left panel and select Input Paths. In the right panel, enter the following information in the Java Source Path field, as displayed in Figure 6–5.

```
<ORACLE_HOME>\mobile\sdk\wtgsdk\src\sample1\servlets
```

Leave the **Default Package** field blank. Do not change the default **HTML Root Directory**.

Figure 6–5 Project Settings - Input Paths



Project Settings - D:\JDev750\jdev\mywork\Workspace11\Project4\Project4.jpr

Default Package:

Java Source Path: d:\oracle\mobile\sdk\wtgsdk\src\sample1\servlets

HTML Root Directory: D:\JDev750\jdev\mywork\Workspace11\Project4\public_html

5. Expand **Configurations** and then **Development** in the left panel. Select **Paths**, which appears below Development in the left panel. In the **Output Directory** field, in the right panel, enter the following information.

   ```
   <ORACLE_HOME>\mobile\sdk\wtgsdk\root\sample1\servlets
   ```

6.2.8.1.2 Creating a Library Oracle9*i* JDeveloper makes it easier to manage sets of `JAR` files by using libraries instead of `CLASSPATH` settings.

Files for the WTGSDK Library

Create a WTGSDK library with the following `JAR` files and add this library to your project.

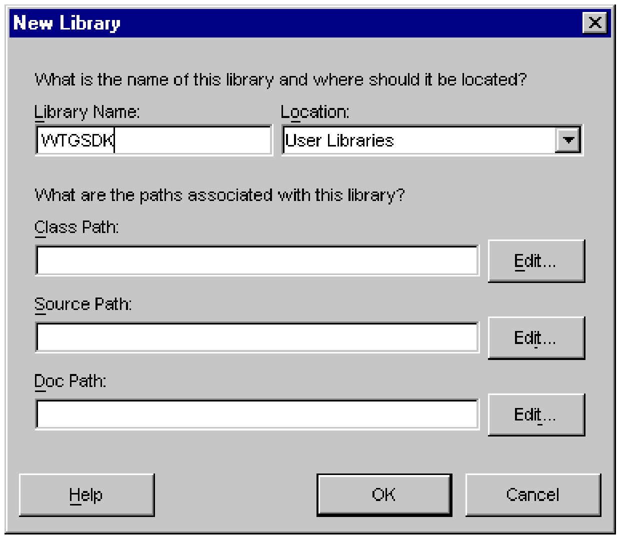
```
<ORACLE_HOME>\mobile\classes\ojsp.jar
<OLITE_HOME>\bin\olite40.jar
<ORACLE_HOME>\mobile\sdk\bin\webtogo.jar
<ORACLE_HOME>\mobile\classes\servlet.jar
<ORACLE_HOME>\mobile\classes\xmlparser.jar
<ORACLE_HOME>\mobile\classes\classgen.jar
<ORACLE_HOME>\mobile\classes\wtgpack.jar
```

Creating a WTGSDK Library

Perform the following steps to create a WTGSDK library.

1. Select **Libraries** in the left panel, then click **New** in the right panel.
2. The **New Library** dialog appears, as illustrated in Figure 6–6. In the **Library Name** field, enter WTGSDK.

Figure 6–6 The New Library Dialog



New Library

What is the name of this library and where should it be located?

Library Name: WTGSDK

Location: User Libraries

What are the paths associated with this library?

Class Path:

Source Path:

Doc Path:

3. Click **Edit...** next to the Class Path field. The **File** dialog appears.
4. From the appropriate directory, select the six `.jar` files that are listed above.
5. To add the files, click **OK**.

6.2.8.1.3 Adding Files to the Project To add the `Sample1` files to your project, perform the following steps.

1. Click the **green plus-sign** in the Oracle9*i* JDeveloper System-Navigator to add the Java sources to the project. The **File** dialog appears.
2. Select the Java source file `Helloworld.java` in the directory <*ORACLE_HOME*>`\mobile\sdk\wtgsdk\src\sample1\servlets`, and click **Open**.
3. Also, add the file `RunWebServer.java`, which is located in the directory <*ORACLE_HOME*>`\mobile\sdk\wtgsdk\src`, to the project.
4. A dialog appears prompting you to update the project source path. Click **No**.

6.2.8.1.4 Running and Debugging Set one or more breakpoints in your code by right-clicking at the statement where you want to break. Select **Toggle breakpoint**. The background of the statement becomes red, indicating the breakpoint.

1. Select the file `RunWebServer.java` in the **System-Navigator** window.
2. Choose **Debug** by right clicking on the file that you selected to start the Mobile Server inside the debugger.

The Mobile Server is now ready for use. You can access it through your Web browser, by accessing the following URL.

```
http://<machine_name>
```

Where `<machine_name>` is the host name of the computer on which you are running Oracle9*i* JDeveloper.

6.2.8.1.5 Troubleshooting This section describes troubleshooting options that you can implement.

Improving Performance

When you run the Mobile Server inside the Java debugger and access it using a Web browser, performance may decrease. To improve performance, perform the following tasks.

1. Run the Web browser on a different machine.
2. Using the **Task Manager**, set the priority of the Web browser process to `LOW` after you start the Web browser.

6.2.9 Customizing the Workspace Application

The Mobile Development Kit for Web-to-Go includes a set of APIs that contain a basic Web-to-Go workspace application. Developers can use these APIs to replace the standard Web-to-Go workspace application with a customized version with the following restrictions:

- This can only be customized for the Web-to-Go client only. This is not supported for the Web-to-Go OC4J client.
- You cannot customize the workspace if both the Web-to-Go client and Web-to-Go OC4J client are used.
- The customized workspace is part of the Windows version of the MDK only.
- The Mobile Server and MDK must be installed in the same <*ORACLE_HOME*>.

These APIs provide the following functionality.

- Login
- Logoff
- Synchronize
- List User Applications
- Change User's Password

For more information on the APIs used to build a customized Web-to-Go workspace application, see the *Oracle Database Lite API Specification*, which you can view off the main documentation index, which is located at the following:

```
<ORACLE_HOME>\mobile\index.htm
```

1. Develop the customized Web-to-Go workspace application using the Web-to-Go APIs.
2. Create an Oracle Database Lite database called `webtogo` and load the new Web-to-Go workspace application into it. The database acts as the Mobile Server Repository in the Mobile Client for Web-to-Go. For more information, refer to the file `crclient.bat`, which is included in the sample Web-to-Go workspace application.
3. Create a `webtogo.ora` file for the Mobile Client for Web-to-Go, which instructs the Mobile Server to use the customized Web-to-Go workspace application. For the correct parameter settings in the `webtogo.ora` file, refer the section, Section 6.2.9.1, "Web-to-Go Parameters".
4. Load the `webtogo.odb` file, which is created by the Mobile Client for Web-to-Go, the `webtogo.ora` file for the Mobile Client for Web-to-Go, and the Web-to-Go workspace into the Mobile Server Repository. For more information, refer to the file `crserver.bat`, which is included in the sample Web-to-Go workspace application.

5. Instruct the Mobile Server to use the new Web-to-Go workspace application by modifying the `webtogo.ora` file on the server. For the correct parameter settings in the `webtogo.ora`, refer the section Section 6.2.9.1, "Web-to-Go Parameters".

6.2.9.1 Web-to-Go Parameters

To instruct Web-to-Go to use a customized Web-to-Go workspace application, you must set the following parameters in the `[WEBTOGO]` section of the `webtogo.ora` file.

Table 6–3 describes `webtogo.ora` parameter settings.

Table 6–3 Setting webtogo.ora Parameters

Parameter	Setting
`CUSTOM_WORKSPACE`	YES
`CUSTOM_DIRECTORY`	Repository directory of the Web-to-Go workspace application. For example, `/myworkspace`.
`DEFAULT_PAGE`	The entry point of the Web-to-Go workspace application. For example, `myfirstpage.html`.
`CUSTOM_FIRSTSERVLET`	The name of the servlet that you want to use in your customized workspace. For example, `CUSTOM_FIRSTSERVLET=HelloWorld;/hello`

Note: Web-to-Go supports only one workspace application per Mobile Server.

6.2.9.2 Sample Workspace

The Mobile Development Kit for Web-to-Go includes a sample Web-to-Go workspace application that illustrates how to use the Web-to-Go workspace API. Developers can use this sample application as a starting point when developing their Web-to-Go workspace applications. The sample Web-to-Go workspace application is written using JavaServer Pages (JSP) and `.html` files. The JSP files are located in the `myworkspace/out` directory in the Mobile Development Kit for Web-to-Go. These files are compiled into class files that are copied into `myworkspace/out` directory. This directory also contains all `.html` files and image files that are used by the sample Web-to-Go workspace application.

The Mobile Development Kit for Web-to-Go includes the following scripts that compile the JSP files, create the Oracle Database Lite named `webtogo` for the Mobile Client for Web-to-Go, and load all necessary files into the Mobile Server Repository.

Table 6–4 describes scripts available for JSP compilation.

Table 6–4 Scripts for JSP Compilation

Script Name	Description
`compile.bat`	Compiles `.jsp` files and copies the class files to the `myworkspace/out` directory.
`crclient.bat`	Copies all files in the `myworkspace/out` directory into the `webtogo.odb` file.
`crserver.bat`	Copies all files in the `myworkspace/webtogo` directory to the Mobile Server Repository, including the `webtogo.odb` and `webtogo.ora` files.

6.2.10 Using the Mobile Server Admin API

The Mobile Server Admin API enables an administrator to manage the application resources programmatically. Using the Mobile Server Admin API set, administrators can potentially create their own customized Mobile Manager application to perform the following functions.

- Creating and modifying users and user groups
- Including users and excluding users from group level access to applications
- Assigning snapshot variables to the user
- Suspending and resuming applications
- Publishing a pre-packaged Web-to-Go application
- Customizing an application's underlying database connections

For more information on using the API to build the Mobile Manager, see the *Oracle Database Lite API Specification*, which you can link to off the following Web page:

`<ORACLE_HOME>\mobile\index.htm`

Note: Administrators cannot use the open API set to change the basic properties of an application, such as snapshot definitions or servlets. This can only be done through the Packaging Wizard. For more information, see Chapter 6, "Using the Packaging Wizard" in the *Oracle Database Lite Developer's Guide*.

7 Using the Packaging Wizard

The following sections enable you to package and publish your Mobile application definitions using the Packaging Wizard.

- Section 7.1, "Using the Packaging Wizard"
- Section 7.2, "Packaging Wizard Synchronization Support"

7.1 Using the Packaging Wizard

After you have completed the code implementation for your application, you need to define the SQL commands that retrieve the data for the user snapshot—also known as a publication. MDW (as described in Chapter 5, "Using Mobile Database Workbench to Create Publications") is a graphical tool that enables you to define the publications for your application. Then, use the Packaging Wizard to package the application and publish the final application product to the Mobile Server to complete the subscription.

In general, you can create a publication—or components of a publication—using the following methods:

- SQL on the back-end Oracle database
- Consolidator APIs
- MDW
- Packaging Wizard
- mSQL on the Mobile Client against the Oracle Lite database

If you create the publication using any method other than the Packaging Wizard, you can import the definition into the Packaging Wizard. However, these tools and the Packaging Wizard are separate. Thus, once the publication is published by the Packaging Wizard, you can only modify it through the Packaging Wizard.

Important: If you modify the publication or any component of the publication using any method other than the Packaging Wizard, then it will not show up in your published application.

The following is the recommended method for creating the publication for the application:

- Create a new Mobile application definition—An application definition is more than the code that you have implemented. It consists of the implementation, the publication with its publication items, and other components. Use the Mobile Database Workbence (MDW) tool (as described in Chapter 5, "Using Mobile

Database Workbench to Create Publications" for performing an iterative approach to defining your publications.

- Edit an existing Mobile application definition within the Packaging Wizard—You can always go back and edit an existing Mobile application definition for tuning purposes, to modify the publication, or other reasons.
- Package a Mobile application definition for easy deployment within the Packaging Wizard—Once the application is finished with development, you need to package the components into either a WAR or JAR file before you can publish the application definition.
- Publish an application definition to the Mobile Server—You can either publish your application definition to the Mobile Server with the Packaging Wizard or through the Mobile Manager.

The following sections describe how to use the Packaging Wizard tool:

- Section 7.1.1, "Starting the Packaging Wizard"
- Section 7.1.2, "Specifying New Application Definition Details"
- Section 7.1.3, "Listing Application Files"
- Section 7.1.4, "Adding Servlets (For OC4J and Web-to-Go Applications Only)"
- Section 7.1.5, "Entering Database Information"
- Section 7.1.6, "Defining Application Roles"
- Section 7.1.7, "Defining Snapshots for Replication"
- Section 7.1.8, "Defining Sequences for Replication"
- Section 7.1.9, "Defining Application DDLs"
- Section 7.1.10, "Editing Application Definition"
- Section 7.1.11, "Troubleshooting"

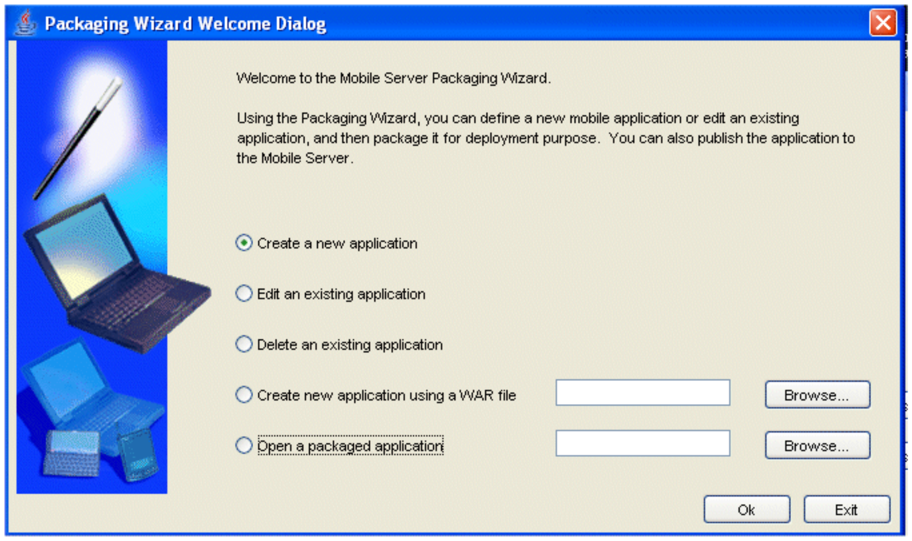
7.1.1 Starting the Packaging Wizard

To launch the Packaging Wizard, enter the following using a Command Prompt window.

```
runwtgpack
```

Figure 7–1 shows the Welcome screen for the Packaging Wizard, which enables you to create, edit, or remove the Mobile application definition as described fully in Table 7–1.

Figure 7–1 Packaging Wizard - Make A Selection Dialog


Packaging Wizard Welcome Dialog

Welcome to the Mobile Server Packaging Wizard.

Using the Packaging Wizard, you can define a new mobile application or edit an existing application, and then package it for deployment purpose. You can also publish the application to the Mobile Server.

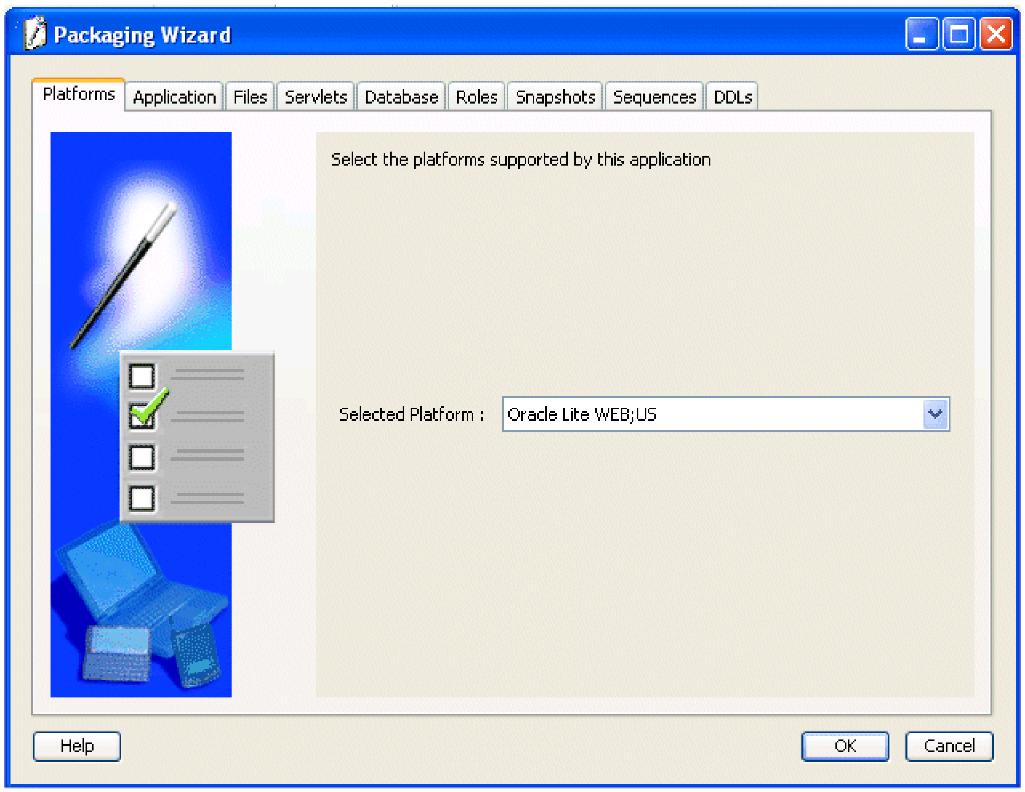
[✓ Create a new application]
Edit an existing application
Delete an existing application
Create new application using a WAR file
Open a packaged application

Table 7–1 Make a Selection Dialog

Feature	Description
Create a new application definition	Define a new Mobile application definition with the application implementation, publication items, and so on.
Edit an existing application definition	Edit an existing Mobile application definition. When selected, all existing application definitions are presented in a drop-down box. Users can select the desired Mobile application definition from the list. All applications listed in this list have been created or published using the Packaging Wizard. Any application definition created by MDW will not appear in this list.
Remove an existing application definition	Remove an existing Mobile application definition. When selected, all existing application definitions are presented in a drop-down box. Users can select the desired Mobile application definition from the list. This option removes the application definition from the Packaging Wizard; it does not delete the application from within the Mobile Server.
Creating a new application definition using a WAR file	Create an application definition using a Web Application Archive (WAR) file. You can enter the name of the WAR file or locate it using the 'Browse' button.
Open a Packaged application definition	Select an application definition that has been packaged a JAR file. You can enter the name of the packaged application or locate it using the 'Browse' button.

Using the 'Select a Platform' dialog, select the platform for which you want to package your application definition. As Figure 7–2 displays, this dialog enables you to specify a platform. If you are packaging a WAR file, this dialog only displays Web based platforms.

Figure 7–2 Select a Platform Dialog



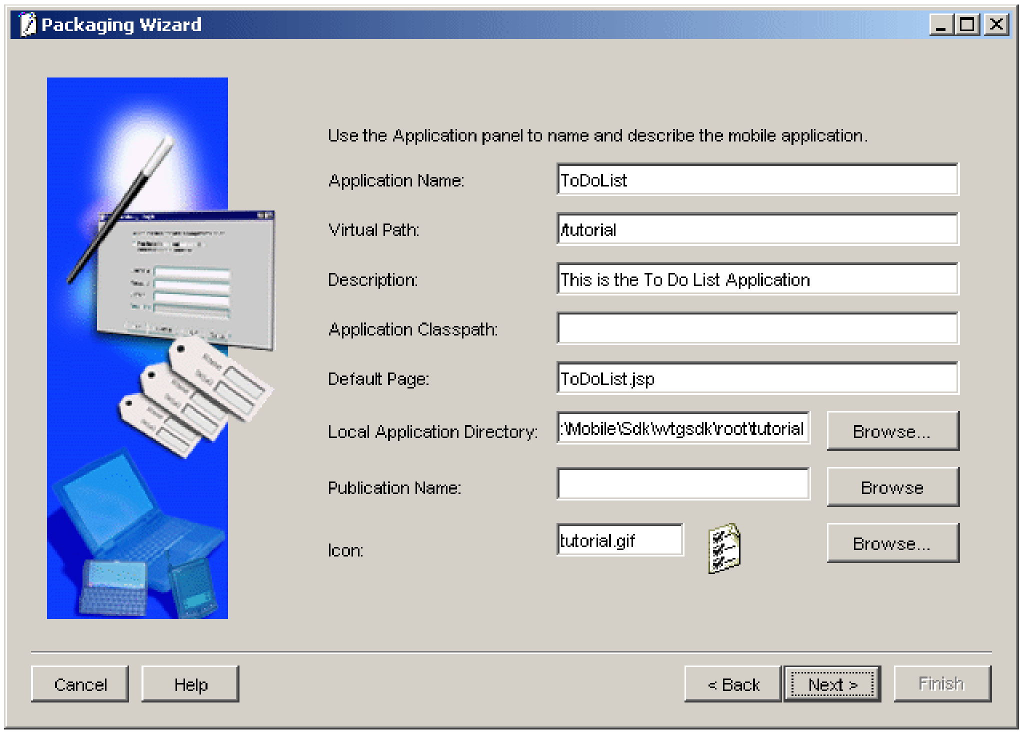
Select the platforms supported by this application

Selected Platform : Oracle Lite WEB;US

7.1.2 Specifying New Application Definition Details

Using the Application dialog, you can name a new application and specify its storage location on the Mobile Server. As Figure 7–3 displays, the Application dialog includes the following fields.

Figure 7–3 Application Dialog



Packaging Wizard

Use the Application panel to name and describe the mobile application.

Application Name:	ToDoList
Virtual Path:	/tutorial
Description:	This is the To Do List Application
Application Classpath:	
Default Page:	ToDoList.jsp
Local Application Directory:	:\Mobile\Sdk\wtgsdk\root\tutorial
Publication Name:	
Icon:	tutorial.gif

Table 7–2 describes the Application dialog.

Table 7–2 Application Dialog Description

Field Name	Description	Required
Application Name	The name of the new Mobile application definition. When packaging a WAR file, the application name must be set to the value of the element `<display-name>`, which can be found under the main element `<web-app>` in the file `web.xml`.	Yes

Table 7–2 (Cont.) Application Dialog Description

Field Name	Description	Required
Virtual Path	A path that is mapped from the root directory of the server repository to the Mobile application itself. The virtual path eliminates the need to refer to the application entire directory structure. It indicates that all of the subdirectories and all of the files that are in the virtual path will be uploaded exactly as they are in the directory structure to the Mobile Server Repository when the application is published. It also provides the application with a unique identity. **Application Root Directory** As Figure 7–3 displays, the name `/tutorial` indicates the virtual path of the application. The name that you enter as the virtual path of the application becomes the **application root directory** within the Mobile Server Repository, when the application is published. Consequently, you can specify the application root directory by the name that you enter in the virtual path field. This name can be different from the application name, but should not contain spaces. For example, your application name can be 'Sales Office' and your virtual path '/Admin'. In this case, '/Admin' becomes the name of the application root directory within the Mobile Server Repository. The application root directory is the location where the actual application files are stored within the Mobile Server Repository. When the administrator publishes the application, the Packaging Wizard automatically uses the name that you entered in the virtual path as the name of the application root directory in the Mobile Server Repository. However, the administrator can change the name of the application root directory in the Mobile Server Repository by entering a different name for it when the administrator publishes the application.	Yes
Description	A brief description of the Mobile application. When packaging a WAR file, the description must be set to the value of the element `<description>` found under the main element `<web-app>` in the `web.xml` file.	Yes

Table 7–2 (Cont.) Application Dialog Description

Field Name	Description	Required
Application Classpath [OC4J and Web-to-Go Applications Only]	The application classpath specifies where the classes (servlets, beans) for the application are located. The default application classpath is always the application root directory. To specify additional locations that the Mobile Server can search for application classes, add other directories or JAR and ZIP files to the application classpath for Web applications. Entries must be separated by semicolons (;) In addition, Web-to-Go automatically appends the following to the application classpath: **1.** Application root directory **2.** Classpath as specified in the 'Application' dialog in the Packaging Wizard **3.** Classes located under `WEB-INF/classes` **4.** All JAR and ZIP files located in the directory `WEB-INF/lib` **5.** Classes located under the directory `/shared/WEB-INF/classes` **6.** All jar and zip files located in the directory `/shared/WEB-INF/lib` **7.** `SYSTEM` classpath	No
Default Page [Web Applications Only]	The server location of the Web page that functions as the Mobile application's entry point. This is a relative path to the repository directory. For example, if the server directory is `/apps` and the default page is `index.htm`, the Default Page is `/apps/index.htm`. The default page can be a servlet. A generic page is issued if the user does not specify a default page. When packaging a WAR file, the default page must be set to the value of the element `<welcome-file-list>` in the `web.xml` file.	Yes
Local Application Directory	The directory on the local machine that contains all components of the application. You can type this location or locate it using the 'Browse' button. During development, the application root directory is set to the local application directory.	Yes
Icon [Web Applications Only]	The GIF image of the Mobile application is used as the application icon in the Mobile workspace. Users may enter the icon name in the corresponding field or locate it using the 'Browse' button. When packaging a WAR file, the description field must be set to the value of the element `<large-icon>` as a primary choice or `<small-icon>` as a secondary choice found under the main element `<web-app>` in the `web.xml` file.	
Publication Name	Publication name of an existing application in the Mobile Server repository. You can enter the publication name or locate it using the Browse button.	No

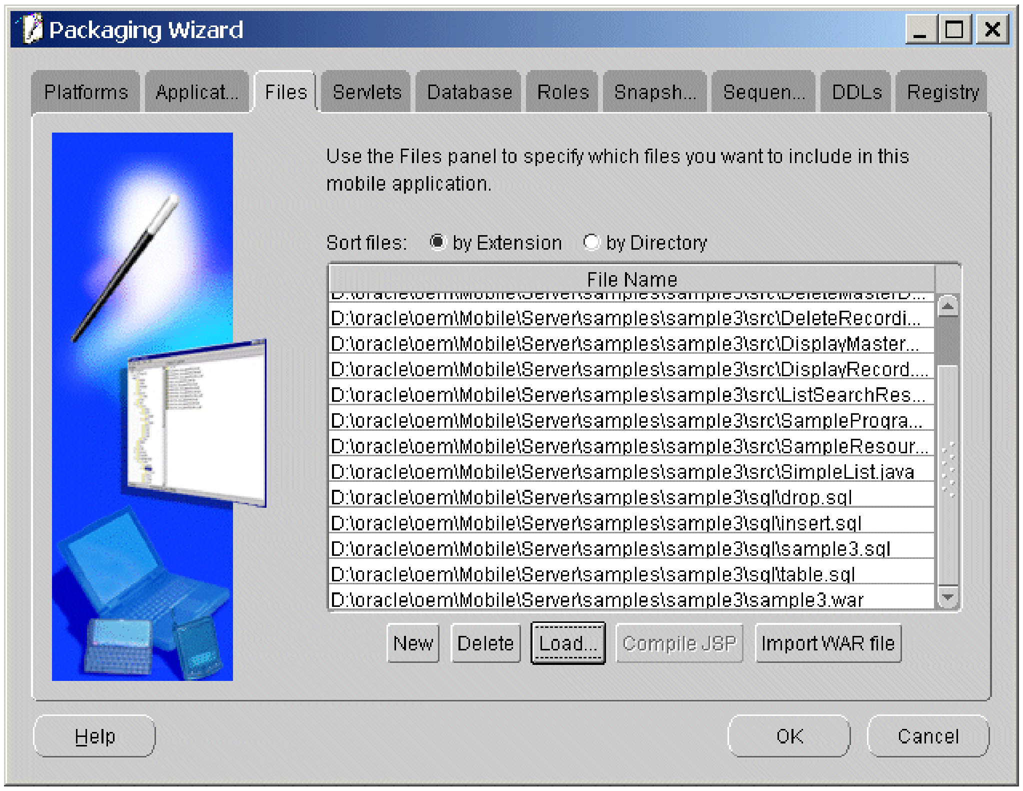
7.1.3 Listing Application Files

Use the Files panel to list your application files and to specify their location on the local machine. The Packaging Wizard analyzes the contents of the Local Application

Directory and displays each file's local path. As Table 7–3 describes, the Files tab contains the following field.

Figure 7–4 displays the Files tab.

Figure 7–4 Files Tab



Use the Files panel to specify which files you want to include in this mobile application.

Sort files: ● by Extension ○ by Directory

File Name
D:\oracle\oem\Mobile\Server\samples\sample3\src\DeleteRecordi...
D:\oracle\oem\Mobile\Server\samples\sample3\src\DisplayMaster...
D:\oracle\oem\Mobile\Server\samples\sample3\src\DisplayRecord....
D:\oracle\oem\Mobile\Server\samples\sample3\src\ListSearchRes...
D:\oracle\oem\Mobile\Server\samples\sample3\src\SampleProgra...
D:\oracle\oem\Mobile\Server\samples\sample3\src\SampleResour...
D:\oracle\oem\Mobile\Server\samples\sample3\src\SimpleList.java
D:\oracle\oem\Mobile\Server\samples\sample3\sql\drop.sql
D:\oracle\oem\Mobile\Server\samples\sample3\sql\insert.sql
D:\oracle\oem\Mobile\Server\samples\sample3\sql\sample3.sql
D:\oracle\oem\Mobile\Server\samples\sample3\sql\table.sql
D:\oracle\oem\Mobile\Server\samples\sample3\sample3.war

Table 7–3 Files Tab Description

Field	Description	Required
Local Path	The absolute path of each Mobile application file. Each entry on the list includes the complete path of the individual file or directory.	Yes

You can add, remove, load, or compile any of the files that are listed in the 'Files' dialog. If you are creating a new application, the Packaging Wizard automatically analyzes and loads all files listed under the local directory when you proceed to the 'Files' dialog. If you are editing an existing application, upload your individual application files using the 'Load' button.

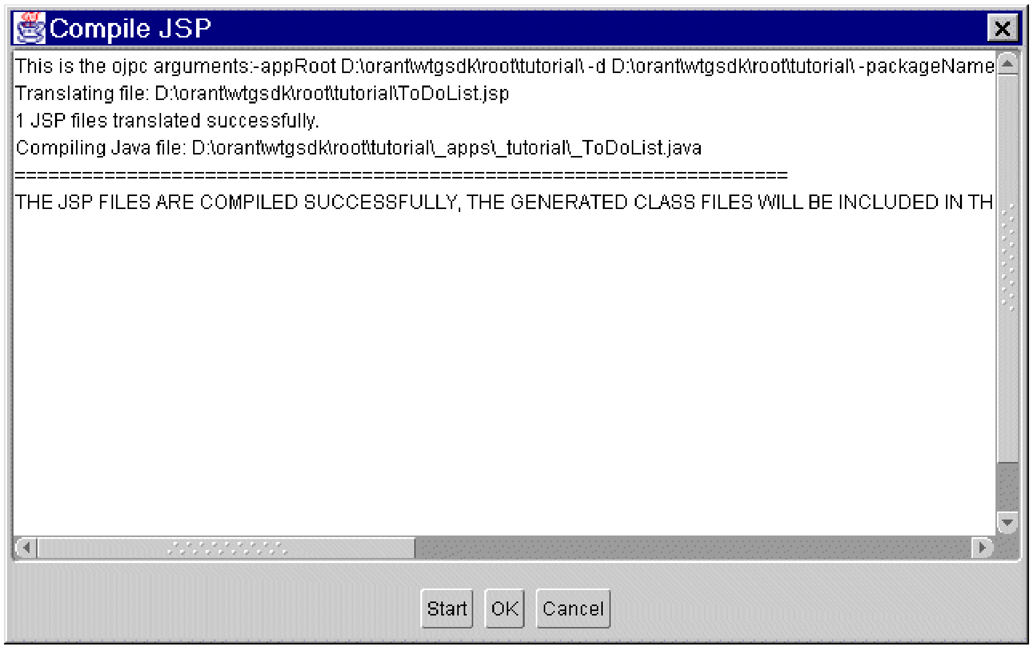
If you are importing a WAR file into an existing application, click the **Import WAR File** button on the 'Files' tab. Once you have specified the location of the WAR file, the 'Files' tab displays content of the WAR file.

7.1.3.1 Compile JSP (For Web-to-Go Applications Only)

The 'Compile JSP' button enables you to compile your JSP files for deployment. If you click the 'Compile JSP' button, the following 'Compile JSP' dialog appears with detailed compilation information. If there are any errors, you should correct the JSP files before proceeding.

Figure 7–5 displays the Compile JSP Dialog.

Figure 7–5 Compile JSP Dialog



```
This is the ojpc arguments:-appRoot D:\orant\wtgsdk\root\tutorial\ -d D:\orant\wtgsdk\root\tutorial\ -packageName
Translating file: D:\orant\wtgsdk\root\tutorial\ToDoList.jsp
1 JSP files translated successfully.
Compiling Java file: D:\orant\wtgsdk\root\tutorial\_apps\_tutorial\_ToDoList.java
=====================================================================
THE JSP FILES ARE COMPILED SUCCESSFULLY, THE GENERATED CLASS FILES WILL BE INCLUDED IN TH
```

You can sort the files by their extensions or by the directory in which they are located. To sort files, click the 'By Extension' or 'By Directory' options.

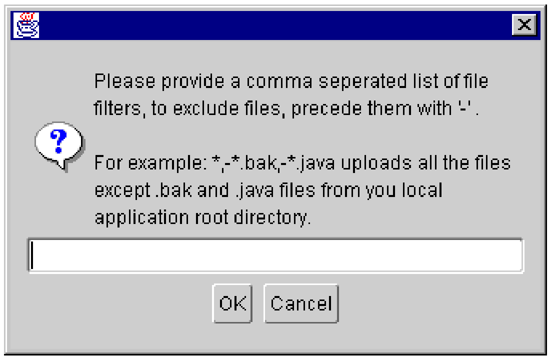
7.1.3.2 Filters

When you click the 'Load' button, the 'Input' dialog appears. You can use the 'Input' dialog to create a comma-separated list of filters that either include or exclude application files from the upload process. To exclude a file, type a preceding minus sign (-) before the file name. For example, to load all files but exclude files with the `.bak` and `.java` suffixes, enter the following.

```
*,-*.bak,-*.java
```

Figure 7–6 displays the Input dialog.

Figure 7–6 Input Dialog



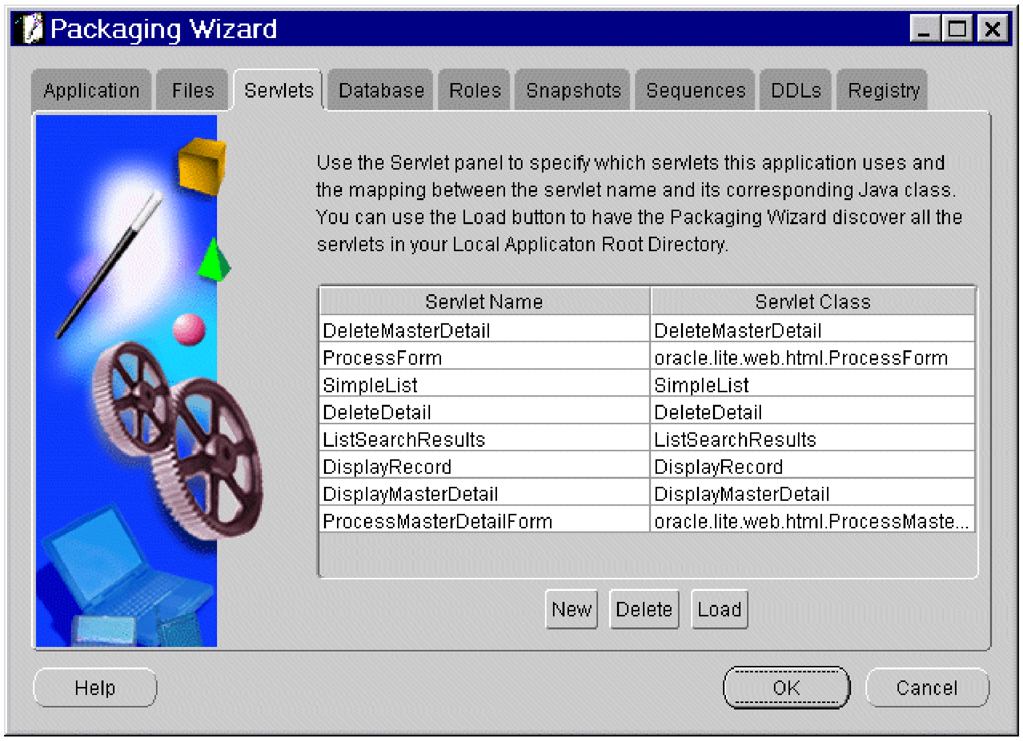
Please provide a comma seperated list of file filters, to exclude files, precede them with '-'.

For example: *,-*.bak,-*.java uploads all the files except .bak and .java files from you local application root directory.

7.1.4 Adding Servlets (For OC4J and Web-to-Go Applications Only)

The Packaging Wizard analyzes servlets in the File tab and defines them on the Mobile Server. As displayed in Figure 7–7, you can view your application's servlets in the Servlets tab.

Figure 7–7 Servlets Tab



Use the Servlet panel to specify which servlets this application uses and the mapping between the servlet name and its corresponding Java class. You can use the Load button to have the Packaging Wizard discover all the servlets in your Local Applicaton Root Directory.

Servlet Name	Servlet Class
DeleteMasterDetail	DeleteMasterDetail
ProcessForm	oracle.lite.web.html.ProcessForm
SimpleList	SimpleList
DeleteDetail	DeleteDetail
ListSearchResults	ListSearchResults
DisplayRecord	DisplayRecord
DisplayMasterDetail	DisplayMasterDetail
ProcessMasterDetailForm	oracle.lite.web.html.ProcessMaste...

As described in Table 7–4, the 'Servlets' tab includes the following fields.

Table 7–4 Servlets Tab Description

Field	Description	Required
Servlet Name	The servlet's name. For example: `DeleteDetail`. You will then refer the servlet as: `application_virtualpath/servlet name`	Yes
Servlet Class	The fully qualified class of the servlets to be added.	Yes

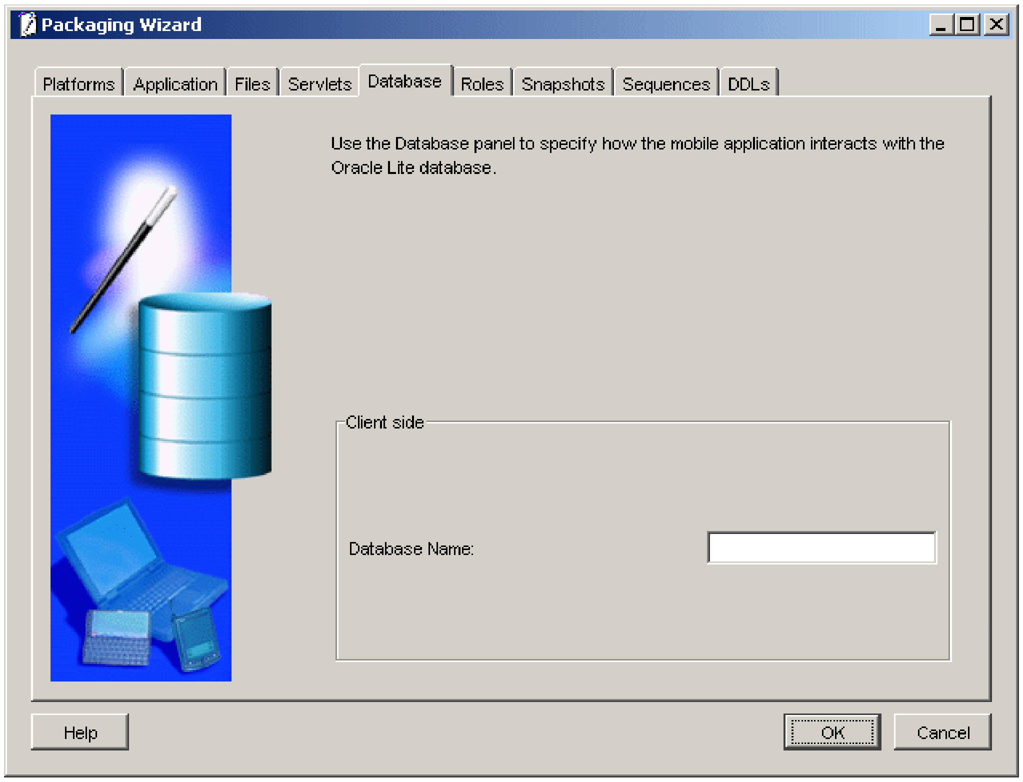
Using the 'Servlets' tab, you can add, remove, or load any servlets that are listed under the 'Servlets' tab. If you are creating a new application, the Packaging Wizard automatically lists all 'Servlets' based on files that are listed in the 'Files' tab. If you are editing an existing application, use the 'Load' button to locate and load individual servlets.

7.1.5 Entering Database Information

Using the Database tab, you can provide connection information and specify how the Mobile application user connects to the replication master groups on the Oracle server.

Figure 7–8 displays the Database tab.

Figure 7–8 Database Tab



Use the Database panel to specify how the mobile application interacts with the Oracle Lite database.

Client side

Database Name:

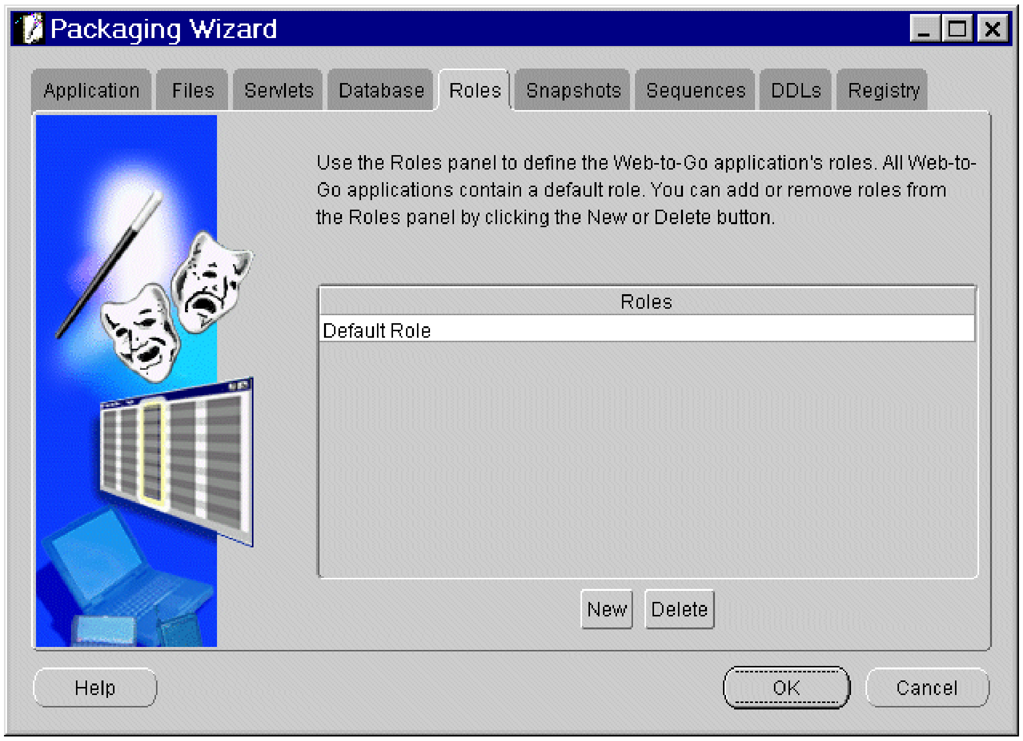
Enter the database name that you want to create on the client side. For example, a native Windows 32 application accesses the client database with this name. However, this is not required for Web applications.

7.1.6 Defining Application Roles

Use the 'Roles' tab to define the Mobile Server application's roles. Developers create roles in the application's code and the Packaging Wizard re-declares them for the Oracle database. After you publish the application to the Mobile Server, you can assign roles to users and groups, using the Mobile Manager.

Figure 7–9 displays the Roles tab.

Figure 7–9 Roles Tab



Use the Roles panel to define the Web-to-Go application's roles. All Web-to-Go applications contain a default role. You can add or remove roles from the Roles panel by clicking the New or Delete button.

Default Role

As described in Table 7–5, the Roles tab includes the following field.

Table 7–5 Roles Tab Description

Field	Description
Roles	Assigns roles to the Web-to-Go/Mobile Server application.

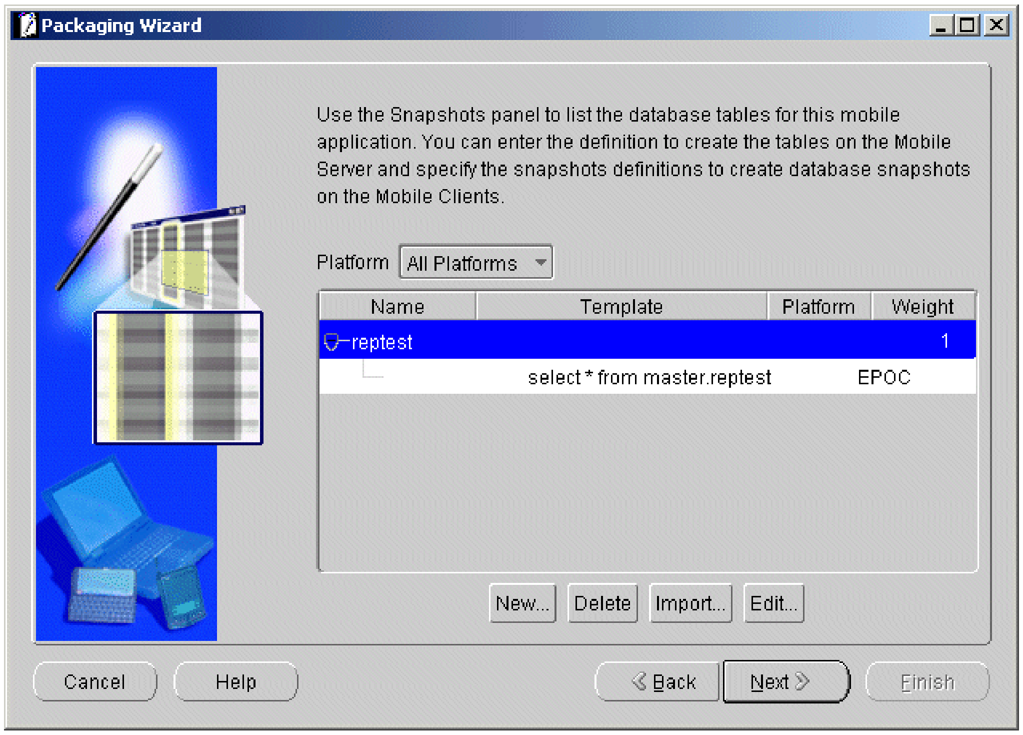
All Web-to-Go/Mobile Server applications contain a default role. You can add or remove roles from the Roles dialog using the 'New' or 'Delete' button.

7.1.7 Defining Snapshots for Replication

If you did not use MDW to create a subscription, then you can use the Snapshots tab to create replication snapshots for your application. A snapshot must have the same name as the database object such as a table or view. It must be unique across all applications. However, you must ensure that you use unique names when creating database objects. The Packaging Wizard enables you to create snapshots for the chosen platform. When you specify a view as the base object type, the Packaging Wizard enables you to specify the Parent Hint, Virtual Primary Hint, and the Primary Key Hint. For Web-to-Go, use the Windows 32 platform.

Figure 7–10 displays the Snapshots tab.

Figure 7–10 Snapshots Tab


Packaging Wizard

Use the Snapshots panel to list the database tables for this mobile application. You can enter the definition to create the tables on the Mobile Server and specify the snapshots definitions to create database snapshots on the Mobile Clients.

Platform: All Platforms

Name	Template	Platform	Weight
reptest			1
	select * from master.reptest	EPOC	

Note: Once you have specified a database connection, it is used for the remainder of your Packaging Wizard session. If you need to switch between an Oracle database and Oracle Database Lite, but have already established a connection, you must quit the Packaging Wizard application completely and run `runwtgpack.bat` again.

Table 7–6 describes the Snapshots tab.

Table 7–6 Snapshots Tab Description

Field	Description	Required
Name	The name(s) of the snapshot(s) associated with the Web-to-Go/Mobile Server application. It must be the same name as the underlining database object.	Yes
Template	Lists available snapshot templates. The template is a SQL statement that is used to create the snapshot. The template may contain variables. After you publish the template to the Mobile Server, you can specify user-specific template variables using the Mobile Manager. However, you cannot modify snapshots in the Mobile Manager.	Yes
Weight	This is the order of tables to be replicated. For tables with a master-detail relationship, the master table needs to be replicated first and therefore should have a lower weight.	No

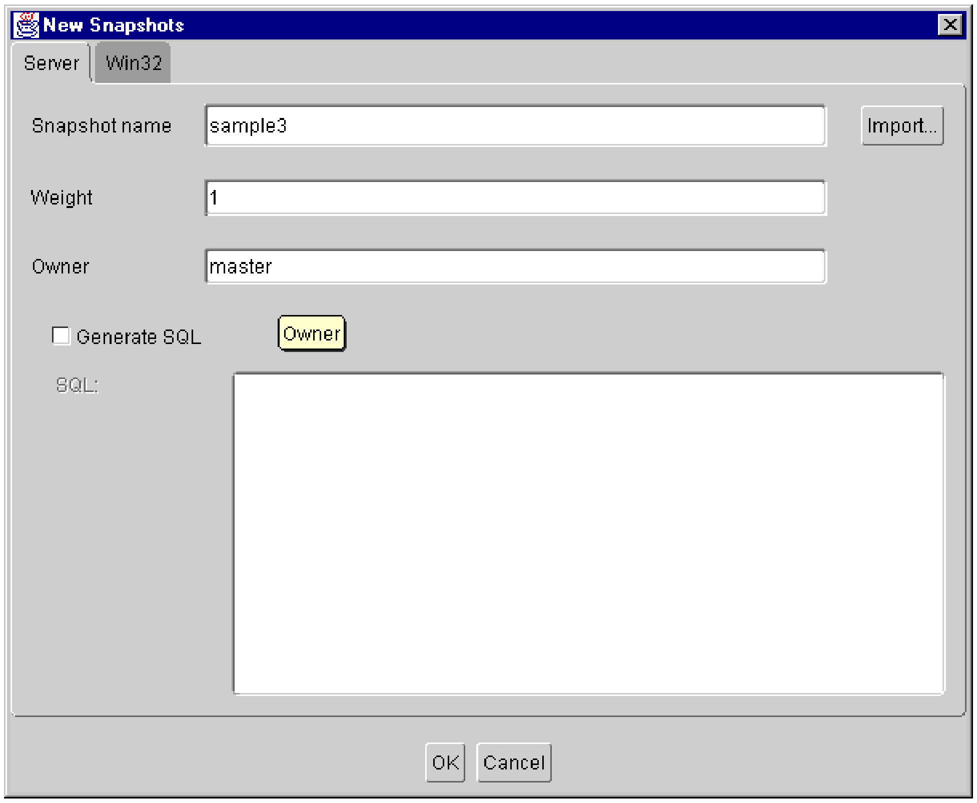
You can add or remove snapshots from the Snapshots tab using the 'New' or 'Delete' button. You can also import or edit snapshots using the 'Import' or 'Edit' button.

Note: You can import multiple snapshots from the Snapshots tab or import one when you create a new table from the 'New Table Dialog'.

7.1.7.1 Creating New Snapshots

To create new snapshots, click 'New'. The 'New Snapshots' dialog appears. As Figure 7–11 displays, if you click the Server tab, the Server dialog appears, which contains fields for snapshot name, weight, owner, and SQL, as well as a check box for generating SQL.

Figure 7–11 New Snapshots Dialog - Server Tab



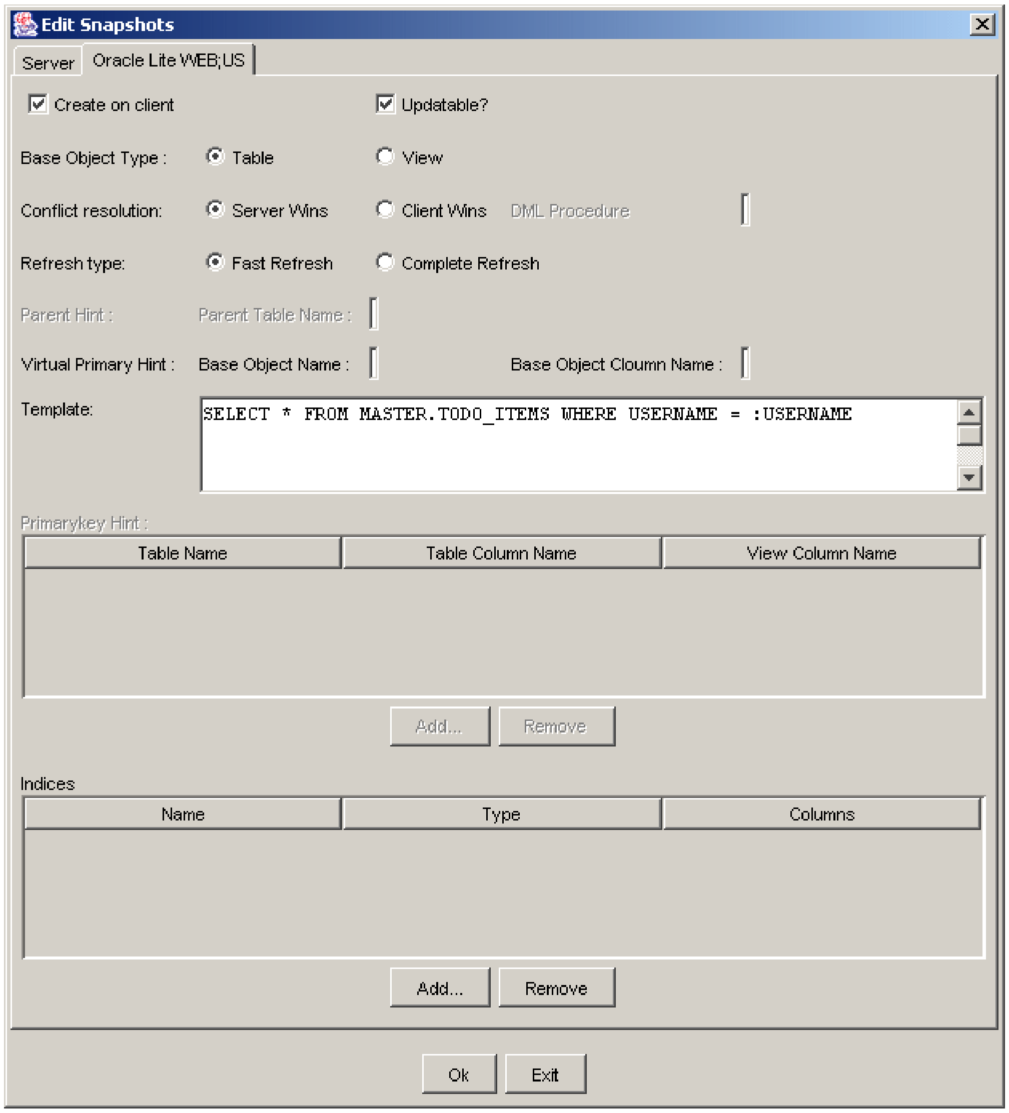
Snapshot name: sample3
Weight: 1
Owner: master
☐ Generate SQL

For a description of Weight, see Section 7.1.7, "Defining Snapshots for Replication".

By default, Generate SQL is enabled, which automatically generates the SQL statement for you. Use the Win32 tab for the Mobile Client for Web-to-Go.

If you click the Win32 tab, the following dialog appears.

Figure 7–12 Edit Snapshots Dialog - Win32 Tab



```
SELECT * FROM MASTER.TODO_ITEMS WHERE USERNAME = :USERNAME
```

Create a new snapshot on the Mobile Client for Web-to-Go by modifying the following features in the New Snapshots dialog.

As Figure 7–7 describes, the New Snapshots dialog displays the following information.

Table 7–7 New Snapshots Dialog Description

Field	**Description**
Updatable	When selected, this check box creates an updatable snapshot of the named table.
Template	Displays the snapshot template for the named table. You can modify the snapshot template. Administrators can instantiate variables for different users to this template using the Mobile Manager. For more information about template variables, see Section 7.1.7, "Defining Snapshots for Replication".

7.1.7.2 Creating Indexes for Snapshots

To create an index for a snapshot using the Packaging Wizard, use the following procedure.

1. From the Snapshots dialog, select the Edit button to create an index from an existing snapshot, or the New button for creating a new snapshot and new index.
2. Select the platform tab on the dialog which appears, for example Win 32. The SQL statement which defines your snapshot appears in the 'Template' field. Below that is an 'Indices' table; to create a new index, select the 'New' button beneath this table.

   As Table 7–8 describes, enter values in the Win32 tab of the Edit Snapshots dialog.

Table 7–8 Win32 Tab - Edit Snapshots Dialog

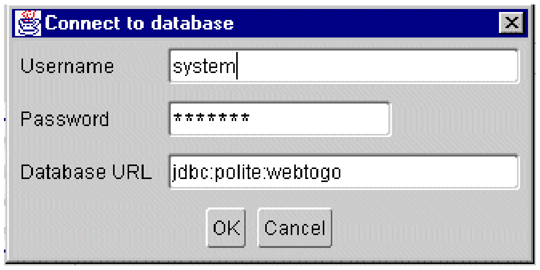
Field	Description
Create on Client	If selected, creates the snapshot on the client machine.
Updatable	If selected, creates an updatable snapshot of the specified table or view.
Base Object Type	Select **Table** to include a table as the base object type. or Select **View** to include a view as the base object type.
Conflict Resolution	Select **Server Wins** to specify conflict resolution in favour of the server. or Select **Client Wins** to specify conflict resolution in favour of the client.
DML Procedure	To specify the DML procedure, enter the name of the Callout Package for DML operation.
Refresh Type	Select **Fast Refresh** to specify a quick refresh of the snapshot. or Select **Complete Refresh** to specify a complete refresh of the snapshot.
Parent Hint	To specify the parent hint, enter the **Parent Table Name**.
Virtual Primary Hint	To specify the virtual primary hint, enter the **Base Object Name** and **Base Object Column** in the corresponding fields.
Template	Displays the snapshot template for the named table. You can modify the snapshot template. Administrators can instantiate variables for different users to this template using the Mobile Manager. For more information about template variables, see Section 7.1.7, "Defining Snapshots for Replication".
Primary Key Hint	This section displays the **table name**, **column name**, and **mapping column name** of the snapshot.
Indices	This section displays the **name**, **type**, and **column name** of indices used in a snapshot.

3. There are three columns in the 'Indices' table:
   a. Name - This is the name of the index.
   b. Type - Indexes can be Regular, Primary, or Unique. There is a drop down menu to select this.
   c. Columns - Enter the column name which the index uses.

7.1.7.3 Importing Snapshots

To import snapshots from an Oracle database or from Oracle Database Lite, click the 'Import' button. As Figure 7–13 describes, the database connection window appears if you have not specified a connection.

Figure 7–13 Connect to Database Dialog



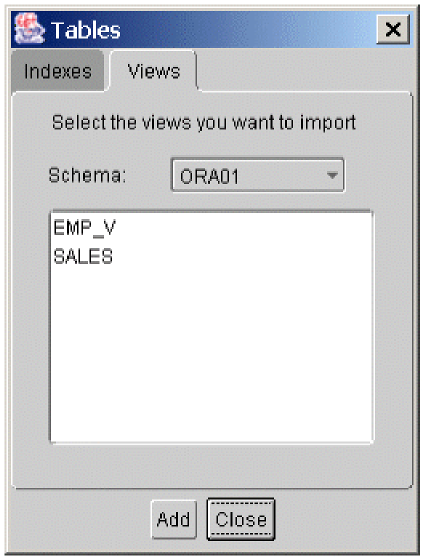
Username: system
Password: *******
Database URL: jdbc:polite:webtogo

Enter the user name, password, and database URL for the Oracle database, or Oracle Database Lite from which you are importing your snapshot(s). The Tables window appears.

Note: Use the following format when entering the database URL for an Oracle database: `jdbc:oracle:thin:@<MOBILESERVER_JDBC_URL>`. For Oracle Database Lite, use `jdbc:polite:webtogo`.

Figure 7–14 displays the Tables dialog.

Figure 7–14 Tables Dialog



Select the views you want to import

Schema: ORA01

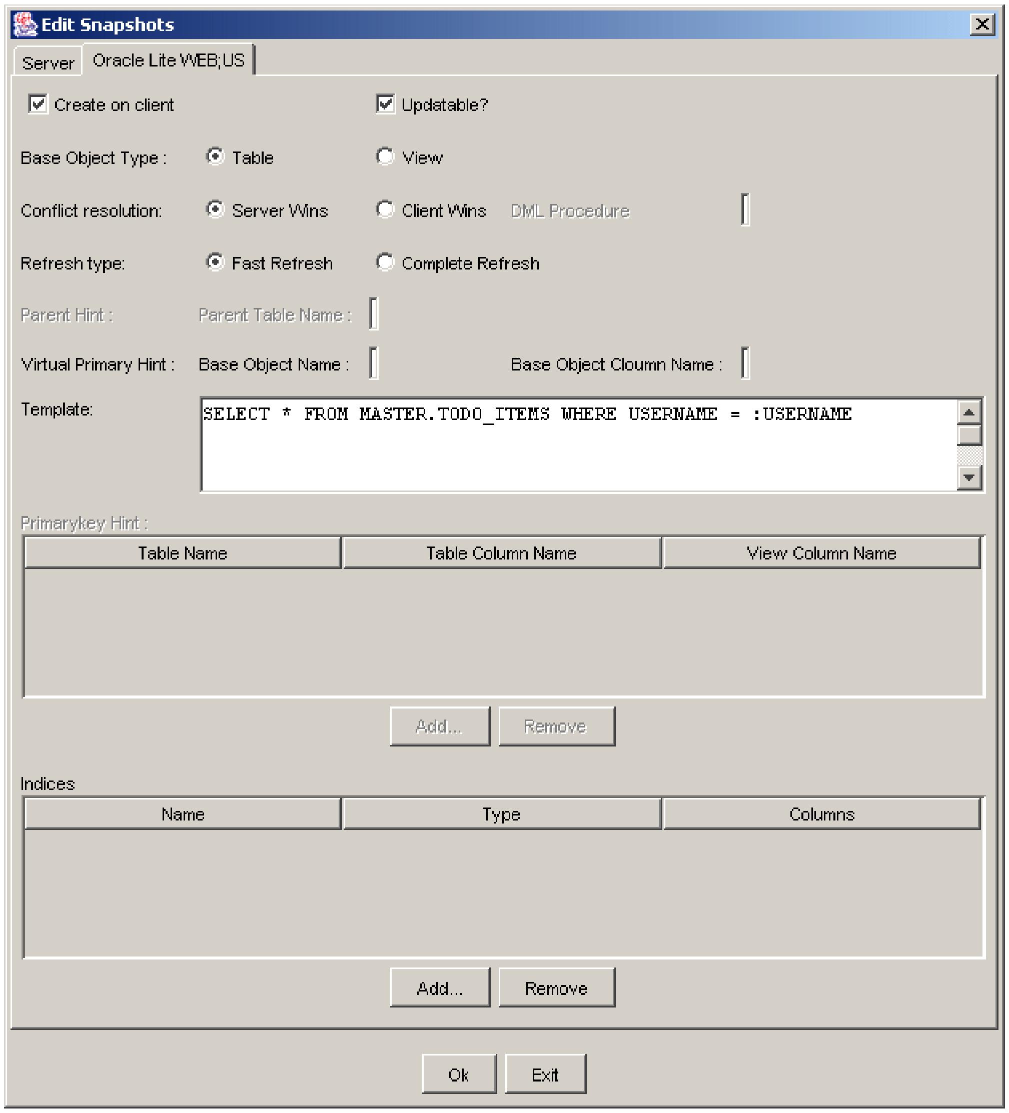
EMP_V
SALES

Click the Schema list and choose the required schema from the list displayed. The Tables dialog displays views associated with the chosen schema. Select the view that you need to import. Click Add and click Close.

7.1.7.4 Editing Snapshots

To edit a snapshot, select the snapshot from the Snapshots dialog and click Edit. As displayed in Figure 7–15, the Edit Snapshots dialog appears.

Figure 7–15   Edit Snapshots Dialog - Win32 Tab


Edit Snapshots

Server | Oracle Lite WEB;US

[✓] Create on client   [✓] Updatable?

Base Object Type : (●) Table ( ) View

Conflict resolution: (●) Server Wins ( ) Client Wins   DML Procedure

Refresh type: (●) Fast Refresh ( ) Complete Refresh

Parent Hint :   Parent Table Name :

Virtual Primary Hint :   Base Object Name :   Base Object Cloumn Name :

Template:

```
SELECT * FROM MASTER.TODO_ITEMS WHERE USERNAME = :USERNAME
```

Primarykey Hint :

Table Name	Table Column Name	View Column Name

Add... Remove

Indices

Name	Type	Columns

Add... Remove

Ok Exit

As described in Table 7–9, edit the snapshot by modifying the following features of the Edit Table window:

Table 7–9   Edit Snapshots Dialog - Win32 Tab Description

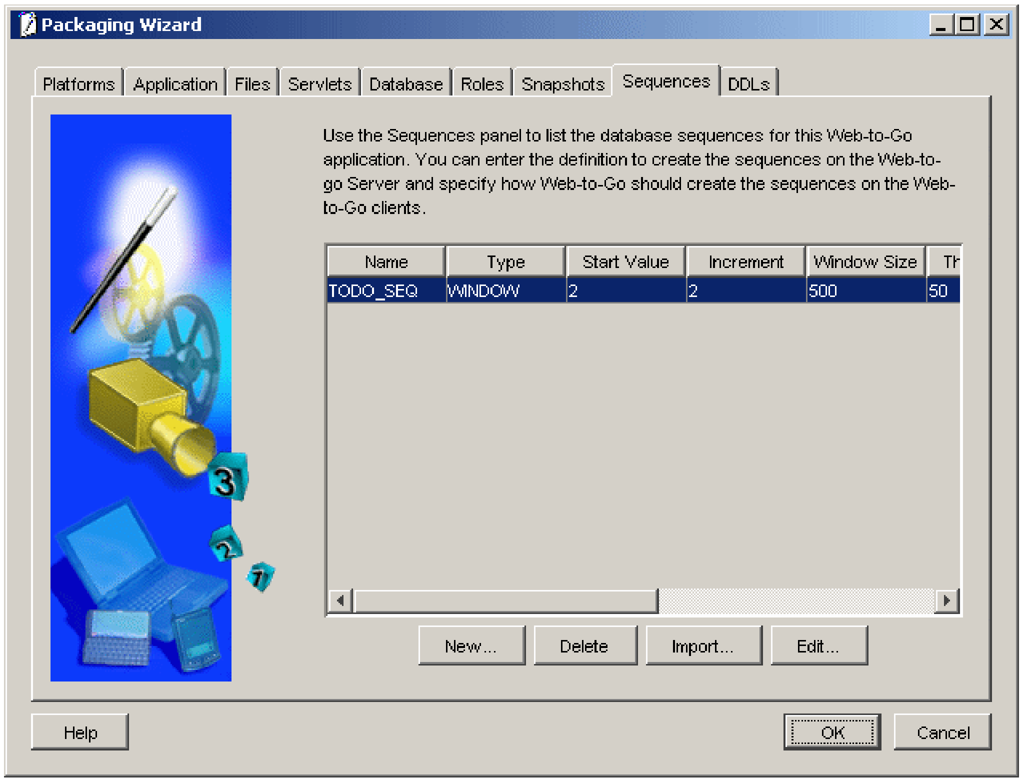
Feature	**Description**
Create on Client	When selected, the checkbox allows you to edit the snapshot on the Mobile Client for Web-to-Go.
Updatable	When selected, this check box creates an updatable snapshot of the named table.
Template	Displays the snapshot template for the named table. You can modify the snapshot template. Administrators can instantiate variables for different users to this template using the Mobile Manager.

7.1.8 Defining Sequences for Replication

Use the Sequences dialog to define sequence support for the Web-to-Go application. Web-to-Go uses sequences to assign unique primary key values to an application before it disconnects from the back-end Oracle database. These unique primary key values are used for replication when the client goes back online. Sequences are important because they eliminate replication conflicts by preventing duplicate primary key values across disconnected applications. All sequences must have a unique name. You can accomplish this by modifying your sequence names by preceding them with your application name.

Figure 7–16 displays the Sequences tab.

Figure 7–16 Sequences Tab



Use the Sequences panel to list the database sequences for this Web-to-Go application. You can enter the definition to create the sequences on the Web-to-go Server and specify how Web-to-Go should create the sequences on the Web-to-Go clients.

Name	Type	Start Value	Increment	Window Size	Th
TODO_SEQ	WINDOW	2	2	500	50

As described in Table 7–10, the Sequences dialog includes the following fields.

Table 7–10 Sequences Dialog Description

Field	Description	Required
Name	The name of the sequence used by the Web-to-Go application in disconnected mode.	Yes
Type	The type of sequence used by the Web-to-Go application in disconnected mode. Window. The window sequence assigns a unique range of values to each client. Window sequences are unique to each client and never overlap with those of other clients. When a client uses all the values in its sequence range, Web-to-Go recreates the sequence with a new, unique range of values the next time the client disconnects from the back-end Oracle database.	Yes

Table 7–10 (Cont.) Sequences Dialog Description

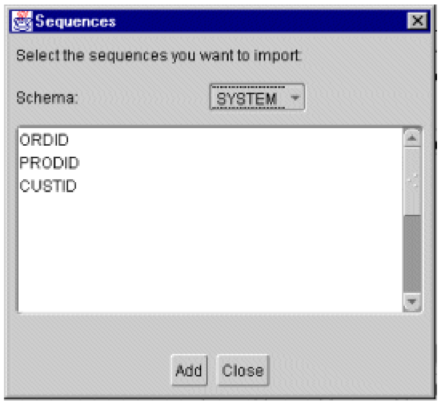
Field	Description	Required
Start Value	The sequence's start value on the Mobile Client for Web-to-Go. The sequence begins at this number and then increments according to the increment number you define.	Yes
Increment	The number by which the sequence increments on the Mobile Client for Web-to-Go, beginning at its start value.	Yes
Window Size	Defines the range of numbers in a window sequence.	Yes
Threshold	Defines the minimum range of required numbers in a window sequence. Web-to-Go creates a new sequence when the existing one reaches this range and when the client disconnects from the back-end database.	Yes
Server Start	The sequence's start value on the Oracle database. The sequence begins at this number and then increments according to the increment number you define. This number must be different from the sequence start value on the Mobile Client for Web-to-Go.	No
Server Increment	The number by which the sequence increments on the Oracle database, beginning at its start value.	No
Server Minimum	The minimum start value for an ascending sequence on the Oracle database. For example, an ascending sequence could start at 1 and continue on in ascending order.	No
Server Maximum	The maximum start value for a descending sequence on the Oracle database. For example, a descending sequence could start at -1 and continue in descending order.	No

You can add or remove sequences from the Sequences dialog by clicking the Add or Remove button.

7.1.8.1 Importing Sequences

To import sequences from an Oracle database, click the Import button. As Figure 7–17 displays, the Sequences dialog appears.

Figure 7–17 Sequences Dialog


Sequences

Select the sequences you want to import:

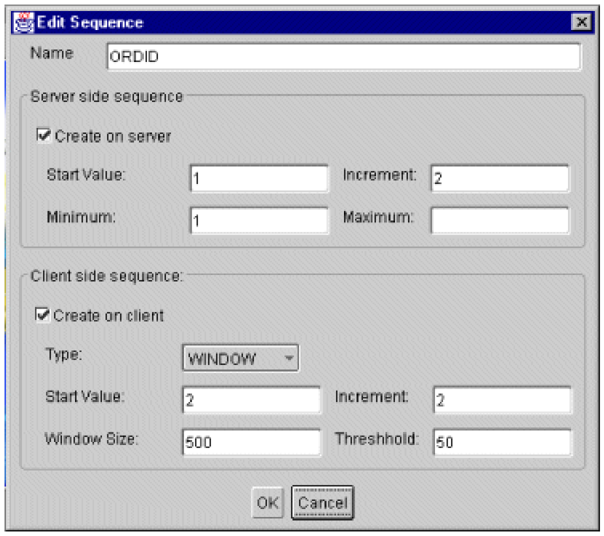
Schema: SYSTEM

ORDID
PRODID
CUSTID

Select the sequence you want to import, click Add, and then click Close.

To edit a sequence, select the sequence from the Sequences dialog and click Edit. As Figure 7–18 displays, the Edit Sequences dialog appears.

Figure 7–18 Edit Sequences Dialog



Name	ORDID
☑ Create on server	
Start Value:	1
Increment:	2
Minimum:	1
Maximum:	
☑ Create on client	
Type:	WINDOW
Start Value:	2
Increment:	2
Window Size:	500
Threshhold:	50

As Table 7–11 describes, edit the sequence by modifying the following features of the Edit Sequences dialog.

Table 7–11 Edit Sequences Dialog Description

Feature	Description
Name	The name of the sequence.
Create on Server	When selected, this check box enables the options for creating a sequence on the Oracle database. Information entered by the user is used to generate a SQL script to create the sequence on the Oracle server.
Start Value	The start value of the sequence on the Oracle database.
Increment	The increment of the sequence on the Oracle database, beginning with its start value.
Minimum	The minimum start value for an ascending sequence on the Oracle database. For example, an ascending sequence could start at 1 and continue in ascending order.
Maximum	The maximum start value for a descending sequence on the Oracle database. For example, a descending sequence could start at -1 and continue in descending order.
Create on Client	When selected, this check box enables the options for creating a sequence on the Mobile Client for Web-to-Go.
Type	Defines the type of sequence on the Mobile Client for Web-to-Go. Options include the window and leapfrog sequences.
Start Value	The sequence start value on the Mobile Client for Web-to-Go.
Increment	The increment of the sequence on the Mobile Client for Web-to-Go, beginning with its start value.
Window Size	The range of numbers that constitute a window sequence on the Mobile Client for Web-to-Go. This information is not used by the leapfrog sequence.

Table 7–11 (Cont.) Edit Sequences Dialog Description

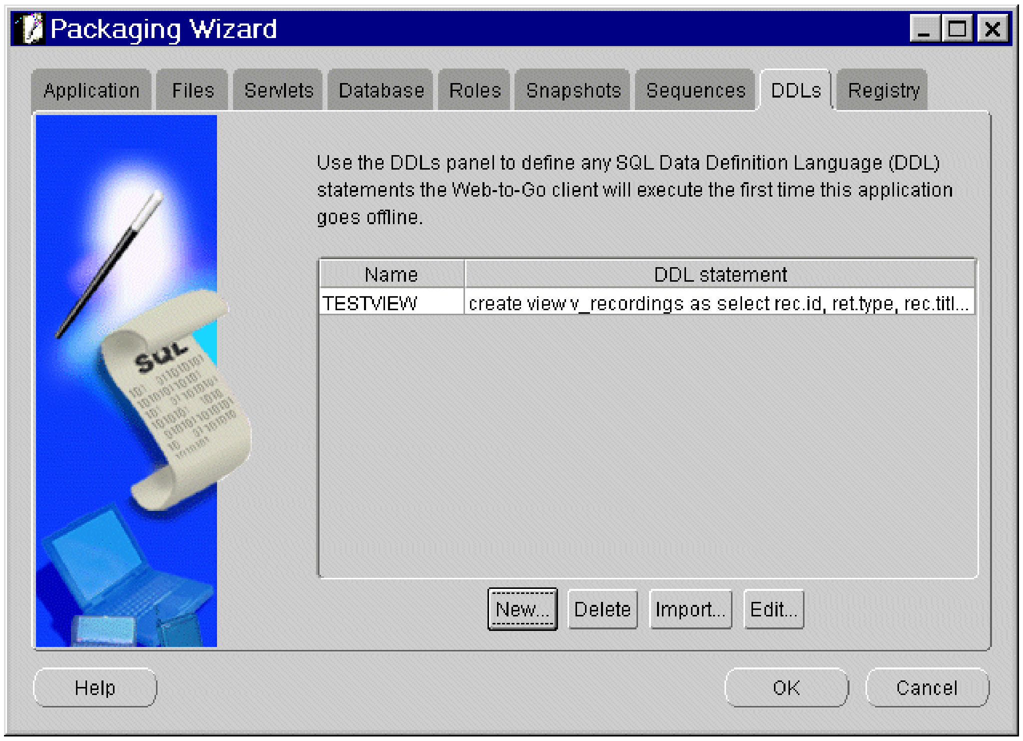
Feature	Description
Threshold	The minimum range of required numbers in a window sequence. Web-to-Go creates a new sequence when the existing one reaches this range and when the client disconnects from the back-end Oracle database. This information is not used by the leapfrog sequence.

7.1.9 Defining Application DDLs

Use the DDLs dialog to define any DDL (Data Definition Language) statements that the Web-to-Go application can execute. DDLs are only supported on Windows 32 and Windows CE platforms. All DDL statements must have a unique name and the weight must be specified for every DDL. One way to accomplish this is to modify your DDL names by preceding them with your application name. After you publish the application to the Mobile Server, you can create additional DDL statements using the Mobile Manager.

Figure 7–19 displays the DDLs dialog.

Figure 7–19 DDLs Dialog



Use the DDLs panel to define any SQL Data Definition Language (DDL) statements the Web-to-Go client will execute the first time this application goes offline.

Name	DDL statement
TESTVIEW	create view v_recordings as select rec.id, ret.type, rec.titl...

As described in Table 7–12, the DDLs dialog includes the following fields.

Table 7–12 DDLs Dialog Description

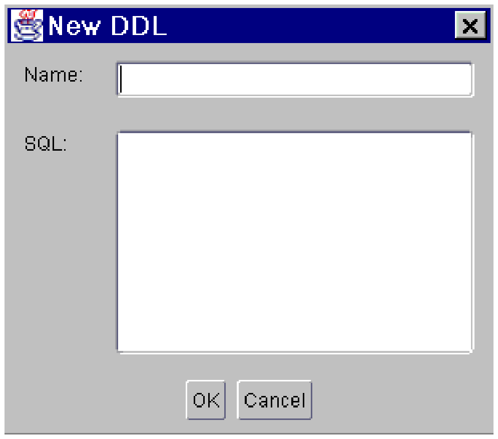
Field	Description
Name	The DDL name.
DDL Statement	Defines DDL statements with the Web-to-Go application. These DDL statements will be executed when the Web-to-Go application runs on the client.

Table 7–12 (Cont.) DDLs Dialog Description

Field	Description
Weight	The order of DDLs to be executed on the Mobile Client.

You can add or remove DDLs from the DDLs dialog by clicking the Add or Remove button. When you click the ADD button, the New DDL dialog appears, as described in Figure 7–20.

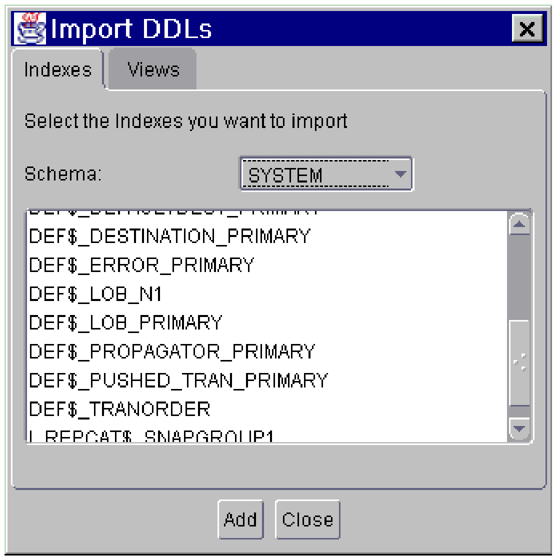
Figure 7–20 New DDL Dialog



7.1.9.1 Importing Views and Index Definitions

To import views and index definitions from an Oracle database, click the Import button. As displayed in Table 7–21, the Import DDLs dialog appears.

Figure 7–21 Import DDLs Dialog



Select the Indexes you want to import

Schema: SYSTEM

DEF$_DESTINATION_PRIMARY
DEF$_ERROR_PRIMARY
DEF$_LOB_N1
DEF$_LOB_PRIMARY
DEF$_PROPAGATOR_PRIMARY
DEF$_PUSHED_TRAN_PRIMARY
DEF$_TRANORDER

To import an index definition, click the Indexes tab and then click the schema from which you want to import an index. Select the index you want to import, click Add, and then click Close.

To import a view definition, click the Views tab and then click the schema from which you want to import a view. Select the view you want to import, click Add, and then click Close.

7.1.10 Editing Application Definition

You can edit application definitions by launching the Packaging Wizard and selecting "Edit an existing application definition."

7.1.11 Troubleshooting

The Packaging Wizard also supports development mode. In this mode, the Packaging Wizard only enables you to define Web application information, list the application files, compile JSPs, add servlets, and make registry changes. Since the application is packaged to your local machine, it requires neither connectivity nor database information.

To launch the Packaging Wizard in development mode, enter the following using the Command Prompt.

```
runwtgpack -d
```

7.2 Packaging Wizard Synchronization Support

The Packaging Wizard and the Mobile Manager provide the ability to perform the most commonly used functions of the publish and subscribe model, package and publish applications, create or drop users, and create or drop subscriptions. More sophisticated functionality is provided by the Consolidator Manager and Resource Manager APIs. Table 7–13 describes basic features.

Table 7–13 Packaging Wizard Synchronization Support

Function	Packaging Wizard	Mobile Manager	API
Open Connection	No	No	Yes
Create User	No	Yes	Yes
Drop User	No	Yes	Yes
Create Publication	Yes	No	Yes
Create Publication Item	Yes	No	Yes
Create Publication Item Index	Yes	No	Yes
Drop Publication	No	Yes	Yes
Drop Publication Item	Special - See the Packaging Wizard documentation for more details.	No	Yes
Drop Publication Item Index	Yes	No	Yes
Create Sequence	Yes	No	Yes
Create Sequence Partition	Yes	No	Yes
Drop Sequence	Yes	No	Yes
Drop Sequence Partition	Yes	No	Yes
Add Publication Item	Yes	No	Yes
Remove Publication Item	No	No	Yes

Table 7–13 (Cont.) Packaging Wizard Synchronization Support

Function	Packaging Wizard	Mobile Manager	API
Create Subscription	No	Yes	Yes
Deinstantiate Subscription	No	No	Yes
Set Subscription Parameter	No	Yes	Yes
Drop Subscription	No	Yes	Yes
Commit Transaction	No	No	Yes
Rollback Transaction	No	No	Yes
Close Connection	No	No	Yes

More advanced features of Data Synchronization are only generally available by using the Consolidator Manager and Resource Manager APIs. Table 7–14 describes these features.

Table 7–14 Data Synchronization Advanced Function Description

Function	Packaging Wizard	Mobile Manager	API
Create Virtual Primary Key Column	Yes	No	Yes
Drop Virtual Primary Key Column	Yes	No	Yes
Add Mobile DML Procedure	Yes	No	Yes
Remove Mobile DML Procedure	Yes	No	Yes
Reinstantiate Publication Item	No	No	Yes
Parent Hint	Yes	No	Yes
Dependency Hint	Yes	No	Yes
Remove Dependency Hint	Yes	No	Yes
Enable Publication Item Query Cache	No	No	Yes
Disable Publication Item Query Cache	No	No	Yes
Primary Key Hint	Yes	No	Yes
Purge Transaction	No	No	Yes
Execute Transaction	No	No	Yes
Complete Refresh	Yes	Yes	Yes
Execute Statement	No	No	Yes
Generate Metadata	No	No	Yes
Reset Cache	No	No	Yes
Cache Dependencies	No	No	Yes
Remove Cache Dependencies	No	No	Yes
Get Current Time	No	No	Yes
Authenticate	No	Yes	Yes
Set Restricting Predicate	No	No	Yes
Alter Publication	Yes	No	Yes

8 Native Application Development

This document discusses Mobile application development for native platforms. The discussion covers the following topics:

- Section 8.1, "Supported APIs for Oracle Database Lite"
- Section 8.2, "Data Source Name"

8.1 Supported APIs for Oracle Database Lite

The following lists the supported APIs for Oracle Database Lite:

Table 8–1 Supported Native APIs

Native API	Description
C, C++, C#	Can use ODBC to access database. Use Oracle-specific APIs for programmatic synchronization. See Section 4.1, "Synchronization APIs For C or C++ Applications" for more information.

In addition, you can use the following APIs for accessing database.

Table 8–2 Supported APIs

Native API	Description
JDBC	Use JDBC to access the database. See Oracle Database JDBC manuals and Chapter 10, "JDBC Programming" for instructions on how to use this API.
ODBC	Use ODBC to access the database. See Microsoft ODBC manuals for instructions on how to use this API.
.NET environment	Use the ADO.NET API. You can use Oracle-specific APIs for connecting to the database, programmatic synchronization, and other functions. See Section 13.1, "Discussion of the Classes That Support the ADO.NET Provider" for more information.
SODA	See Chapter 12, "Using Simple Object Data Access (SODA) for PocketPC Platforms" for more information.
Visual Basic	Use ODBC to access database.
.NET environment	Use the ADO.NET API. You can use Oracle-specific APIs for connecting to the database, programmatic synchronization, and other functions. See Section 13.1, "Discussion of the Classes That Support the ADO.NET Provider" for more information.

Table 8–2 (Cont.) Supported APIs

Native API	Description
Java	There are several specifications for Java applications. See Chapter 9, "Java Application Development" for the Java application support.

8.2 Data Source Name

When you create a data source name using the ODBC Manager, you should use the following conventions:

- In Windows 32, the data source name is automatically created as `<username_dbname>` after the first synchronization, where both the username and database name are taken from within the publication.
- In Windows CE, the data source name is simply the database name; that is, `<dbname>`.

It is helpful to create a data source name to contain all of the properties of your connection to the database.

9

Java Application Development

The following sections describe how to develop and test Java applications:

- Section 9.1, "Java Support for Applications"
- Section 9.2, "Oracle Database Lite Java Development Environment"
- Section 9.3, "Java Development Tools"

9.1 Java Support for Applications

Table 9–1 lists the Java support provided for each platform in Oracle Database Lite.

Table 9–1 Java Support

Category	Web-to-Go (both Windows and Linux)	Windows 32 Native	Windows CE	Linux Native	For More Information...
JDBC	Yes Oracle Database Lite offer three JDBC drivers. Refer to Section 9.1.1, "JDBC Drivers".	Yes	Yes	Yes On Linux, only JDBC and ODBC access is supported.	Chapter 10, "JDBC Programming"
Java Stored Procedures /Triggers	Yes Java Stored Procedures/Triggers are not supported in the Web-to-Go application model. However Java Stored Procedures can be replicated using the Consolidator Manager API.	Yes	N/A	Yes	Chapter 11, "Stored Procedures and Triggers"
Java Server Pages	1.1	N/A	N/A	N/A	Section 6.2.3, "Developing JavaServer Pages"

Table 9–1 (Cont.) Java Support

Category	Web-to-Go (both Windows and Linux)	Windows 32 Native	Windows CE	Linux Native	For More Information...
Java Servlet	2.2	N/A	N/A	N/A	Section 6.2.4, "Developing Java Servlets for Web-to-Go" and Section 6.2.7, "Developing Applet Servlet Communication"
BC4J	Yes Latest version of Oracle JDeveloper 10*g*.	N/A	N/A	N/A	
Struts	Yes	N/A	N/A	N/A	

For programmatically synchronizing from a Java application, see Chapter 3, "Synchronization".

9.1.1 JDBC Drivers

The Oracle Database Lite JDBC driver is JDBC 1.2 compliant. Oracle Lite provides a limited number of extensions specified by JDBC 2.0. These extensions are compatible with the Oracle Database JDBC implementation.

Oracle Database Lite offers the following JDBC drivers:

- Type 2 driver: There are two types of type 2 driver: one provides an embedded, direct connection. This driver allows Java applications to communicate directly with the Oracle Lite database. The other type 2 driver provides a remote connection and requires Multi-User service support.
- Type 4 driver : 100% Java implementation. Requires the multi-user database version.

9.2 Oracle Database Lite Java Development Environment

To develop Java applications, you need to set up your development environment to create Oracle Database Lite applications, as follows:

- You must have the Sun Microsystems Java Development Kit (JDK), version 1.4.2 (or higher).
- To enable Oracle Database Lite to work with the JDK, set your `PATH` and `CLASSPATH` environment variables after you install Oracle Database Lite. See Section 9.2.1, "Setting Variables for the JDK" for full details.

> **Note:** If your environment includes a `CLASSPATH` user variable before you install Oracle Database Lite and the user variable does not include the `CLASSPATH` system variable (is not specified as `CLASSPATH=...;%CLASSPATH%`), then you must modify the `CLASSPATH` user variable to include the `olite40.jar` file in the *`OLITE_HOME`*`\bin` directory.

Note: All command prompt windows must be closed and reopened to reflect changes made to your CLASSPATH.

9.2.1 Setting Variables for the JDK

The directory with the JDK 1.4.2 or 5.0 Java compiler (`javac.exe`) should be in the `PATH` variable before any other directories that contain other Java compilers.

Add the directory that contains the Classic Java Virtual Machine (JVM) shared library, `jvm.dll`, to the PATH. `jvm.dll` should be in your `JDK_Home\jre\bin\classic` directory.

For example,

```
set PATH=C:\JDK_Home\bin;c:\JDK_Home\jre\bin\classic
set CLASSPATH=c:\JDK_Home\jrc\lib\rt.jar;c:\OLITE_HOME\bin\olite40.jar
```

As an alternative to using the Classic JVM, you can use the HotSpot JVM. HotSpot is a JDK add on module provided by Sun Microsystems. HotSpot is available from the Sun Microsystems Web site.

After installing HotSpot, set your `PATH` as given below.

```
set PATH=c:\jdk\bin;c:\jdk\jre\bin\hotspot;%PATH%
```

In the example above, your installation of the JDK and HotSpot is on Drive `C:\`. Verify the location of your installation before amending your `PATH` statement. To test whether your system is set up correctly, run the Java examples in the <*ORACLE_HOME*>`\Mobile\Sdk\Samples\JDBC` directory.

9.3 Java Development Tools

To write and debug Java programs, you can use any Java development tool. However, you must ensure that you set the `CLASSPATH` and `PATH` correctly.

10 JDBC Programming

This chapter discusses the Oracle Database Lite support for JDBC programming. It includes the following topics:

10.1 JDBC Compliance

Oracle Database Lite provides a native JDBC driver that allows Java applications to communicate directly with the Oracle Database Lite object relational database engine. The Oracle Database Lite implementation of JDBC complies with JDBC 1.2. In addition, Oracle Database Lite provides certain extensions specified by JDBC 2.0, which are compatible with the Oracle database JDBC implementation. For a complete JDBC reference, see the Sun Microsystems Web site.

10.2 JDBC Environment Setup

For your Java applications using the client/server model, include the `olite40.jar`, which is located in *`OLITE_HOME`*`/bin`, in the system `CLASSPATH` on the server machine and in the user `CLASSPATH` on the client machine.

When using the Oracle Lite JDBC driver in your OC4J application, use the default classloader instead of a per-application classloader, which many J2EE containers use. Ensure that the `olite40.jar` is in the OC4J `CLASSPATH` when OC4J initiates and not in the `/lib` subdirectory of your application WAR file.

Note: For more information on how to start the Multiuser Oracle Database Lite Database Service, see Section 2.5, "Oracle Database Lite Multi-User Service".

10.3 JDBC Drivers to Use When Connecting to Oracle Database Lite

Note: JDK 1.4.2 or 5.0 is required to connect to Oracle Database Lite.

Oracle Database Lite supports Type 2 and Type 4 drivers.

- The Type 2 driver uses native code on the client side through which it interfaces with the Oracle Database Lite ODBC driver.
- The Type 4 JDBC driver is a pure Java driver and uses the Oracle Database Lite network protocol to communicate with the Oracle Database Lite service. Before using this driver, ensure that you start Oracle Database Lite. Any Java applet can use the Type 4 JDBC driver.

The supported Type 2 and Type 4 drivers are described in the following sections:

- Section 10.3.1, "Type 2 Driver"
- Section 10.3.2, "Type4 (Pure Java) Driver Connection URL Syntax"

10.3.1 Type 2 Driver

For most applications, use the type 2 driver for connecting to the database. You can use the type 2 driver to connect either to the local Oracle Lite database or to the server where a Multi-User Service is managing the Oracle Lite databases.

- To connect to the local Oracle Lite database, use the following URL syntax:

  ```
  jdbc:polite[:uid / pwd]:localDSN[;key=value]*
  ```

  where the localDSN is the DSN name for the local Oracle Lite database (the ODB file on the local machine) and the optional key=value pairs are listed in Table 10–1.

  The following example retrieves a connection to the local Oracle Lite database, where the DSN name is `polite`:

  ```
  DriverManager.getConnection("jdbc:polite:polite","system","admin");
  ```

- To access the Oracle Lite database on a remote host where a Multi-User Service or Branch Office is located, use the following URL syntax:

  ```
  jdbc:polite[:uid / pwd]@[host]:[port]:serverDSN [;key=value]*
  ```

  where the host, port, and serverDSN identify the host, port and DSN of the remote host where the Oracle Lite database (the ODB file on the local machine) and the multi-user service is located. The optional key=value pairs are listed in Table 10–1.

  For more information on how to install and start the Multiuser Oracle Database Lite Service, refer to Section 2.5, "Oracle Database Lite Multi-User Service".

You can provide additional configuration information in the JDBC driver URL within key-value pairs, as specified in Table 10–1, each of which are separated by a semi-colon. The information specified within the key-value pairs always overrides the information that is specified in the URL.

Table 10–1 Key/Value Pairs for JDBC Connect URL

Argument	Description
`jdbc`	Identifies the protocol as JDBC.
`polite`	Identifies the subprotocol as `polite`.

Table 10–1 (Cont.) Key/Value Pairs for JDBC Connect URL

Argument	Description
uid / pwd	The optional user ID and password for Oracle Database Lite, each of which are limited to 28 characters. If specified, this overrides the specification of a username and password defined in the UID and PWD arguments. If the database is encrypted, you must include the password in the key-value pair.
host	The name of the machine that hosts the Multi-User Service or Branch Office and on which the Oracle Database Lite service `olsv2040.exe` runs. This host name is optional. If omitted, it defaults to the local machine on which the JDBC application runs.
port	The port number at which the Multi-User or Branch Office service listens. The port number is optional. If omitted, the port number defaults to port 1160.
dsn	Identifies the data source name (DSN) entry in the `odbc.ini` file. This entry contains all the necessary information to complete the connection to the server. **Note**: For a JDBC program, you need not create a DSN if you have supplied all the necessary values for the data directory and database through key=value pairs. On the windows platform, you can use the ODBC administrator to create a DSN. For more information, see Section 2.5.1.4, "Creating DSNs".
Data_Directory=	Directory in which the `.odb` file resides.
Database=	Name of database as given during its creation.
IsolationLevel=	Transaction isolation level: READ COMMITTED, REPEATABLE READ, SERIALIZABLE or SINGLE USER. For more information on isolation levels, see Section 15.2, "What Are the Transaction Isolation Levels?". **Note**: If you are retrieving a large object, such as a BLOB, within a READ COMMITTED transaction, see Section 4.3.46.10 "Select Statement Behavior When Retrieving BLOBs in a READ COMMMITTED transaction" section in the *Oracle Database Lite SQL Reference*.
Autocommit=	Commit behavior, either `ON` or `OFF`.
CursorType=	Cursor behavior: DYNAMIC, FORWARD ONLY, KEYSET DRIVEN or STATIC. For more information on cursor types, see Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types".
UID=	User name
PWD=	Password

Example of Using JDBC Type 2 Connection for Local Oracle Lite Database

```
String ConnectMe=("jdbc:polite:SCOTT/tiger:polite;
Data_Directory=<ORACLE_HOME>;Database=polite;IsolationLevel=SINGLE USER;
Autocommit=ON;CursorType=DYNAMIC")

try
   {Class.forName("oracle.lite.poljdbc.POLJDBCDriver")
   Connection conn = DriverManager.getConnection(ConnectMe)
   }
catch (SQLException e)
{
...
}
```

Example of Using Type 2 in Multi-User Service Situation

An example of this type of connection is given below.

```
try {
Connection conn = DriverManager.getConnection(
   "jdbc:polite@yourhostname
        ;Data_Directory=<ORACLE_HOME>
        ;Database=polite
        ;IsolationLevel=SINGLE USER
        ;Autocommit=ON
        ;CursorType=DYNAMIC", "Scott", "tiger")
}
catch (SQLException e)
{
}
```

10.3.2 Type4 (Pure Java) Driver Connection URL Syntax

Use the JDBC Type 4 driver for any pure Java application that uses the Multi-User or Branch office services. The URL syntax for the type 4 driver as follows:

```
jdbc:polite4[:uid/pwd]@[host]:[port]:serverDSN[;key=value]*
```

The parameter `polite4` indicates that the JDBC type 4 driver is being used. For the rest of the parameters, see the definitions of those parameters for the type 2 driver, as described in Table 10–1.

Note: The URL works with the Oracle Database Lite service only.

10.4 DataSource Connection

In JDBC 2.0, the `DataSource` object is an alternative to the `DriverManager` facility. The `DataSource` object is the preferred method for retrieving a connection and is typically registered with a naming service based on the JNDI API. A driver that is accessed through a `DataSource` object does not register itself with the `DriverManager`.

The Oracle Database Lite JDBC driver contains the basic `DataSource` implementation to produce a standard `Connection` object. The retrieved connection is identical to a connection obtained through the `DriverManager`.

Oracle Database Lite implements `javax.sql.DataSource` interface with the `POLJDBCDataSource` class in the `oracle.lite.poljdbc` package.

As with any class that implements the `DataSource` interface, the `POLJDBCDataSource` object defines properties for connecting to a specific database. In addition to the standard DataSource properties, the `POLJDBCDataSource` class has one additional property, which is a `String` to define the URL of the database connection string, as described in Section 10.3.1, "Type 2 Driver" and Section 10.3.2, "Type4 (Pure Java) Driver Connection URL Syntax".

- The URL can include username and password, which then overrides any previous individual property settings. The following URL specifies a username and password of `system/manager`:

  ```
  jdbc:polite4:system/manager@::polite
  ```

- You must have a username and password defined either in the URL or in the `getConnection` method. If defined in both places, then the username and password in the getConnection takes precedence over the URL definitions.

See the JDBC example—`JDBCEXJSR169`—on the CD for how to use the `POLJDBCDataSource` object.

10.5 Java Datatypes and JDBC Extensions

The Oracle Database Lite JDBC driver supports JDBC 1.2 and provides extensions that support certain features defined in JDBC 2.0. The extensions include support for BLOB (large binary object) and CLOB (large character object) datatypes and scrollable result sets. The Oracle Database Lite JDBC extensions are compatible with the Oracle database JDBC implementation. However, Oracle Database Lite does not support the following Oracle database JDBC datatype extensions: `Array`, `Struct`, or `REF`.

The following sections list and describe the Oracle Database Lite datatypes and data access extensions. For details regarding function syntax and call parameters, see the Sun Microsystems Java 2 specification at the Sun Microsystems Web site.

- Section 10.5.1, "Mapping Datatypes Between Java and Oracle"
- Section 10.5.2, "Datatype Extensions"
- Section 10.5.3, "Data Access Extensions"

10.5.1 Mapping Datatypes Between Java and Oracle

Oracle Database Lite performs type conversions between Java and Oracle datatypes as indicated by the following table. Table 10–2 lists the Java datatypes and the corresponding SQL datatypes that result from the type conversion.

Table 10–2 Datatype Conversions

Java Datatype	SQL Datatype
`byte[], byte[][], Byte[]`	`BINARY, RAW, VARBINARY, BLOB`
`boolean, Boolean`	`BIT`
`String, String[]`	`CHAR, VARCHAR, VARCHAR2, CLOB`
`short, short[], short[][], Short, Short[]`	`SMALLINT`
`int,int[], int[][], Integer, Integer[]`	`INT`
`float, float[], float[][], Float, Float[]`	`REAL`
`double, double[], double[][], Double, Double[]`	`DOUBLE, NUMBER` (without precision)
`BigDecimal, BigDecimal[]`	`NUMBER(n)`
`java.sql.Date, java.sql.Date[]`	`DATE`
`java.sql.Time, java.sql.Time[]`	`TIME`
`java.sql.Timestamp, java.sql.Timestamp[]`	`TIMESTAMP`
`java.sql.Connection`	Default JDBC connection to database

10.5.2 Datatype Extensions

BLOBs and CLOBs store data items that are too large to store directly in a database table. Rather than storing the data, the database table stores a locator that points to the location of the actual data. BLOBs contain a large amount of unstructured binary data items and CLOBs contain a large amount of fixed-width character data items (characters that require a fixed number of bytes per character).

You can select a BLOB or CLOB locator from the database using a standard SELECT statement. When you select a BLOB or CLOB locator using SELECT, you acquire only the locator for the large object, not the data itself. Once you have the locator, however, you can read data from or write data to the large object using access functions.

Note: If you are retrieving a large object, such as a BLOB, within a READ COMMITTED transaction, see the "Select Statement Behavior When Retrieving BLOBs in a READ COMMMITTED transaction" section in the *Oracle Database Lite SQL Reference*.

Table 10–3 lists the methods included in the Oracle Database Lite BLOB class and their descriptions:

Table 10–3 Methods in the Oracle Database Lite BLOB Class

Function	Description
`length`	Returns the length of a BLOB in bytes.
`getBinaryOutputStream`	Returns BLOB data.
`getBinaryStream`	Returns a BLOB instance as a stream of bytes.
getBytes	Reads BLOB data, starting at a specified point, into a buffer.
`getConnection`	Returns the current connection.
`isConvertibleTo`	Determines if a BLOB can be converted to a particular class.
`putBytes`	Writes bytes to a specified point in the BLOB data.
`makeJdbcArray`	Returns the JDBC array representation of a BLOB.
`toJdbc`	Converts a BLOB to a JDBC class.
`trim`	Trims to length.

Table 10–4 lists the methods included in the Oracle Database Lite CLOB class and their descriptions.

Table 10–4 Methods in the Oracle Database Lite CLOB Class

Function	Description
`length`	Returns the length of a CLOB in bytes.
`getSubString`	Retrieves a substring from a specified point in the CLOB data.
`getCharacterStream`	Returns CLOB data as a stream of Unicode characters.
`getAsciiStream`	Returns a CLOB instance as an ASCII stream.
`getChars`	Retrieves characters from a specified point in the CLOB data into a character array.
`getCharacterOutputStream`	Writes CLOB data from a Unicode stream.
`getAsciiOutputStream`	Writes CLOB data from an ASCII stream.

Table 10–4 (Cont.) Methods in the Oracle Database Lite CLOB Class

Function	Description
`getConnection`	Returns the current connection.
`putChars`	Writes characters from a character array to a specified point in the CLOB data.
`putString`	Writes a string to a specified point in the CLOB data.
`toJdbc`	Converts a CLOB to a JDBC class.
`isConvertibleTo`	Determines if a CLOB can be converted to a particular class.
`makeJdbcArray`	Returns a JDBC array representation of a CLOB.
`trim`	Trims to length.

10.5.3 Data Access Extensions

Oracle Database Lite provides access functions to set and return values of the CLOB and BLOB datatypes. In addition, stream classes provide functions that enable stream-format access to large objects.

The large object access functions are located in the `OraclePreparedStatement`, the `OracleCallableStatement`, and the `OracleResultSet` class.

Table 10–5 lists the data access functions included in the `OracleResultSet` class.

Table 10–5 Data Access Functions in the OracleResultSet Class

Function	Description
`getBLOB`	Returns a locator to BLOB data.
`getCLOB`	Returns a locator to CLOB data.

The stream format access classes are `POLLobInputStream`, `POLLobOutputStream`, `POLClobReader`, and `POLClobWriter`.

The `POLLobInputStream` class includes the following data access function.

Function	Description
`read`	Reads from a large object into an array.

The `POLLobOutputStream` class includes this data access function.

Function	Description
`write`	Writes from an output stream into a large object.

The `POLClobReader` class extends the class `java.io.reader`. It includes these data access functions.

Function	Description
`read`	Reads characters from a CLOB into a portion of an array.
`ready`	Indicates whether a stream is ready to read.
`close`	Closes a stream.

Function	Description
`markSupported`	Indicates whether the stream supports the `mark` operation.
`mark`	Marks the current position in the stream. Subsequent calls to the `reset` function reposition the stream to the marked location.
`reset`	Resets the current position in the stream to the marked location. If the stream has not been marked, this function attempts to reset the stream in a way appropriate to the particular stream, such as by repositioning it at its starting point.
`skip`	Skips characters in the stream.

The `POLClobWriter` class extends the class `java.io.writer`. It includes these data access functions:

Function	Description
`write`	Writes an array of characters to the output stream.
`flush`	Writes any characters in a buffer to their intended destination.
`close`	Flushes and closes the stream.

10.5.3.1 Reading from a BLOB Sample Program

The following sample uses the `getBinaryStream` method to read BLOB data into a byte stream. It then reads the byte stream into a byte array, and returns the number of bytes read.

```
// Read BLOB data from BLOB locator.
InputStream byte_stream = my_blob.getBinaryStream();
byte [] byte_array = new byte [10];
int bytes_read = byte_stream.read(byte_array);
```

10.5.3.2 Writing to a CLOB Sample Program

The following sample reads data into a character array, then uses the `getCharacterOutputStream` method to write the array of characters to a CLOB.

```
java.io.Writer writer;
char[] data = {'0','1','2','3','4','5','6','7','8','9'};

// write the array of character data to a CLOB
writer = ((CLOB)my_clob).getCharacterOutputStream();
writer.write(data);
writer.flush();
writer.close();
```

10.6 Limitations

If data truncation occurs during a write, a SQL data truncation exception is thrown. A SQL data truncation warning results if data truncation occurs during a read.

The Oracle Database Lite JDBC classes and the JDBC 2.0 classes use the same name for certain datatypes (for example, `oracle.sql.Blob` and `java.sql.Blob`). If your program imports both `oracle.sql.*` and `java.sql.*`, attempts to access the overlapping classes without fully qualifying their names may result in compiler errors. To avoid this problem, use one of the following steps:

1. Use fully qualified names for BLOB, CLOB, and data classes.

2. Import the class explicitly (for example, import `oracle.sql.Blob`).

Class files always contain fully qualified class names, so the overlapping datatype names do not cause conflicts at runtime.

10.7 New JDBC 2.0 Features

This section describes JDBC 2.0 methods or interfaces that are supported by the Oracle Database Lite JDBC driver. Topics include:

- Section 10.7.1, "Interface Connection"
- Section 10.7.2, "Interface Statement"
- Section 10.7.3, "Interface ResultSet"
- Section 10.7.4, "Interface Database MetaData"
- Section 10.7.5, "Interface ResultMetaData"
- Section 10.7.6, "Interface PreparedStatement"

10.7.1 Interface Connection

This section describes the JDBC 2.0 Interface methods that are implemented by the Oracle Database Lite JDBC driver.

10.7.1.1 Methods

Statement

```
createStatement(int resultSetType, int resultSetConcurrency)
```

Creates a statement object that generates ResultSet objects with the given type and concurrency.

Map

```
getTypeMap()
```

Gets the TypeMap object associated with this connection.

CallableStatement

```
prepareCall(String sql, int resultSetType, int
resultSetConcurrency)
```

Creates a CallableStatement object that generates ResultSet objects with the given type and concurrency.

PreparedStatement

```
prepareStatement(String sql, int resultSetType, int
resultSetConcurrency)
```

Creates a `PreparedStatement` object that generates `ResultSet` objects with the given type and concurrency.

void

```
setTypeMap(Map map)
```

Installs the given type map as the type map for this connection.

10.7.2 Interface Statement

This section describes the JDBC 2.0 Interface Statement methods that are implemented by the Oracle Database Lite JDBC driver.

Connection

`getConnection()`

Returns the Connection object that produced this Statement object.

int

`getFetchDirection()`

Retrieves the direction for fetching rows from database tables that is the default for result sets generated from this Statement object. Only FETCH_FORWARD is supported for now.

int

`getFetchSize()`

Retrieves the number of result set rows that is the default fetch size for result sets generated from this Statement object. Only fetch size = 1 is supported for now.

int

`getResultSetConcurrency()`

Retrieves the result set concurrency. Only CONCUR_READ_ONLY is supported for now.

int

`getResultSetType()`

Determine the result set type. Only TYPE_FORWARD_ONLY and TYPE_SCROLL_INSENSITIVE are supported for now.

void

`setFetchDirection(int direction)`

Gives the driver a hint as to the direction in which the rows in a result set will be processed.

void

`setFetchSize(int rows)`

Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed.

10.7.3 Interface ResultSet

This section describes the JDBC 2.0 Interface ResultSet methods that are implemented by the Oracle Database Lite JDBC driver.

10.7.3.1 Fields

The following fields can be used to implement the Interface ResultSet feature.

static int

`CONCUR_READ_ONLY`

The concurrency mode for a ResultSet object that may NOT be updated.

static int

`CONCUR_UPDATABLE`

The concurrency mode for a ResultSet object that may be updated. Not supported for now.

static int

`FETCH_FORWARD`

The rows in a result set will be processed in a forward direction; first-to-last.

static int

`FETCH_REVERSE`

The rows in a result set will be processed in a reverse direction; last-to-first. Not supported for now.

static int

`FETCH_UNKNOWN`

The order in which rows in a result set will be processed is unknown.

static int

`TYPE_FORWARD_ONLY`

The type for a ResultSet object whose cursor may move only forward.

static int

`TYPE_SCROLL_INSENSITIVE`

The type for a ResultSet object that is scrollable but generally not sensitive to changes made by others.

static int

`TYPE_SCROLL_SENSITIVE`

The type for a ResultSet object that is scrollable and generally sensitive to changes made by others. Not supported for now.

10.7.3.2 Methods

This section describes the JDBC 2.0 ResultSet method implemented by the Oracle Database Lite JDBC driver.

boolean

`absolute(int row)`

Moves the cursor to the given row number in the result set.

void

`afterLast()`

Moves the cursor to the end of the result set, just after the last row.

void

`beforeFirst()`

Moves the cursor to the front of the result set, just before the first row.

boolean

`first()`

Moves the cursor to the first row in the result set.

Array

`getArray(String colName)`

Gets an SQL ARRAY value in the current row of this ResultSet object.

BigDecimal

`getBigDecimal(int columnIndex)`

Gets the value of a column in the current row as a java.math.BigDecimal object with full precision.

BigDecimal

`getBigDecimal(String columnName)`

Gets the value of a column in the current row as a java.math.BigDecimal object with full precision.

int

`getConcurrency()`

Returns the concurrency mode of this result set.

Date

`getDate(int columnIndex, Calendar cal)`

Gets the value of a column in the current row as a java.sql.Date object.

int

`getFetchDirection()`

Returns the fetch direction for this result set.

int

`getFetchSize()`

Returns the fetch size for this result set.

int

`getRow()`

Retrieves the current row number.

Statement

`getStatement()`

Returns the Statement that produced this ResultSet object.

int

`getType()`

Returns the type of this result set.

boolean

`isAfterLast()`

boolean

`isBeforeFirst()`

boolean

`isFirst()`

boolean

`isLast()`

boolean

`last()`

Moves the cursor to the last row in the result set.

boolean

`previous()`

Moves the cursor to the previous row in the result set.

void

`refreshRow()`

Refreshes the current row with its most recent value in the database. Currently does nothing.

boolean

`relative(int rows)`

Moves the cursor a relative number of rows, either positive or negative.

10.7.3.3 Methods that Return False

The following three methods always return false because this release does not support deletes, inserts, or updates.

boolean

`rowDeleted()`

Indicates whether a row has been deleted.

boolean

`rowInserted()`

Indicates whether the current row has had an insertion.

boolean

`rowUpdated()`

Indicates whether the current row has been updated.

void

`setFetchDirection(int direction)`

Gives a hint as to the direction in which the rows in this result set will be processed.

void

`setFetchSize(int rows)`

Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed for this result set.

10.7.4 Interface Database MetaData

This section describes the JDBC 2.0 Interface Database MetaData methods that are implemented by the Oracle Database Lite JDBC driver.

10.7.4.1 Methods

The following methods can be used to implement the Interface Database MetaData feature.

Connection

`getConnection()`

Retrieves the connection that produced this metadata object.

boolean

`supportsResultSetConcurrecny(int type, int concurrency)`

Supports the concurrency type in combination with the given result set type.

boolean

`supportsResultSetType(int Type)`

Supports the given result set type.

10.7.4.2 Methods that Return False

The following methods return false, because this release does not support deletes or updates.

boolean

`deletesAreDetected(int Type)`

Indicates whether or not a visible row delete can be detected by calling ResultSet.rowDeleted().

boolean

`insertsAreDetected(int Type)`

Indicates whether or not a visible row insert can be detected by calling ResultSet.rowInserted().

boolean

`othersDeletesAreVisible(int Type)`

Indicates whether deletes made by others are visible.

boolean

`othersInsertsAreVisible(int Type)`

Indicates whether inserts made by others are visible.

boolean

`othersUpdatesAreVisible(int Type)`

Indicates whether updates made by others are visible.

boolean

`ownDeletesAreVisible(int Type)`

Indicates whether a result set's own deletes are visible.

boolean

`ownInsertsAreVisible(int Type)`

Indicates whether a result set's own inserts are visible.

boolean

`ownUpdatesAreVisisble(int Type)`

Indicates whether a result set's own updates are visible.

boolean

`updatesAreDetected(int Type)`

Indicates whether or not a visible row update can be detected by calling the method ResultSet.rowUpdated.

10.7.5 Interface ResultMetaData

This section lists methods that can be implemented using the Interface ResultMetaData feature.

10.7.5.1 Methods

The following method can be used to implement the Interface ResultMetaData feature.

String

`getColumnClassName(int column)`

Returns the fully-qualified name of the Java class whose instances are manufactured if the method ResultSet.getObject is called to retrieve a value from the column.

10.7.6 Interface PreparedStatement

This section describes methods that can be implemented using the Interface PreparedStatement feature.

10.7.6.1 Methods

The following methods can be used to implement the Interface PreparedStatement feature.

Result

`SetMetaDatagetMetaData()`

Gets the number, types and properties of a ResultSet's columns.

void

`setDate(int parameter Index, Date x, Calendar cal)`

Sets the designated parameter to a `java.sql.Date` value, using the given Calendar object.

void

`setTime(int parameterIndex, Time x, Calendar cal)`

Sets the designated parameter to a `java.sql.Time` value, using the given Calendar object.

void

`setTimestamp(int parameter Index, Timestamp x, Calendar cal)`

Sets the designated parameter to a `java.sql.Timestamp` value, using the given Calendar object.

10.7.6.1.1 Limitation currently, the option `setQueryTimeOut` is not supported.

10.8 J2ME Support

The following sections describe what Oracle Database Lite supports for J2ME:

- Section 10.8.1, "JDBC Drivers for J2ME CDC and CLDC"
- Section 10.8.2, "J2ME Support for Windows CE"

10.8.1 JDBC Drivers for J2ME CDC and CLDC

Oracle Database Lite JDBC drivers for J2ME are supported in a limited capacity. The following sections describe what are supported in Oracle Database Lite JDBC/J2ME drivers:

- Section 10.8.1.1, "JDBC Driver for J2ME CDC"
- Section 10.8.1.2, "JDBC Driver for J2ME CLDC"

10.8.1.1 JDBC Driver for J2ME CDC

You can use the `olite40.jar` file for JDBC J2ME CDC application development. However, the Oracle Database Lite JDBC driver for J2ME CDC does not implement all classes and methods of the Sun Microsystems "JSR-000169 JDBC Optional Package for CDC/Foundation Profile".

The JDBC definition classes (`java.sql.*`) are an optional package for CDC/Foundation profile based JVMs, as defined by the Javasoft JSR 169 specification. Obtain these classes from your JVM vendor. If your JVM vendor does not supply these classes, then use the sample implementation provided with Oracle Database Lite, which can be found in the *<OLITE_HOME>*`\Mobile\Sdk\samples\j2me` directory.

See Section 10.1, "JDBC Compliance" and Section 10.5, "Java Datatypes and JDBC Extensions" for what is supported in this JAR file for the JDBC driver on J2ME CDC.

10.8.1.2 JDBC Driver for J2ME CLDC

Oracle Database Lite provides the JDBC driver for J2ME CLDC application development. Only a subset of the JDBC APIs are available. The JDBC APIs for J2ME CLDC are provided in the `oracle.lite.jdbc` package, which is included in the `olitejdbccldc.jar` file. You must include this JAR file in your application to use JDBC for J2ME CLDC. The interface is a subset of features available in the Oracle Lite JDBC driver.

See Section 10.1, "JDBC Compliance" and Section 10.5, "Java Datatypes and JDBC Extensions" for what is supported in this JAR file for the JDBC driver on J2ME CDC.

The following lists what is NOT currently implemented for the JDBC API:

- You cannot use any methods requiring floating point data types, such as `ResultSet.getDouble`.
- You cannot use any methods requiring `java.sql.Date`, `java.sql.Time` or `java.sql.Timestamp` data types. When working with SQL date, time and timestamp data, consider using one of the following methods instead: `ResultSet.getString`, `PreparedStatement.setString` and other string getter and setter methods.
- You cannot use any `DatabaseMetaData` objects.
- The JDBC driver has no `finalize()` methods. Applications must explicitly close database objects when done.
- Since the `java.math` library is omitted from MIDP, you may not use the `BigDecimals` object. Use the appropriate `java.lang.String` getter or setter method.

Table 10–6 details the classes and exceptions available in this package. For more information on these classes, such as the supported methods, see the Javadoc in the *Oracle Database Lite API Specification*.

Table 10–6 J2ME CLDC Class and Exception Summary

Classes and Exceptions	Description
`BLOB`	The representation (mapping) in the Java programming language of an SQL BLOB value.
`CLOB`	The representation (mapping) in the Java programming language of an SQL CLOB value.
`Connection`	Connection represents a JDBC connection to an Oracle Lite database.
`PreparedStatement`	A `PreparedStatement` contains a pre-compiled SQL statement which may have parameter markers.
`ResultSet`	Represents a result set which is usually generated by executing a statement that queries the database.
`ResultsSetMetaData`	The `ResultSetMetaData` class can be used to get information about the types and properties of the columns in a `ResultSet` object.

Table 10–6 (Cont.) J2ME CLDC Class and Exception Summary

Classes and Exceptions	Description
`OliteDataSource`	`OLiteDataSource` partially implements the `javax.sql.DataSource` interface. It is for retrieving a `Connection` object.
`OracleConnection`	Provides the same functional support as the `Connection` object and also provides support for `BLOB`/`CLOB` objects.
`OraclePreparedStatement`	Provides the same functional support as the `PreparedStatement` object and also provides support for `BLOB`/`CLOB` objects.
`OracleResultSet`	Provides the same functional support as the `ResultSet` object and also provides support for `BLOB`/`CLOB` objects.
`Statement`	A `Statement` object is used for executing a static SQL statement and obtaining the results produced by it.
`Types`	The class that defines the constants that are used to identify generic SQL types, called JDBC types. This class is never instantiated.
`DataTuncation`	An exception that reports a `DataTruncation` warning (on reads) or throws a `DataTruncation` exception (on writes) when JDBC unexpectedly truncates a data value.
`SQLException`	An exception that provides information on a database access error or other errors.
`SQLWarning`	An exception that provides information on database access warnings.

For the JDBC/J2ME/CLDC driver, only the JDBC Type 4 connection URL syntax is supported. The following example demonstrates how to create a `Connection` object using the `OLiteDataSource` object.

```
import oracle.lite.jdbc.OLiteDataSource;
import oracle.lite.jdbc.Connection;
import oracle.lite.jdbc.Statement;
…

public class TestOLiteDataSource implements Runnable
{
  …
  public void run()
  {
    String strUrl = "jdbc:polite4:system/manager@::polite";
    try {
      OLiteDataSource olds = new OLiteDataSource();
      olds.setUrl(strUrl);
      Connection conn = olds.getConnection();
      conn.setAutoCommit(true);
      Statement stmt = conn.createStatement();
      stmt.execute("create table t1(c1 int) ");
    }
    catch (SQLException e) {
  …
    }
  }
}
```

Note: For a full description of what is supported for each object, see the *Oracle Database Lite API Specification* for the Javadoc on these objects.

10.8.2 J2ME Support for Windows CE

Oracle Database Lite is certified with the following JVMs on Windows Mobile 2003 Second Edition:

- IBM J9 Websphere Everyplace Micro Environment for Windows Mobile 2003 ARM Personal Profile
- Creme JVM, which can be obtained at `http://www.nsicom.com`

The following sections describe which class to use in connecting to an Oracle Lite database for each JVM type:

- Section 10.8.2.1, "Using IBM J9"
- Section 10.8.2.2, "Using Creme 4.1"

10.8.2.1 Using IBM J9

When using IBM J9, use the `DataSource` class to connect to an Oracle Lite database, as shown below:

```
POLJDBCDataSource dsPolite = new POLJDBCDataSource();
dsPolite.setUrl(DSN);
dsPolite.setUser(UserName);
dsPolite.setPassword(Password)
politeConnection = dsPolite.getConnection();
```

Perform the following to execute the `ExampleClass` sample class, which is part of the `ExamplePackage.jar`:

```
j9 -jcl:ppro10 "-Xbootclasspath/p:path\classes.zip;path\jdbcjsr169.jar"
 -classpath "path\jdbcjsr169.jar;\Orace\olite40.jar;path\ExamplePackage.jar"
 ExampleClass
```

Note: The `jdbcjsr169.jar` can be obtained from the SDK installation in the *<ORACLE_HOME>*`\Mobile\Sdk\samples\j2me` directory.

Where the `jdbcjsr169.jar` file contains the optional JDBC definitions. For more details, refer to Section 10.8.1.1, "JDBC Driver for J2ME CDC".

Note: Ensure that you replace the path with the correct path to the required JAR and ZIP files.

10.8.2.2 Using Creme 4.1

When using the Creme 4.1 JVM, use the `DriverManager` class to connect to an Oracle Lite database, as shown below:

```
politeConnection = DriverManager.getConnection(DSN,UserName,Password);
```

Perform the following command at the Creme Command prompt to execute the `ExampleClass` sample class, which is part of the `ExamplePackage.jar` file:

```
Creme -Of -classpath '<path>\jdbcjsr169.jar;\Oracle\olite40.jar;
        <path>\ExamplePackage.jar' ExampleClass <command_line_arguments>
```

11 Stored Procedures and Triggers

The following sections describe how to use either Java, C++, or .Net stored procedures and triggers within the Oracle Database Lite relational model:

11.1 Java Stored Procedure Features in Oracle Database Lite

Oracle Database Lite supports the Oracle database server development model for stored procedures. The "load and publish" development model occurs when you load the Java class into the Oracle Database Lite database instead of attaching the classes to tables. To implement a Java stored procedure in the Oracle Lite database, do the following:

1. Load the Java class into the Oracle Database Lite database with either the `loadjava` command-line utility or the SQL statement `CREATE JAVA`.
2. Publish the methods in the class that you want to call from SQL with a call specification, which is created with either the `CREATE FUNCTION` or `CREATE PROCEDURE` commands.

Note: For more information, see Section 11.3.1, "Using the Load and Publish Stored Procedure Development Model".

Oracle Database Lite supports the traditional model of creating stored procedures. In the traditional model, you attach the Java class to a table where:

- The static methods in the class become table-level stored procedures of the table.
- The non-static (instance) methods become row-level stored procedures.

The `loadjava` utility automates the task of loading Java classes into the database. Using `loadjava`, you can load Java class, source, and resource files, individually or in archives.

11.2 Overview of Java Stored Procedures and Triggers

- Java stored procedure: A Java stored procedure is a Java method that is stored in Oracle Database Lite. The procedure can be invoked by applications that access the database. Java stored procedures can return a single value, a row, or multiple rows.
- Trigger: A trigger is a stored procedure that executes when a specific event occurs, such as a row update, insertion, or deletion. An update of a specific column can also fire a trigger. Triggers, however, cannot return a value. A trigger can operate at the statement-level or row-level.
  - A statement-level trigger fires once per triggering statement, no matter how many rows are affected.
  - A row-level trigger fires once for every row affected by the triggering statement.

11.2.1 Creating Java Stored Procedures

To create a stored procedure, perform the following:

1. Create the class that you want to store in Oracle Database Lite. You can use any Java IDE to write the procedure, or you can simply reuse an existing procedure that meets your needs.

   When creating the class, consider the following restrictions on calling Java stored procedures from SQL DML statements:

   - When called from an `INSERT`, `UPDATE`, or `DELETE` statement, the method cannot query or modify any database tables modified by that statement.
   - When called from a `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement, the method cannot execute SQL transaction control statements, such as `COMMIT` or `ROLLBACK`.

   Note: Any SQL statement in a stored procedure that violates a restriction produces an error at run time.

2. Provide your class with a unique name for its deployment environment, since only one Java Virtual Machine is loaded for each Oracle Database Lite application. If the application executes methods from multiple databases, then the Java classes from these databases are loaded into the same Java Virtual Machine. We recommend that you prefix the Java class name with the database name to ensure that the Java class names are unique across multiple databases.
3. If you are executing any DML statements in your Java stored procedure, then—in order for these statements to exist within the same transaction—you must pass an argument of type `java.sql.Connection` as the first argument in the method. You must have the `Connection` object in order to prepare and execute any statements. Oracle Database Lite supplies the appropriate argument value of the Oracle Lite database `Connection` object for you; the application executing the method does not need to provide a value for this parameter.

11.3 Creating Java Stored Procedures

Oracle Database Lite supports the following development models for creating Java stored procedures:

- Section 11.3.1, "Using the Load and Publish Stored Procedure Development Model"
- Section 11.3.2, "Using the Attached Stored Procedure Development Model"

In addition, see the following sections for additional information about managing your Java stored procedures:

- Section 11.3.3, "Calling Java Stored Procedures From a Multithreaded C or C++ Application"

11.3.1 Using the Load and Publish Stored Procedure Development Model

You do not need to publish every procedure that you store in Oracle Database Lite, only those that should be callable from SQL. Many stored procedures are only called by other stored procedures, and do not need to be published. The load and publish model only supports static methods. Perform the following to create Java stored procedures:

1. Develop a Java class that contains the methods you want to store. Make sure that the class compiles and executes without errors.
2. Load the Java class into Oracle Database Lite with either the `loadjava` utility or the `SQL CREATE JAVA` command.
3. Publish any static methods in the Java class that you want to make accessible to SQL by creating call specifications for these methods. By publishing a method, you associate a SQL name to the method. SQL applications use this name to invoke the method.

   This model is supported by Oracle database, which enables you to utilize skills and resources you have already developed in implementing Oracle database enterprise applications and data. There is the following difference:

   - In Oracle Database Lite, you cannot publish a method that is mapped to a `main` method.
   - In the Oracle database, call specs that publish `main` methods are permitted.
4. Invoke the stored procedure through a SQL DML statement.
5. If you no longer intend to use the stored procedure, you can drop it from the database.

Note: The load and publish development model only supports Java static methods. To store static and non-static (instance) methods, you must attach the class to database tables, as described in Section 11.3.2, "Using the Attached Stored Procedure Development Model".

The following sections describe these activities in detail:

- Section 11.3.1.1, "Loading Java Stored Procedure Classes Into the Oracle Lite Database"
- Section 11.3.1.2, "Publishing Stored Procedures to SQL"

- Section 11.3.1.3, "Calling Published Stored Procedures"
- Section 11.3.1.4, "Dropping Published Stored Procedures"
- Section 11.3.1.5, "Example Using the Load and Publish Model"

11.3.1.1 Loading Java Stored Procedure Classes Into the Oracle Lite Database

To load Java classes into the Oracle Database Lite database, you can use one of the following:

- Section 11.3.1.1.1, "loadjava"—The `loadjava` database command-line utility automates the task of loading Java classes into Oracle Database Lite and Oracle databases.
- Section 11.3.1.1.2, "Using CREATE JAVA"—The SQL statement `CREATE JAVA` loads Java classes manually.

11.3.1.1.1 loadjava The `loadjava` command-line utility creates schema objects from files and loads them into the database. Schema objects can be created from Java source files, class files, and resource files. Resource files may be image files, resources, or anything else a procedure may need to access as data. You can pass files to `loadjava` individually, or as ZIP or JAR archive files.

Oracle Database Lite does not keep track of class dependencies. Make sure that you load into the database, or place in the `CLASSPATH`, all supporting classes and resource files required by a stored procedure. To query the classes that are loaded in the database, you can query the `okJavaObj` meta class.

Note: The table name and column names are case sensitive.

Syntax

```
loadjava {-user | -u} username/password[@database]
   [-option_name -option_name ...] filename filename ...
```

Arguments

This section discusses the `loadjava` arguments in detail.

User

The `user` argument specifies a username, password, and database directory in the following format:

```
<user>/<password>[@<database>]
```

For example:

```
scott/tiger@ ORACLE_HOME\Mobile\Sdk\OLDB40\Polite.odb
```

Options

Oracle Database Lite supports the following options that are listed and described in Table 11–1.

Table 11–1 Options

Option	Description
`-force | -f`	Forces files to be loaded, even if a schema object with the same name already exists in the database.
`-verbose | -v`	Directs `loadjava` to display detailed status messages while running.

Table 11–1 (Cont.) Options

Option	Description
`-meta | -m`	Creates the meta information in the database but does not load the classes. This is useful when the classes are in a .jar file and are not loaded into the database.

When specifying multiple options, you must separate the options with spaces. For example:

```
-force -verbose
```

The Oracle database supports additional options. If used with Oracle Database Lite, the additional options are recognized but not supported. Using them does not result in an error.

To view the options supported by Oracle database, see the `loadjava` help information using the following syntax.

```
loadjava {-help | -h}
```

Filenames

On the command line, you can specify as many class, source, JAR, ZIP, and resource files as you like, in any order. You must separate multiple file names with spaces, not commas. If passed a source file, `loadjava` invokes the Java compiler to compile the file before loading it into the database. If passed a JAR or ZIP file, `loadjava` processes each file in the JAR or ZIP. It does not create a schema object for the JAR or ZIP archive. The `loadjava` utility does not process a JAR or ZIP archive within another JAR or ZIP archive.

The best way to load files is to place them in a JAR or ZIP and then load the archive. Loading archives avoids the complications associated with resource schema object names. If you have a JAR or ZIP that works with the JDK, then you can be sure that loading it with `loadjava` also works, and you can avoid the complications associated with resource schema object naming.

As it loads files into the database, `loadjava` must create a name for the schema objects it creates for the files. The names of schema objects differ slightly from filenames, and different schema objects have different naming conventions. Class files are self-identifying, so `loadjava` can map their filenames to the names of schema objects automatically. Likewise, JAR and ZIP archives include the names of the files they contain.

However, resource files are not self-identifying; `loadjava` derives the names of Java resource schema objects from the literal names you enter on the command-line (or the literal names in a JAR or ZIP archive). Because classes use resource schema objects while executing, it is important that you specify the correct resource file names on the command line.

The best way to load individual resource files is to run `loadjava` from the top of the package tree, specifying resource file names relative to that directory. If you decide not to load resource files from the top of the package tree, you must be aware of how `loadjava` derives a name for your resource.

When you load a resource file, `loadjava` derives the name of the resource schema object from the file name that you enter on the command line. Suppose you type the following relative and absolute pathnames on the command line:

```
cd \scott\javastuff
loadjava options alpha\beta\x.properties
loadjava options  \scott\javastuff\alpha\beta\x.properties
```

Although you have specified the same file with a relative and an absolute pathname, `loadjava` creates *two* schema objects:

- `alpha\beta\x.properties`
- `\scott\javastuff\alpha\beta\x.properties`.

The `loadjava` utility generates the resource schema object's name from the file names *you enter*.

Classes can refer to resource files relatively (for example, `b.properties`) or absolutely (for example, `\a\b.properties`). To ensure that `loadjava` and the class loader use the same name for a schema object, pass `loadjava` *the name of the resource that the class passes to the* `java.lang.Object.getResource` *or* `java.lang.Class.`*`getResourceAsStream`* *method.*

Instead of remembering whether classes use relative or absolute resource names and changing directories so that you can enter the correct name on the command line, you can load resource files into a JAR file, as follows:

```
cd \scott\javastuff
jar -cf alpharesources.jar alpha\*.properties
loadjava options alpharesources.jar
```

Or, to simplify further, put both the class and resource files in a JAR, which makes the following invocations equivalent:

```
loadjava options alpha.jar
loadjava options \scott\javastuff\alpha.jar
```

Example
The following loads a class and resource file into Oracle Database Lite. It uses the `force` option; if the database already contains objects with the specified names, `loadjava` replaces them.

```
c:\> loadjava -u scott/tiger@c:\Olite\Mobile\Sdk\OLDB40\Polite.odb -f Agent.class\
images.dat
```

11.3.1.1.2 Using CREATE JAVA

To load Java classes manually, use the following syntax:

```
CREATE [OR REPLACE] [AND RESOLVE] [NOFORCE]
   JAVA {CLASS [SCHEMA <schema_name>] |
   RESOURCE NAMED [<schema_name>.]<primary_name>}
   [<invoker_rights_clause>]
   RESOLVER <resolver_spec>]
   USING BFILE ('<dir_path>', '<class_name>')
```

The following apply to the `CREATE JAVA` parameters:

- The `OR REPLACE` clause, if specified, recreates the function or procedure if one with the same name already exists in the database.
- For compatibility with the Oracle database, Oracle Database Lite recognizes but ignores the `<resolver_spec>` clause. Unlike the Oracle database, Oracle Database Lite does not resolve class dependencies. When loading classes manually, be sure to load all dependent classes.
- Oracle Database Lite recognizes, but ignores, `<invoker_rights_clause>`.

Example

The following demonstrates a `CREATE JAVA` statement. It loads a class named `Employee` into the database.

```
CREATE JAVA CLASS USING BFILE ('c:\myprojects\java',
   'Employee.class');
```

11.3.1.2 Publishing Stored Procedures to SQL

After loading the Java class into the Oracle Database Lite database using `loadjava` or `CREATE JAVA`, publish any static method in the class that you want to call from SQL by creating a call specification for it. The call spec maps the Java method's name, parameter types, and return types to SQL counterparts.

You do not need to publish every stored procedure, only those that serve as entry points for your application. In a typical implementation, many stored procedures are called only by other stored procedures, not by SQL users.

To create a call spec, use the SQL commands `CREATE FUNCTION` for methods that return a value or `CREATE PROCEDURE` for methods that do not return a value. The `CREATE FUNCTION` and `CREATE PROCEDURE` statements have the following syntax:

```
CREATE [OR REPLACE]
   { PROCEDURE [<schema_name>.]<proc_name> [([<sql_parms>])] |
   FUNCTION [<schema_name>.]<func_name> [([<sql_parms>])]
   RETURN <sql_type> }
   <invoker_rights_clause>
   { IS | AS } LANGUAGE JAVA NAME
   '<java_fullname>  ([<java_parms>])
   [return <java_type_fullname>]';
   /
```

The following apply to this statement's keywords and parameters:

- `<sql_parms>` has the following format:

  ```
  <arg_name> [IN | OUT | IN OUT]
           <datatype>
  ```

- `<java_parms>` is the fully qualified name of the Java datatype.
- For compatibility with the Oracle database, Oracle Database Lite recognizes but ignores the `<invoker_rights_clause>` clause.
- `<java_fullname>` is the fully qualified name of a static Java method.
- `IS` and `AS` are synonymous.

For example, assume the following class has been loaded into the database:

```
import java.sql.*;
import java.io.*;

public class GenericDrop {
   public static void dropIt (Connection conn, String object_type,
                    String object_name) throws SQLException {
      // Build SQL statement
      String sql = "DROP " + object_type + " " + object_name;
      try {
         Statement stmt = conn.createStatement();
         stmt.executeUpdate(sql);
         stmt.close();
      } catch (SQLException e) {
         System.err.println(e.getMessage());}
```

```
   }  // dropIt
}  // GenericDrop
```

Class `GenericDrop` has one method named `dropIt`, which drops any kind of schema object. For example, if you pass the arguments "table" and "emp" to `dropIt`, the method drops the database table EMP from your schema.

The following call specification publishes the method to SQL:

```
CREATE OR REPLACE PROCEDURE drop_it (
      obj_type VARCHAR2,
      obj_name VARCHAR2)
   AS LANGUAGE JAVA
   NAME 'GenericDrop.dropIt(java.sql.Connection,
      java.lang.String, java.lang.String)';
   /
```

Note: You must fully qualify the Java datatype parameters.

Given that you have a table named `TEMP` defined in your schema, you can execute the `drop_it` procedure from SQL Plus as follows.

```
Select drop_it('TABLE', 'TEMP') from dual;
```

You can also execute the `drop_it` procedure from within a ODBC application using an ODBC CALL statement. For more information, refer Section 11.3.3, "Calling Java Stored Procedures From a Multithreaded C or C++ Application".

11.3.1.3 Calling Published Stored Procedures

After publishing the stored procedure to SQL, call it with a SQL DML statement. For example, assume that this class is stored in the database:

```
public class Formatter {
   public static String formatEmp (String empName, String jobTitle) {
      empName = empName.substring(0,1).toUpperCase() +
         empName.substring(1).toLowerCase();
      jobTitle = jobTitle.trim().toLowerCase();
      if (jobTitle.equals("analyst"))
         return (new String(empName + " is an exempt analyst"));
      else
      return (new String(empName + " is a non-exempt " + jobTitle));
  }
}
```

Class `Formatter` has one method named `formatEmp`, which returns a formatted string containing an employee's name and job status. Create a call spec for `Formatter` as follows:

```
CREATE OR REPLACE FUNCTION format_emp (ename VARCHAR2, job VARCHAR2)
   RETURN VARCHAR2
   AS LANGUAGE JAVA
   NAME 'Formatter.formatEmp (java.lang.String, java.lang.String)
   return java.lang.String';
   /
```

The call spec publishes the method `formatEmp` as `format_emp`. Invoke it as follows:

```
SELECT FORMAT_EMP(ENAME, JOB) AS "Employees" FROM EMP
   WHERE JOB NOT IN ('MANAGER', 'PRESIDENT') ORDER BY ENAME;
```

This statement produces the following output:

```
Employees
-------------------------------------------
Adams is a non-exempt clerk
Allen is a non-exempt salesman
Ford is an exempt analyst
James is a non-exempt clerk
Martin is a non-exempt salesman
Miller is a non-exempt clerk
Scott is an exempt analyst
Smith is a non-exempt clerk
Turner is a non-exempt salesman
Ward is a non-exempt salesman
```

Note: Oracle Database Lite does not support the Oracle database `SQL CALL` statement for invoking stored procedures.

For information on calling stored procedures from C and C++ applications, see Section 11.3.3, "Calling Java Stored Procedures From a Multithreaded C or C++ Application".

11.3.1.4 Dropping Published Stored Procedures

Oracle Database Lite provides tools and SQL commands for dropping stored procedures. You should use caution when dropping procedures from the database, since Oracle Database Lite does not keep track of dependencies between classes. You must ensure that the stored procedure you drop is not referenced by other stored procedures. Dropping a class invalidates classes that depend on it directly or indirectly.

To remove Java stored procedure classes from Oracle Database Lite that were loaded using the load and publish method, use either of the following:

- Section 11.3.1.4.1, "Using dropjava" for directions on how to use the `dropjava` utility
- Section 11.3.1.4.2, "Using SQL Commands" for directions on how to use the `SQL DROP JAVA` statement

To drop call specifications, use either `DROP FUNCTION` or `DROP PROCEDURE`.

11.3.1.4.1 Using dropjava The `dropjava` command-line utility automates the task of dropping Java classes from Oracle Database Lite and Oracle databases. `dropjava` converts file names into the names of schema objects and drops the schema objects. Use the following syntax to invoke `dropjava`:

```
dropjava {-user | -u} username/password[@database]
  [-option] filename filename ...
```

Arguments

This section describes the arguments to `dropjava`.

User

The `user` argument specifies a username, password, and absolute path to the database file in the following format:

```
<user>/<password>[@<database>]
```

For example:

```
scott/tiger@c:\Olite\Mobile\Sdk\OLDB40\Polite.odb
```

Option

By specifying the verbose option (`-verbose | -v`), you can direct `dropjava` to produce detailed status messages while running.

Oracle database supports additional options. If used with Oracle Database Lite, the additional options are recognized but not supported. Using them does not result in an error.

For a complete list of supported and recognized options, from the command prompt type:

```
dropjava -help
```

Filename

For the `filename` argument, you can specify any number of Java class, source, JAR, ZIP, and resource files, in any order. JAR and ZIP files must be uncompressed. `dropjava` interprets most file names the same way `loadjava` does:

- For class files, `dropjava` finds the class name in the file and drops the corresponding schema object.
- For source files, `dropjava` finds the first class name in the file and drops the corresponding schema object.
- For JAR and ZIP files, `dropjava` processes the archived file names as if they had been entered on the command line.

If a file name has an extension other than **.java**, **.class**, **.jar**, or **.zip**, or has no extension, then `dropjava` assumes that the file name is the name of a schema object, then drops all source, class, and resource schema objects with that name. If `dropjava` encounters a file name that does not match the name of any schema object, it displays an error message and then processes the remaining file names.

11.3.1.4.2 Using SQL Commands To drop a Java class from Oracle Database Lite manually, use the `DROP JAVA` statement, which has the following syntax:

```
DROP JAVA { CLASS | RESOURCE } [<schema-name> .]<object_name>
```

To drop a call specification, use the `DROP FUNCTION` or `DROP PROCEDURE` statement:

```
DROP { FUNCTION | PROCEDURE } [<schema-name>.]<object_name>
```

The schema name, if specified, is recognized but ignored by Oracle Database Lite.

11.3.1.5 Example Using the Load and Publish Model

The following example creates a Java stored procedure using the load and publish model.

In this example, you store the Java method `paySalary` in the Oracle Database Lite. `paySalary` computes the take-home salary for an employee.

This example covers the following steps.

- Step 1: Create the Java Class
- Step 2: Load the Java Class into the Database
- Step 3: Publish the Function

- Step 4: Execute the Function

More examples of Java stored procedures are located in the *<ORACLE_HOME>*`\Mobile\SDK\samples\jdbc` directory.

Step 1: Create the Java Class

Create the Java class `Employee` in the file `Employee.java`. The `Employee` class implements the `paySalary` method:

```
import java.sql.*;
public class Employee {
   public static String paySalary(float sal, float fica, float sttax,
                         float ss_pct, float espp_pct)  {
      float deduct_pct;
      float net_sal;
        // compute take-home salary
      deduct_pct = fica + sttax + ss_pct + espp_pct;
      net_sal = sal * deduct_pct;
      String returnstmt = "Net salary is " + net_sal;
      return returnstmt;
   } // paySalary
}
```

Note: The keyword "public class" should not be used in a comment before the first public class statement.

Step 2: Load the Java Class into the Database

From mSQL, load the Java class using `CREATE JAVA`, as follows:

```
CREATE JAVA CLASS USING BFILE ('c:\myprojects\doc',
'Employee.class');
```

This command loads the Java class located in `c:\myprojects\doc` into the Oracle Database Lite.

Step 3: Publish the Function

Create a call spec for the `paySalary` method. The following call spec publishes the Java method `paySalary` as function `pay_salary`:

```
CREATE FUNCTION pay_salary (
sal float, fica float, sttax float, ss_pct float, espp_pct float)
RETURN VARCHAR2
AS LANGUAGE JAVA NAME
'Employee.paySalary(float, float, float, float, float)
return java.lang.String';
/
```

Step 4: Execute the Function

To execute `pay_salary` in mSQL:

```
SELECT pay_salary(6000.00, 0.2, 0.0565, 0.0606, 0.1)
FROM DUAL;
```

To execute `pay_salary` in ODBC:

```
SQLExecDirect(hstm,
   "SELECT pay_salary(6000.00,0.2,0.0565,0.0606,0.1)
   FROM DUAL);
```

Because the arguments to `pay_salary` are constants, the `FROM` clause specifies the dummy table `DUAL`. This `SELECT` statement produces the following output:

```
Net salary is 2502.6
```

11.3.2 Using the Attached Stored Procedure Development Model

You can create Java stored procedures by attaching classes to a table and invoking methods in the class by name. Using this model, you can store both class-level (static) methods and object-level (non-static) methods.

For this model, follow these steps:

1. Develop a Java class with the methods you want to store. Make sure that the class compiles and executes without errors.
2. Attach the class to a table using the `SQL ALTER TABLE` command. Once the class is attached, then the methods in the class become table-level or row-level stored procedures of the table.
3. Invoke methods in the class directly from SQL. Identify the method with `table_name.method_name`.

This information is specific to Oracle Database Lite; you cannot attach classes to Oracle database tables as described here. The load and publish model for developing stored procedures, described in Section 11.3.1, "Using the Load and Publish Stored Procedure Development Model", only supports class (static) methods. By attaching classes to tables, however, you can store and call Java class and instance methods.

The following sections describe the details for attaching and using Java stored procedures:

- Section 11.3.2.1, "Attaching a Java Class to a Table"
- Section 11.3.2.2, "Table-Level Stored Procedures"
- Section 11.3.2.3, "Row-Level Stored Procedures"
- Section 11.3.2.4, "Calling Attached Stored Procedures"
- Section 11.3.2.5, "Dropping Attached Stored Procedures"
- Section 11.3.2.6, "Example of An Attached Java Stored Procedure"

11.3.2.1 Attaching a Java Class to a Table

To attach a Java class to a table, use the SQL command `ALTER TABLE`, which has the following syntax:

```
ALTER TABLE [schema.]table
   ATTACH JAVA {CLASS|SOURCE} "cls_or_src_name "
   IN {DATABASE|'cls_or_src_path '}
   [WITH CONSTRUCTOR ARGS (col_name_list )]
```

Note: You can attach either a source file or a class file. Source files are compiled by the Java compiler found in the system path.

Where:

- The `cls_or_src_name` variable specifies a fully qualified name of a class or source file. This includes the package name followed by class name, such as

`Oracle.lite.Customer`. Do not include the file extension in the class or source file name. The name is case-sensitive. If you use lowercase letters, enclose the name in double quotes (" "). Make sure that the source or class is in the package specified by `cls_or_src_name`. For example, the source file of the example class `Customer` should contain the line "`package Oracle.lite;`". The class file is stored in the database in the same package. Oracle Database Lite creates the package if it does not already exist.

- If you have already attached the Java class to another table in the database, you can use the `IN DATABASE` clause. If the class has not yet been attached, specify the directory location of the class or source file in `cls_or_src_path`.
- Prior to executing a row-level stored procedure, Oracle Database Lite creates a Java object for the row, if one does not already exist. If the `ALTER TABLE` statement includes a `WITH CONSTRUCTOR` clause, then Oracle Database Lite creates the object using the class constructor that is the best match given the datatypes of the columns included in `col_name_list`. If the `ALTER TABLE` statement does not include a `WITH CONSTRUCTOR` clause, then Oracle Database Lite uses the default constructor.

You can use the ODBC functions `SQLProcedures` and `SQLProcedureColumns` to retrieve information about methods defined in a table.

11.3.2.2 Table-Level Stored Procedures

Table-level stored procedures are the static methods of the attached Java class. Therefore, when executing the method, Oracle Database Lite does not instantiate the class to which it belongs. In a call statement, you refer to table-level stored procedures as *table_name.method_name*.

Statement-level triggers and `BEFORE INSERT` and `AFTER DELETE` row-level triggers (see "Section 11.5.1, "Statement-Level vs. Row-Level Triggers") must be table-level stored procedures.

11.3.2.3 Row-Level Stored Procedures

Row-level stored procedures are the non-static methods in the attached Java class. To execute a row-level stored procedure, Oracle Database Lite instantiates the class to which the procedure belongs. The arguments to the class constructor determine which column values the constructor uses as parameters to create the class instances. In a call statement, you refer to row-level stored procedures as method_name (without the table qualifier). Row-level triggers can indirectly execute row-level stored procedures.

11.3.2.4 Calling Attached Stored Procedures

After attaching the class to a table using the `ALTER TABLE` statement, you can call it with a `SELECT` statement. Refer to table-level stored procedures as *table_name.method_name* and row-level procedures as *method_name*.

For example, to execute a table-level stored procedure:

```
SELECT table_name.proc_name[arg_list]
   FROM {DUAL|[schema.]table WHERE condition};
```

The `proc_name` is the name of the table-level stored procedure. Each argument in `arg_list` is either a constant or a reference to a column in the table. If all the arguments of `arg_list` are constants, the FROM clause should reference the dummy table DUAL.

Execute a row-level stored procedure as follows:

```
SELECT [schema.]proc_name[arg_list]
   FROM [schema.]table
   WHERE condition;
```

If you call a procedure in the form *table_name.method_name*, and a table or method with that name does not exist, Oracle Database Lite assumes that *table_name* refers to a schema name and *method_name* refers to a procedure name. If you reference *method_name* only, Oracle Database Lite assumes that the referenced method is a row-level procedure. If there is no such procedure defined, however, Oracle Database Lite assumes that *method_name* refers to a procedure in the current schema.

Note: Oracle Database Lite does not support the Oracle8*i* SQL `CALL` statement for invoking stored procedures.

You can use a callable statement to execute a procedure from ODBC or JDBC applications. For more information, see Chapter 10, "JDBC Programming" or Section 11.3.3, "Calling Java Stored Procedures From a Multithreaded C or C++ Application".

11.3.2.5 Dropping Attached Stored Procedures

Oracle Database Lite provides tools and SQL commands for dropping stored procedures. You should use caution when dropping procedures from the database, since Oracle Database Lite does not keep track of dependencies between classes. You must ensure that the stored procedure you drop is not referenced by other stored procedures. Dropping a class invalidates classes that depend on it directly or indirectly.

You use the `ALTER TABLE` command to drop stored procedures, which has the following syntax:

```
ALTER TABLE [schema.]table
   DETACH [AND DELETE] JAVA CLASS "class_name"
```

Note: You must enclose the class name in double quotes (`"  "`) if it contains lowercase letters.

Detaching the Java class does not delete it from the database. To delete the Java class file from the database, use the `DETACH AND DELETE` statement.

If you delete a Java class from the database after invoking it as a stored procedure or trigger, the class remains in the Java Virtual Machine attached to the application. To unload the class from the Java Virtual Machine, commit changes to the database, if necessary, and close all applications connected to the database. To replace a Java class, you must close all connections to the database and reload the class.

11.3.2.6 Example of An Attached Java Stored Procedure

The following example shows how to create a Java stored procedure in Oracle Database Lite. In this example, you attach the Java method `paySalary` to the table EMP. `paySalary` computes the take-home salary for an employee.

This example covers the following steps:

- Step 1: Create the Table
- Step 2: Create the Java Class

- Step 3: Attach the Java Class to the Table
- Step 4: Execute the Method

Step 1: Create the Table

Create the table using the following SQL command:

```
CREATE TABLE EMP(Col1 char(10));
```

Step 2: Create the Java Class

Create the Java class `Employee` in the file `Employee.java`. The `Employee` class implements the `paySalary` method:

```
import java.sql.*;
public class Employee {
   public static String paySalary(float sal, float fica, float sttax,
                                 float ss_pct, float espp_pct)  {
      float deduct_pct;
      float net_sal;
        // compute take-home salary
      deduct_pct = fica + sttax + ss_pct + espp_pct;
      net_sal = sal * deduct_pct;
      String returnstmt = "Net salary is " + net_sal;
      return returnstmt;
   } // paySalary
}
```

Step 3: Attach the Java Class to the Table

From mSQL, attach the Java class using the `ALTER TABLE` command:

```
ALTER TABLE EMP ATTACH JAVA SOURCE "Employee" IN 'C:\tmp';
```

This command attaches the Java source file for the `Employee` class, which resides in the directory `C:\tmp`, to the EMP table.

Step 4: Execute the Method

To execute the `paySalary` method in mSQL, type the following statement:

```
SELECT EMP."paySalary"(6000.00,0.2,0.0565,0.0606,0.1)
   FROM DUAL;
```

To execute `paySalary` from ODBC, invoke `SQLExecDirect`:

```
SQLExecDirect(hstm,
   "SELECT EMP.\"paySalary\"(6000.00,0.2,0.0565,0.0606,0.1)
   FROM DUAL);
```

This statement produces the following result:

```
Net salary is 2502.6
```

11.3.3 Calling Java Stored Procedures From a Multithreaded C or C++ Application

When invoking a Java stored procedure from a multithreaded C or C++ application, you should load `jvm.dll` from the application's `main` function. This resolves a problem that occurs with the Java Virtual Machine's garbage collection when a C or C++ application creates multiple threads that invoke a stored procedure directly or indirectly. The Java Virtual Machine runs out of memory because the threads do not detach from the Java Virtual Machine before exiting. Since Oracle Database Lite cannot determine whether the Java Virtual Machine or the user application created the thread, it does not attempt to detach them.

The `main` function should load the library before taking any other action, as follows:

```
int main (int argc, char** argv)
{
   LoadLibrary("jvm.dll");
   ...
}
```

The library loads the Java Virtual Machine into the application's main thread. It attempts to detach any thread from the Java Virtual Machine if the thread detaches from the process. The `jvm.dll` behaves correctly even if the thread is not attached to a Java Virtual Machine.

11.4 Converting Datatypes Between Java and SQL For Stored Procedures

Oracle Database Lite performs type conversion between Java and SQL datatypes according to standard SQL rules. For example, if you pass an integer to a stored procedure that takes a string, Oracle Database Lite converts the integer to a string. For information about row-level triggers arguments, see Section 11.5.5, "Trigger Arguments". For a complete list of Java to SQL datatype mappings, see Section 10.5.1, "Mapping Datatypes Between Java and Oracle".

Note: In Oracle database, `DATE` columns are created as `TIMESTAMP`. Also, note that `TIMESTAMP WITH TIME ZONE` data type is not supported.

You must specify trigger methods accordingly.

Java does not allow a method to change the value of its arguments outside the scope of the method. However, Oracle Database Lite supports `IN`, `OUT`, and `IN/OUT` parameters.

Many Java datatypes are immutable or do not support `NULL` values. To pass `NULL` values and use `IN/OUT` parameters for those datatypes, a stored procedure can use an array of that type or use the equivalent object type. Table 11–2 shows the Java integer datatypes you can use to enable an integer to be an `IN/OUT` parameter or carry a `NULL` value.

Table 11–2 The Java Integer Datatypes

Java Argument	Can Be IN/OUT	Can Be NULL
`int`	No	No
`int[]`	Yes	Yes
`Integer`	No	Yes
`Integer[]`	Yes	Yes
`int[][]`	Yes	Yes

You can use mutable Java datatypes, such as `Date`, to pass a `NULL` or an `IN/OUT` parameter. However, use a `Date` array if a stored procedure needs to change the `NULL` status of its argument.

Note: Passing a NULL when the corresponding Java argument cannot be NULL causes an error.

11.4.1 Declaring Parameters for Java Stored Procedures

The return value of a Java method is the OUT parameter of the procedure. A primitive type or immutable reference type can be an IN parameter. A mutable reference type or array type can be an IN/OUT parameter. Table 11–3 shows the Java type to use to make the corresponding Oracle Database Lite parameter an IN/OUT parameter.

Table 11–3 Java Types for Oracle Database Lite IN/OUT Parameters

For IN/OUT parameters of type...	Use...
Number	`Integer[]` or `int[]`
Binary	`byte[]` or `byte[][]`
String	`string[]`

If the stored procedure takes a `java.sql.Connection`, Oracle Database Lite automatically supplies the argument using the value of the current transaction or row. This argument is the first argument passed to the procedure.

11.4.2 Using Stored Procedures to Return Multiple Rows

You can use stored procedures to return multiple rows. You can invoke stored procedures that return multiple rows only from JDBC or ODBC applications, however. For a stored procedure to return multiple rows, its corresponding Java method must return a `java.sql.ResultSet` object. By executing a SELECT statement, the Java method obtains a `ResultSet` object to return. The column names of the `ResultSet` are specified in the SELECT statement. If you need to address the result columns by different names than those used in the table, the SELECT statement should use aliases for the result columns. For example:

```
SELECT emp.name Name,  dept.Name Dept
   FROM  emp, dept
   WHERE emp.dept# = dept.dept#;
```

Because the return type of a stored procedure that returns multiple rows must be `java.sql.ResultSet`, the signature of that stored procedure cannot be used to obtain the column names or types of the result. Consequently, you should design additional tables to track the column names or result types for the stored procedures. For example, if you embed the preceding SELECT statement in a Java method, the method return type should be `java.sql.ResultType`, not `char Name` and `char Dept`.

Note: You can only create Java stored procedures that return multiple rows using the attached stored procedure development model, described in Section 11.3.2, "Using the Attached Stored Procedure Development Model".

11.4.2.1 Returning Multiple Rows in ODBC

To execute a stored procedure that returns multiple rows in an OBDC application, use the following CALL statement, in which P is the name of the stored procedure and a_1 through a_n are arguments to the stored procedure.

{CALL P(a_1,...,a_n)}

You use a marker (?) for any argument that should be bound to a value before the statement executes. When the statement executes, the procedure runs and the cursor on the result set is stored in the statement handle. Subsequent fetches using this statement handle return the rows from the procedure.

After you execute the CALL statement, use SQLNumResultCols to find the number of columns in each row of the result. Use the SQLDescribeCol function to return the column name and datatype.

11.4.2.2 Example

The following example shows how to use ODBC to execute a stored procedure that returns multiple rows. This example does not use the SQLNumResultCols or SQLDescribeCol functions. It assumes that you have created a stored procedure, which you have published to SQL as PROC. PROC takes an integer as an argument.

```
rc = SQLPrepare(StmtHdl, "{call PROC(?)}", SQL_NTS);
CHECK_STMT_ERR(StmtHdl, rc, "SQLPrepare");

rc = SQLBindParameter(StmtHdl, 1, SQL_PARAM_INPUT_OUTPUT,
   SQL_C_LONG,SQL_INTEGER, 0, 0, &InOutNum, 0, NULL);
CHECK_STMT_ERR(StmtHdl, rc, "SQLBindParameter");

rc = SQLExecute(StmtHdl);
CHECK_STMT_ERR(StmtHdl, rc, "SQLExecute");

/* you can use SQLNumResultCols and SQLDescribeCol here */

rc = SQLBindCol(StmtHdl, 1, SQL_C_CHAR, c1, 20, &pcbValue1);
CHECK_STMT_ERR(StmtHdl, rc, "SQLBindCol");

rc = SQLBindCol(StmtHdl, 2, SQL_C_CHAR, c2, 20, &pcbValue2);
CHECK_STMT_ERR(StmtHdl, rc, "SQLBindCol");

while ((rc = SQLFetch(StmtHdl)) != SQL_NO_DATA_FOUND) {
   CHECK_STMT_ERR(StmtHdl, rc, "SQLFetch");
   printf("%s, %s\n", c1, c2);
}
```

11.5 Using Triggers With Java Stored Procedures

Triggers are stored procedures that execute, or "fire", when a specific event occurs. A trigger can fire when a column is updated, or when a row is added or deleted. The trigger can fire before or after the event.

Triggers are commonly used to enforce a database's business rules. For example, a trigger can verify input values and reject an illegal insert. Similarly, a trigger can ensure that all tables depending on a particular row are brought to a consistent state before the row is deleted.

- Section 11.5.1, "Statement-Level vs. Row-Level Triggers"
- Section 11.5.2, "Creating Triggers"

- Section 11.5.3, "Dropping Triggers"
- Section 11.5.4, "Trigger Example"
- Section 11.5.5, "Trigger Arguments"
- Section 11.5.6, "Trigger Arguments Example"

11.5.1 Statement-Level vs. Row-Level Triggers

There are two types of triggers: row-level and statement-level. A row-level trigger is fired once for each row affected by the change to the database. A statement-level trigger fires only once, even if multiple rows are affected by the change.

The BEFORE INSERT and AFTER DELETE triggers can only fire table-level stored procedures, since a row object cannot be instantiated to call the procedures. The AFTER INSERT, BEFORE DELETE, and UPDATE triggers may fire table-level or row-level stored procedures.

11.5.2 Creating Triggers

Use the CREATE TRIGGER statement to create a trigger. The CREATE TRIGGER statement has the following syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name {BEFORE | AFTER} [{INSERT | DELETE |
   UPDATE [OF column_list]} [OR ]] ON table_reference
   [FOR EACH ROW] procedure_ref
   (arg_list)
```

In the CREATE TRIGGER syntax:

- Use the OR clause to specify multiple triggering events.
- Use FOR EACH ROW to create a row-level trigger. For a table-level trigger, do not include this clause.
- Use `procedure_ref` to identify the stored procedure to execute.

You can create multiple triggers of the same kind for a table if each trigger has a unique name within a schema.

In the following example, assume that you have stored and published a procedure as PROCESS_NEW_HIRE. The trigger AIEMP fires every time a row is inserted into the EMP table.

```
CREATE TRIGGER AIEMP AFTER INSERT ON EMP FOR EACH ROW
   PROCESS_NEW_HIRE(ENO);
```

UPDATE triggers that use the same stored procedure for different columns of a table are fired only once when a subset of the columns is modified within a statement. For example, the following statement creates a BEFORE UPDATE trigger on table T, which has columns C1, C2, and C3:

```
CREATE TRIGGER T_TRIGGER BEFORE UPDATE OF C1,C2,C3 ON T
   FOR EACH ROW trigg(old.C1,new.C1,old.C2,new.C2,
   old.C3,new.C3);
```

This update statement fires `T_TRIGGER` only once:

```
UPDATE T SET C1 = 10, C2 = 10 WHERE ...
```

11.5.2.1 Enabling and Disabling Triggers

When you create a trigger, it is automatically enabled. To disable triggers, use the ALTER TABLE or ALTER TRIGGER statement.

To enable or disable individual triggers, use the ALTER TRIGGER statement, which has the following syntax:

```
ALTER TRIGGER <trigger_name> {ENABLE | DISABLE}
```

To enable or disable all triggers attached to a table, use ALTER TABLE:

```
ALTER TABLE <table_name> {ENABLE | DISABLE} ALL TRIGGERS
```

11.5.3 Dropping Triggers

To drop a trigger, use the DROP TRIGGER statement, which has the following syntax:

```
DROP TRIGGER [schema.]trigger
```

11.5.4 Trigger Example

This example creates a trigger. It follows the development model described in Section 11.3.2, "Using the Attached Stored Procedure Development Model". For an example of creating triggers using the load and publish model, see Section 11.5.6, "Trigger Arguments Example". In the example, you first create a table and a Java class. Then you attach the class to the table. And finally, you create and fire the trigger.

The `SalaryTrigger` class contains the `check_sal_raise` method. The method prints a message if an employee gets a salary raise of more than ten percent. The trigger fires the method before updating a salary in the EMP table.

Since `check_sal_raise` writes a message to standard output, use mSQL to issue the mSQL commands in the example. To start mSQL, invoke the Command Prompt and enter the following.

```
msql username/password@connect_string
```

`connect_string` is JDBC URL syntax. For example, to connect to the default database as user SYSTEM, at the Command Prompt.

```
msql system/passwd@jdbc:polite:polite
```

At the mSQL command line, create and populate the EMP table as follows.

```
CREATE TABLE EMP(E# int, name char(10), salary real,
   Constraint E#_PK primary key (E#));

INSERT INTO EMP VALUES (123,'Smith',60000);
INSERT INTO EMP VALUES (234,'Jones',50000);
```

Place the following class in **SalaryTrigger.java**:

```
class SalaryTrigger {
   private int eno;
   public SalaryTrigger(int enum) {
      eno = enum;
   }
   public void check_sal_raise(float old_sal,
      float new_sal)
   {
      if (((new_sal - old_sal)/old_sal) > .10)
      {
            // raise too high  do something here
            System.out.println("Raise too high for employee " + eno);
        }
    }
}
```

The `SalaryTrigger` class constructor takes an integer, which it assigns to attribute `eno` (the employee number). An instance of `SalaryTrigger` is created for each row (that is, for each employee) in the table `EMP`.

The `check_sal_raise` method is a non-static method. To execute, it must be called by an object of its class. Whenever the salary column of a row in `EMP` is modified, an instance of `SalaryTrigger` corresponding to that row is created (if it does not already exist) with the employee number (*E#)* as the argument to the constructor. The trigger then calls the `check_sal_raise` method.

After creating the Java class, you attach it to the table, as follows:

```
ALTER TABLE EMP ATTACH JAVA SOURCE "SalaryTrigger" IN '.'
   WITH CONSTRUCTOR ARGS(E#);
```

This statement directs Oracle Database Lite to compile the Java source file `SalaryTrigger.java` found in the current directory, and attach the resulting class to the EMP table. The statement also specifies that, when instantiating the class, Oracle Database Lite should use the constructor that takes as an argument the value in the `E#` column.

After attaching the class to the table, create the trigger as follows:

```
CREATE TRIGGER CHECK_RAISE BEFORE UPDATE OF SALARY ON EMP FOR EACH ROW
   "check_sal_raise"(old.salary, new.salary);
/
```

This statement creates a trigger called `check_raise`, which fires the `check_sal_raise` method before any update to the salary column of any row in EMP. Oracle Database Lite passes the old value and the new value of the salary column as arguments to the method.

In the example, a row-level trigger fires a row-level procedure (a non-static method). A row-level trigger can also fire table-level procedures (static methods). However, because statement-level triggers are fired once for an entire statement and a statement may affect multiple rows, a statement-level trigger can only fire a table-level procedure.

The following command updates the salary and fires the trigger:

```
UPDATE EMP SET SALARY = SALARY + 6100  WHERE E# = 123;
```

This produces the following output:

```
Raise too high for employee 123
```

11.5.5 Trigger Arguments

If using attached stored procedures, as described in Section 11.3.2, "Using the Attached Stored Procedure Development Model", row-level triggers do not support Java-to-SQL type conversion. Therefore, the Java datatype of a trigger argument must match the corresponding SQL datatype (shown in section Section 11.4, "Converting Datatypes Between Java and SQL For Stored Procedures") of the trigger column. However, if you are using the load and publish model, Oracle Database Lite supports datatype casting.

Table 11–4 describes how trigger arguments work in each type of column.

Table 11–4 Trigger Arguments

Trigger Argument	New Column Access	Old Column Access
`insert`	Yes	No
`delete`	No	Yes
`update`	Yes	Yes

Note: Triggers that have a `java.sql.Connection` object as an argument may be used only with applications that use the relational model.

11.5.6 Trigger Arguments Example

The following example shows how to create triggers that use IN/OUT parameters.

1. First, create the Java class `EMPTrigg`.

```
import java.sql.*;

public class EMPTrigg {
   public static final String goodGuy = "Oleg";

   public static void NameUpdate(String oldName, String[] newName)
   {
      if (oldName.equals(goodGuy))
         newName[0] = oldName;
   }

   public static void SalaryUpdate(String name, int oldSalary,
          int newSalary[])
   {
      if (name.equals(goodGuy))
         newSalary[0] = Math.max(oldSalary, newSalary[0])*10;
   }

   public static void AfterDelete(Connection conn,
                  String name, int salary) {
      if (name.equals(goodGuy))
         try {
            Statement stmt = conn.createStatement();
            stmt.executeUpdate(
             "insert into employee values('" + name + "', " +
                       salary + ")");
            stmt.close();
         } catch(SQLException e) {}
     }
 }
```

2. Create a new table EMPLOYEE and populate it with values.

```
CREATE TABLE EMPLOYEE(NAME VARCHAR(32), SALARY INT);
INSERT INTO EMPLOYEE VALUES('Alice', 100);
INSERT INTO EMPLOYEE VALUES('Bob', 100);
INSERT INTO EMPLOYEE VALUES('Oleg', 100);
```

3. Next, load the class into Oracle Database Lite.

```
CREATE JAVA CLASS USING BFILE ('c:\myprojects', 'EMPTrigg.class');
```

4. Use the CREATE PROCEDURE statement to publish the EMPTrigg methods that you want to call:

```
CREATE PROCEDURE NAME_UPDATE(
   OLD_NAME IN VARCHAR2, NEW_NAME IN OUT VARCHAR2)
   AS LANGUAGE JAVA NAME
   'EMPTrigger.NameUpdate(java.lang.String, java.lang.String[])';
   /

CREATE PROCEDURE SALARY_UPDATE(
   ENAME VARCHAR2, OLD_SALARY INT, NEW_SALARY IN OUT INT)
   AS LANGUAGE JAVA NAME
   'EMPTrigger.SalaryUpdate(java.lang.String, int, int[])';
   /

CREATE PROCEDURE AFTER_DELETE(
   ENAME VARCHAR2, SALARY INT)
   AS LANGUAGE JAVA NAME
   'EMPTrigger.AfterDelete(java.sql.Connection,
           java.lang.String, int)';
   /
```

5. Now, create a trigger for each procedure:

```
CREATE TRIGGER NU BEFORE UPDATE OF NAME ON EMPLOYEE FOR EACH ROW
   NAME_UPDATE(old.name, new.name);

CREATE TRIGGER SU BEFORE UPDATE OF SALARY ON EMPLOYEE FOR EACH ROW
   SALARY_UPDATE(name, old.salary, new.salary);

CREATE TRIGGER AD AFTER DELETE ON EMPLOYEE FOR EACH ROW
   AFTER_DELETE(name, salary);
```

6. Enter the following commands to fire the triggers and view the results:

```
SELECT * FROM EMPLOYEE;
UPDATE EMPLOYEE SET SALARY=0 WHERE NAME = 'Oleg';
SELECT * FROM EMPLOYEE;

DELETE FROM EMPLOYEE WHERE NAME = 'Oleg';
SELECT * FROM EMPLOYEE;

UPDATE EMPLOYEE SET NAME='TEMP' WHERE NAME = 'Oleg';
DELETE FROM EMPLOYEE WHERE NAME = 'TEMP';

SELECT * FROM EMPLOYEE;
```

11.6 Creating a Java Stored Procedure That Is Invoked With a Trigger

In this tutorial, you create a Java class EMAIL, load the class into Oracle Database Lite, publish its method to SQL, and create a trigger for the method. The EMAIL class appears in the source file EMAIL.java, and is available in the Java examples directory at the following location.

<ORACLE_HOME>\Mobile\Sdk\Samples\JDBC

EMAIL has a method named assignEMailAddress, which generates an email address for an employee based on the first letter of the employee's first name and up to seven letters of the last name. If the address is already assigned, the method attempts to find a unique email address using combinations of letters in the first and last name.

After creating the class, you load it into Oracle Database Lite using mSQL. For this example you use the SQL statement `CREATE JAVA`. Alternatively, you can use the `loadjava` utility to load the class into Oracle Database Lite. After loading the class, you publish the `assignEMailAddress` method to SQL.

Finally, you create a trigger that fires the `assignEMailAddress` method whenever a row is inserted into `T_EMP`, the table that contains the employee information.

As arguments, `assignEMailAddress` takes a JDBC connection object, the employee's identification number, first name, middle initial, and last name. Oracle Database Lite supplies the JDBC connection object argument. You do not need to provide a value for the connection object when you execute the method. `assignEMailAddress` uses the JDBC connection object to ensure that the generated e-mail address is unique.

- Section 11.6.1, "Start mSQL"
- Section 11.6.2, "Create a Table"
- Section 11.6.3, "Create a Java Class"
- Section 11.6.4, "Load the Java Class File"
- Section 11.6.5, "Publish the Stored Procedure"
- Section 11.6.6, "Populate the Database"
- Section 11.6.7, "Execute the Procedure"
- Section 11.6.8, "Verify the Email Address"
- Section 11.6.9, "Create a Trigger"
- Section 11.6.10, "Commit or Roll Back"

11.6.1 Start mSQL

Start mSQL and connect to the default Oracle Database Lite. Since the Java application in this tutorial prints to standard output, use the DOS version of mSQL. From a DOS prompt, type:

```
msql system/mgr@jdbc:polite:polite
```

The SQL prompt should appear.

11.6.2 Create a Table

To create a table, type:

```
CREATE TABLE T_EMP(ENO INT PRIMARY KEY,
   FNAME VARCHAR(20),
   MI CHAR,
   LNAME VARCHAR(20),
   EMAIL VARCHAR(8));
```

11.6.3 Create a Java Class

Create and compile the Java class `EMAIL` in the file `EMAIL.java` in `C:\tmp`. `EMAIL.java` implements the `assignEMailAddress` method. The code sample given below lists the contents of this file. You can copy this file from the following location.

<ORACLE_HOME>\Mobile\Sdk\Samples\JDBC

```
import java.sql.*;

public class EMAIL {
   public static void assignEMailAddress(Connection conn,
            int eno, String fname,String lname)
            throws Exception
   {
      Statement stmt = null;
      ResultSet retset = null;
      String emailAddr;
      int i,j,fnLen, lnLen, rowCount;

      /* create a statement */
      try {
         stmt = conn.createStatement();
      }
      catch (SQLException e)
      {
         System.out.println("conn.createStatement failed: " +
         e.getMessage() + "\n");
         System.exit(0);
      }
      /* check fname and lname */
      fnLen = fname.length();
      if(fnLen > 8) fnLen = 8;
      if (fnLen == 0)
         throw new Exception("First name is required");
      lnLen = lname.length();
      if(lnLen > 8) lnLen = 8;
      if (lnLen == 0)
         throw new Exception("Last name is required");
      for (i=1; i <= fnLen; i++)
      {
         /* generate an e-mail address */
         j = (8-i) > lnLen? lnLen:8-i;
         emailAddr =
               new String(fname.substring(0,i).toLowerCase()+
               lname.substring(0,j).toLowerCase());
         /* check if this e-mail address is unique  */
         try {
            retset = stmt.executeQuery(
                   "SELECT * FROM T_EMP  WHERE email = '"+
                   emailAddr+"'");
            if(!retset.next()) {
               /* e-mail address is unique;
               * so update the email column */
               retset.close();
               rowCount = stmt.executeUpdate(
                   "UPDATE T_EMP SET EMAIL = '"
                   + emailAddr + "' WHERE ENO = "
                   + eno);
               if(rowCount == 0)
                  throw new Exception("Employee "+fname+ " " +
                          lname + " does not exist");
               else return;
            }
         }
         catch (SQLException e) {
            while(e != null) {
               System.out.println(e.getMessage());
               e = e.getNextException();
            }
         }
      }
      /* Can't find a unique name */
      emailAddr = new String(fname.substring(0,1).toLowerCase() +
           lname.substring(0,1).toLowerCase() + eno);
      rowCount = stmt.executeUpdate(
           "UPDATE T_EMP SET EMAIL = '"
           + emailAddr + "' WHERE ENO = "
           + eno);
      if(rowCount == 0)
         throw new Exception("Employee "+fname+ " " +
              lname + " does not exist");
      else return;
   }
}
```

11.6.4 Load the Java Class File

To load the `EMAIL` class file into Oracle Database Lite, type:

```
CREATE JAVA CLASS USING BFILE
   ('c:\tmp', 'EMAIL.class');
```

If you want to make changes to the class after loading it, you need to:

1. Drop the class from the database, using `dropjava` or `DROP JAVA CLASS`
2. Commit your work
3. Exit mSQL
4. Restart mSQL

This unloads the class from the Java Virtual Machine.

11.6.5 Publish the Stored Procedure

You make the stored procedure callable from SQL by creating a call specification (call spec) for it. Since `assignEMailAddress` does not return a value, use the CREATE PROCEDURE command, as follows:

```
CREATE OR REPLACE PROCEDURE
   ASSIGN_EMAIL(E_NO INT, F_NAME VARCHAR2, L_NAME VARCHAR2)
   AS LANGUAGE JAVA NAME 'EMAIL.assignEMailAddress(java.sql.Connection,
int, java.lang.String,
     java.lang.String)';
```

11.6.6 Populate the Database

Insert a row into `T_EMP`:

```
INSERT INTO T_EMP VALUES(100,'John','E','Smith',null);
```

11.6.7 Execute the Procedure

To execute the procedure, type:

```
SELECT ASSIGN_EMAIL(100,'John','Smith')
  FROM dual
```

11.6.8 Verify the Email Address

To see the results of the ASSIGN_EMAIL procedure, type:

```
SELECT * FROM T_EMP;
```

This command produces the following output:

```
     ENO  FNAME                M LNAME                EMAIL
     ---- ------------------   - -------------------- --------
     100  John                 E Smith                jsmith
```

11.6.9 Create a Trigger

To make ASSIGN_EMAIL execute whenever a row is inserted into T_EMP, create an AFTER INSERT trigger for it. Create the trigger as follows:

```
CREATE TRIGGER EMP_TRIGG AFTER INSERT ON T_EMP FOR EACH ROW
  ASSIGN_EMAIL(eno,fname,lname);
```

A trigger named EMP_TRIGG fires every time a row is inserted into T_EMP. The actual arguments for the procedure are the values of the columns eno, fname, and lname.

You do not need to specify a connection argument.

11.6.9.1 Testing the Trigger

Test the trigger by inserting a row into T_EMP:

```
INSERT INTO T_EMP VALUES(200,'James','A','Smith',null);
```

11.6.9.2 Verify the Email Address

Issue a SELECT statement to verify that the trigger has fired:

```
SELECT * FROM T_EMP;
   ENO FNAME                M LNAME                EMAIL
   --- -------------------- - -------------------- --------
   100 John                 E Smith                jsmith
   200 James                A Smith                jasmith
```

11.6.10 Commit or Roll Back

Finally, commit your changes to preserve your work, or roll back to cancel changes.

11.7 Executing Java Stored Procedures from JDBC

After creating a Java stored procedures, you can execute the procedure from a JDBC application by performing one of the following:

- Pass a SQL SELECT string, which executes the stored procedure, to the Statement.executeQuery method.
- Use a JDBC CallableStatement.

The executeQuery method executes table-level and row-level stored procedures. CallableStatement currently only supports execution of table-level stored procedures.

11.7.1 Using the executeQuery Method

To call a stored procedure using the executeQuery method, perform the following:

1. Create a `Statement` object and assign the value returned by the `createStatement` method with the current connection object.
2. Execute the `Statement.executeQuery` method, passing the `SQL SELECT` string that invokes the Java stored procedure.

The following example executes a row-level procedure `SHIP` on a table named `INVENTORY` with the argument value stored in the variable *q*. The variable `p` contains the product ID for the product (row) for which you want to execute the stored procedure.

```
int res = 0;
Statement s = conn.createStatement();
ResultSet r = s.executeQuery("SELECT SHIP(" + q + ")" +
   "FROM INVENTORY WHERE PID = " + p);
if(r.next()) res = r.getInt(1);
r.close();
s.close();
return res;
```

If you need to execute a procedure repeatedly with varying parameters, use `PreparedStatement` instead of `Statement`. Because the SQL statements in a `PreparedStatement` are pre-compiled, a `PreparedStatement` executes more efficiently. Additionally, a `PreparedStatement` can accept `IN` parameters, represented in the statement with a question mark `(?)`. However, if the `PreparedStatement` takes a `long` type parameter, such as `LONG` or `LONG RAW`, you must bind the parameter using the `setAsciiStream`, `setUnicodeStream`, or `setBinaryStream` methods.

In the preceding example, if the `SHIP` procedure updates the database and the isolation of the transaction that issues the above query is `READ COMMITTED`, then you must append the `FOR UPDATE` clause to the `SELECT` statement, as follows:

```
"SELECT SHIP(" + q + ")" +
   FROM INVENTORY WHERE PID = " +
   p + "FOR UPDATE");
```

11.7.2 Using a Callable Statement

To execute the stored procedure using a callable statement, create a `CallableStatement` object and register its parameters, as follows:

```
CallableStatement cstmt = conn.prepareCall(
   "{?=call tablename.methodname() }");
cstmt.registerOutParameter(1, ...);
cstmt.executeUpdate();
cstmt.get..(1);
cstmt.close();
```

The following restrictions apply to JDBC callable statements:

- JDBC callable statements can only execute table-level stored procedures.
- Both IN and OUT parameters are supported. However, not all Java datatypes can be used as OUT parameters. For more information, see Section 11.4, "Converting Datatypes Between Java and SQL For Stored Procedures".
- Procedure names correspond to the Java method names, and are case-sensitive.
- As with prepared statements, if the callable statement has a "`long`" type, such as: `LONG`, `LONG VARBINARY`, `LONG VARCHAR`, `LONG VARCHAR2`, or `LONG RAW`, you

must bind the parameter using the `setAsciiStream`, `setUnicodeStream`, or `setBinaryStream` methods.

Note: When no longer needed, you should reclaim system resources by closing JDBC objects, such as `Resultset` and `Statement` objects.

11.8 Using C++ Stored Procedures

A C++ stored procedure is a C++ procedure or function that exists in a DLL outside of Oracle Database Lite. The procedure can be invoked by applications that access the database. C++ stored procedures can return a single value, a row, or multiple rows.

The following sections describe how to create, build, and publish a C++ stored procedure:

- Section 11.8.1, "Creating C++ Stored Procedures"
- Section 11.8.2, "Building Your C++ Stored Procedures"
- Section 11.8.3, "Publish Your C++ Stored Procedure"
- Section 11.8.4, "C++ Stored Procedure Example"

11.8.1 Creating C++ Stored Procedures

When you are creating a C++ stored procedure, you use SODA APIs to access the database and transaction objects. This section demonstrates how to develop your C++ stored procedures.

- Section 11.8.1.1, "C++ Stored Procedure Include File and Procedure Definition"
- Section 11.8.1.2, "Access SODA Objects Within Your C++ Stored Procedure"

11.8.1.1 C++ Stored Procedure Include File and Procedure Definition

When you create the C++ source file, remember to do the following:

- Include the `olcsp.h` include file.
- The following defines the stored procedure or function prototypes. For each, you can have up to 32 parameters.
  - C++ stored procedure prototype:

    ```
    OL_CSP_CALL void cproc (const DBData &d1, const DBData &d2, ... , DBData
    &dN)
    ```

  - C++ stored function prototype:

    ```
    OL_CSP_CALL DBData cproc (const DBData &d1, const DBData &d2, ... , DBData
    &dN)
    ```

- Use `OL_CSP_CALL` before all of your procedures and functions, as it defines these as `extern "C" __declspec(dllexport)`. This enables the procedures and functions to be called from outside the DLL. For example, the following `sum` procedure uses this declaration:

  ```
  OL_CSP_CALL void sum (const DBData &a, const DBData &b, DBData &r)
  ```

- You can use the `DBData` object to represent almost any database type. In addition, it is easily cast to the correct datatype. For input parameters in the procedures, you can use `const DBData &`. For input/output parameters, use `DBData &` as the definition of the parameter.

Once inside the procedure, cast the parameters as shown below:

```
OL_CSP_CALL void sum (const DBData &a, const DBData &b, DBData &r)
{  r = (int)a + (int)b; }
```

11.8.1.2 Access SODA Objects Within Your C++ Stored Procedure

We use SODA, instead of ODBC, to provide a reliable access to the database and transaction objects. To access the SODA API objects, use the methods defined in the `olcsp.h` include file, as follows:

Note: For details on the SODA API, see Chapter 12, "Using Simple Object Data Access (SODA) for PocketPC Platforms".

- After you retrieve the session object, do not close the connection in side the procedure.
- Retrieve the SODA API `DBSession` object with the `olCSPGetSession()` method, as follows:

  ```
  DBSession &sess = olCSPGetSession();
  ```

- Retrieve the SODA API `DBSqlSession` object, which is used in preparing and executing SQL statements, with the `olCSPGetSqlSession` method. Once you retrieve the `DBSqlSession` object, you can prepare the SQL statement within a `DBSqlStmt` object. The returned `DBSqlSession` object is created based on the existing ODBC handle. The following example retrieves the SQL Session, prepares and executes a statement:

  ```
  DBSqlSession sess = olCSPGetSqlSession();
  DBSqlStmt stmt = sess.prepare("insert into testsql values(?)");
  stmt.execute(DBDataList() << v);
  ```

- If this procedure was executed within the trigger, you can retrieve the object on which the trigger was invoked with the `olCSPGetObject` method. This returns a `DBObject` of the trigger object. This will not work for BEFORE CREATE or AFTER DELETE triggers.
- If you want to use ODBC instead of SODA, you can retrieve the ODBC connection handle with the `olCSPGetODBCHdl` method.
- All C++ stored procedures can throw `DBException`. This exception is automatically thrown if any SODA/SODASQL operation fails inside your stored procedure. If you are using a WinCE device, then you must use the ALE library for exceptions. See the ALE documentation for more information.

11.8.2 Building Your C++ Stored Procedures

You can either build your stored procedure manually or by using the `olsp.mak` makefile. The following describes both processes:

- Section 11.8.2.1, "Linking in Appropriate Libraries"
- Section 11.8.2.2, "Automatically Build Your Stored Procedure"
- Section 11.8.2.3, "Manually Building Your Stored Procedure"

11.8.2.1 Linking in Appropriate Libraries

When you build your procedure, link in one or more of the following libraries:

- For all builds, link in `olobj40.lib`, which exists in *`<Mobile_Server>`*`Mobile/SDK/lib`.
- If you are using SODASQL in your stored procedure, then link in `sodasql.lib`.

11.8.2.2 Automatically Build Your Stored Procedure

If you have only a single source file, then you can use the `olsp.mak` makefile to build. The resulting DLL is named the same as the source file. This makefile supports building procedures for both the desktop and Windows CE platforms. Set up the following within the makefile before you execute to ensure a proper build:

1. Place the `olsp.mak` makefile in the same location as your source file.
2. If you build C++ stored procedures for a Windows CE device, then before you execute the makefile, you need to run the batch file from embedded visual C++. This sets the appropriate build environment for your windows CE platform. The following example shows execution of the `wcearmv4.bat` batch file:

   ```
    C:\EVC4.0\EVC\wce420\bin\WCEARMV4.BAT
   ```

3. Set the `MOBILESDK` environment variable—which defines the Oracle Lite Mobile SDK directory. For example,

   ```
   MOBILESDK=C:\oracle\ora90\mobile\sdk
   ```

4. Within the makefile, define `CDEFINE` (compiler defines) and `LFL` (linker flags) macros for your Windows CE platform.
5. If you are building .Net procedures, then set the `NETFRKDIR` environment variable in the makefile to point to your .Net Framework directory
6. If building for the Compact Framework, then define the `CFK` macro and set the `CFSDKDIR` environment variable in the makefile to point to your Compact Framework SDK directory.
7. Execute the makefile, as follows:

   ```
   nmake -f olsp.mak MyProc.dll [macros] [options]
   ```

   The macros that you can use are as follows:

   - For debug mode, use the macro `DEBUG=/DDEBUG`.
   - For compact framework, use the macro `CFK=1`.

8. Copy the new DLL, such as `MyProc.dll`, into a directory on the system path. If your platform is a WinCE device, copy this DLL into the `\Windows` directory.

For example, if your source file is `MyProc.cpp` or `MyProc.cs`, then this makefile builds `MyProc.dll` for you.

11.8.2.3 Manually Building Your Stored Procedure

If you have more than one source file for the stored procedure, then you must manually build. Keep in mind the following:

1. Because you are using the `export "C"` declaration on the stored procedure, which supports the procedure being able to throw exceptions, use the `/EHc-` compiler flag when building the stored procedure.

2. If you are using the `SODASQL` in the procedure, then link with either the `olobj40.lib` and `sodasql.lib`.

11.8.3 Publish Your C++ Stored Procedure

Publish the methods in the class that you want to call from SQL with a call specification, which is created with either the `CREATE FUNCTION` or `CREATE PROCEDURE` commands.

Perform the following to publish C++ stored procedures:

1. Publish any methods in the C++ class that you want to make accessible to SQL by creating call specifications for these methods. By publishing a method, you associate a SQL name to the method. SQL applications use this name to invoke the method.
2. Invoke the stored procedure through a SQL DML statement.

Publish any static method in the class that you want to call from SQL by creating a call specification for it. The call spec maps the method's name, parameter types, and return types to SQL counterparts.

To create a call spec, use the SQL commands `CREATE FUNCTION` for methods that return a value or `CREATE PROCEDURE` for methods that do not return a value. The `CREATE FUNCTION` and `CREATE PROCEDURE` statements have the following syntax:

```
CREATE [OR REPLACE]
   { PROCEDURE <proc_name> [([<sql_parms>])] |
   FUNCTION <func_name> [([<sql_parms>])]
   RETURN <datatype> }
   AS LANGUAGE CPLUSPLUS NAME
   '<lib_name>::<func_name>)';
   /
```

Where:

- `<proc_name>` is a SQL procedure name; `<func_name>` is the name of the function in the DLL used for this procedure.
- `<sql_parms>` can be a maximum of 32 arguments. All arguments passed to the procedures are given as DBData values to the function, which must cast the arguments to the appropriate data type. The syntax has the following format:

  ```
  <arg_name> [IN | OUT | IN OUT] <datatype>
  ```

- `<datatype>` is the datatype.
- `<lib_name>` is the name of the DLL where the function is delcared, without the `.dll` extension.

For example:

The following call specification publishes the method to SQL:

```
CREATE PROCEDURE bu1 (
    oc1 int,
    nc1 int,
    oc2 int,
    nc2 int)
  AS LANGUAGE CPLUSPLUS
  NAME 'CSPLib::bu1';
   /
```

11.8.4 C++ Stored Procedure Example

The following examples show how to create, build and publish the stored procedures.

- Section 11.8.4.1, "C++ Stored Procedure and Trigger Example One"
- Section 11.8.4.2, "C++ Stored Procedure and Trigger Example Two"
- Section 11.8.4.3, "JDBC Calling a C++ Stored Procedure Example"

11.8.4.1 C++ Stored Procedure and Trigger Example One

The following example does the following:

1. Creates the t1 table.
2. Creates the call specification of `bu1` for the C++ stored procedure `bu1` in the `CSPLib.dll`.
3. Creates a `BEFORE UPDATE` trigger, `foo`, which calls the `bu1` C++ stored procedure before the values of `c1` and `c2` in the table are updated.

```
create table t1(c1 int, c2 int);

create procedure bu1(oc1 int, nc1 int, oc2 int, nc2 int)
   as language cplusplus name 'CSPLib::bu1';

create trigger foo before update of c1,c2 on t1
  for each row bu1(old.c1,new.c1,old.c2,new.c2);
```

The following demonstrates how the trigger is executed, which in turn invokes the C++ stored procedure:

```
insert into t1 values(1,2);
insert into t1 values(10,2);

--trigger fired here
update t1 set c1 = 10, c2 = 20 where c1 = 1;
update t1 set c1 = 100 where c1 = 10;
```

11.8.4.2 C++ Stored Procedure and Trigger Example Two

The following example does the same as Example 1, but with a more complicated trigger. The trigger and procedure are dropped at the end of this example.

```
create table t3(c1 int, c2 int);

create procedure bc2(tabref varchar, tranid int, opseq int, c1 int, c2 int) as
language cplusplus name 'CSPLib::bc2';

--special trigger columns here
create trigger foo2 before insert on t3 for each row bc2(OL__TABLEREF, OL__
TRANSID, OL__OPSEQ, new.c1,new.c2);

--trigger fired here
insert into t3 values(1,2);
insert into t3 values(10,20);
insert into t3(c1) values(100);
insert into t3(c2) values(100);

drop trigger foo2;
drop procedure bc2;
```

11.8.4.3 JDBC Calling a C++ Stored Procedure Example

The following example shows JDBC invoking a C++ stored procedure through the `CallableStatement`.

```
//The following statement creates the procedure TESTINOUT1 with in out parameters
stmt.execute("CREATE OR REPLACE PROCEDURE TESTINOUT1(A IN OUT INT,
     B IN OUT DOUBLE PRECISION, C IN OUT VARCHAR, D IN OUT DATE,
     E IN OUT TIME, F IN OUT BINARY) AS LANGUAGE CPLUSPLUS
   NAME CSPLib::testInOut");

CallableStatement cstmt =
          conn.prepareCall("{call TESTINOUT1(?, ?, ?, ?, ?, ?)}");

cstmt.registerOutParameter(1, Types.INTEGER);
cstmt.registerOutParameter(2, Types.DOUBLE);
cstmt.registerOutParameter(3, Types.VARCHAR);
cstmt.registerOutParameter(4, Types.DATE);
cstmt.registerOutParameter(5, Types.TIME);
cstmt.registerOutParameter(6, Types.BINARY);

//setting parameters to null values
cstmt.setNull(1, Types.INTEGER);
cstmt.setNull(2, Types.DOUBLE);
cstmt.setNull(3, Types.VARCHAR);
cstmt.setNull(4, Types.DATE);
cstmt.setNull(5, Types.TIME);
cstmt.setNull(6, Types.BINARY);

for(int i = 0; i < 5; i++) {
 //executing the procedure. The parameters will be modified inside the procedure
 cstmt.execute();
 int a = cstmt.getInt(1);
 double b = cstmt.getDouble(2);
 String c = cstmt.getString(3);
 Date d = cstmt.getDate(4);
 Time e = cstmt.getTime(5);
 byte [] f = cstmt.getBytes(6);
}
```

11.9 Using .Net Stored Procedures

The .Net environment enables you to create stored procedures from any .Net language, such as C++, C#, C, and Visual Basic .Net. You create procedures and functions based on methods of a .Net class that is stored in an external DLL. Unlike C++ procedures, you don't need a fixed signature. Instead, the procedure can receive arguments and return values for any supported data type.

Note: Windows CE devices have the following limitations:

- You cannot pass delegates to native code when using the Compact Framework.
- You cannot start the .Net runtime from native code. The only way to use .Net on a Windows CE device is to start it within a C# application before the stored procedures are invoked.

The following sections detail how to build your .Net stored procedures:

- Section 11.9.1, "Creating the .Net Source for Your Stored Procedure"
- Section 11.9.2, "Building Your .Net Stored Procedures"
- Section 11.9.3, "Publish Your .Net Stored Procedure"
- Section 11.9.4, "Dropping .Net Stored Procedures"
- Section 11.9.5, ".Net Stored Procedure Example"

11.9.1 Creating the .Net Source for Your Stored Procedure

When you are creating a .Net stored procedure, you can use Oracle-specific .Net extension classes to access the database and transaction objects. The .Net extension classes discussed in the following sections are `OracleData`, `OracleDataRow`, and `OracleSPManager`.

The following sections demonstrate how to develop your .Net stored procedures:

- Section 11.9.1.1, "Defining Methods, Imports and Namespace"
- Section 11.9.1.2, "Access and Modify Database Using .Net Extension Classes In Stored Procedures"
- Section 11.9.1.3, "Access and Modify Database Using OracleSPManager Inside Triggers"

11.9.1.1 Defining Methods, Imports and Namespace

1. When you create your .Net source file, be sure to import the `Oracle.DataAccess.Lite` namespace.
2. All stored procedures are declared as public static methods of the class.

The following example defines public static methods and includes the `Oracle.DataAccess.Lite` namespace:

```
using System;
using Oracle.DataAccess.Lite;

public class SPClass
{
 //function which multiplies two integers
  public static int multiply(int a, int b)
  {
   return a * b;
  }

  //returns string length, as in C++ procedure example
  public static int strlen(string s)
    {
 return s.Length;
    }

//stores sum of first two arguments in the third argument
//as in C++ procedure example
    public static void trigSum(int a, int b, out int c)
    {
            c = a + b;
    }
public static void testInOut1(ref int dInt, ref double dDouble,
    ref string dStr, ref DateTime dDate, ref DateTime dTime,
    ref byte [] dBin)
```

```
        {
        dInt += 10;
            dDouble += 12.34;
            dStr += "aaaaa";

            dDate = dDate.AddYears(1);
            dDate = dDate.AddMonths(1);
            dDate = dDate.AddDays(1);

            dTime = dTime.AddMinutes(1);
            dTime = dTime.AddSeconds(1);

            int len = dBin == null ? 0 : dBin.Length;
            byte [] newBin = new byte[len + 5];
            if (dBin != null)
            Array.Copy(dBin, 0, newBin, 0, len);
            for(int i = len; i < newBin.Length; i++)
            newBin[i] = (byte)'x';
                dBin = newBin;
        }
    }
    }
```

11.9.1.2 Access and Modify Database Using .Net Extension Classes In Stored Procedures

The following are Oracle-specific .Net extension classes:

- Section 11.9.1.2.1, "OracleData"—A .Net version of the SODA `DBData` class.
- Section 11.9.1.2.2, "OracleDataRow"—encapsulates and Oracle Lite database row.

11.9.1.2.1 OracleData The `OracleData` object is a .Net version of the SODA `DBData` class.

Data types that are supported by Oracle Database Lite can be used in the `OracleData` object. These types can be implicitly cast to other compatible types; the cast must still follow normal database SQL casting rules. Casting between scalar types are implicit; casting to array types are explicit. If you try to cast an incompatible type, and `OracleException` is thrown.

This includes the following data types:

Table 11–5 Data Type For OracleData Object

Data Type			
int, int[]	byte, byte[] database type binary	short, short[]	bool, bool []
long, long[]	double, double[]	string, string[]	
DateTime, DateTime[]	OracleDataRow, OracleDataRow[]	OracleBlob, Oracle Blob[] (Oracle Lite Blob object)	

For example, the following code shows how you can cast a `String` to an `Integer` using the `OracleData` object:

```
string s = "10";
OracleData d = new OracleData(s);
int i = d;
```

11.9.1.2.2 OracleDataRow

The `OracleDataRow` object encapsulates and Oracle Lite database row. You can query and modify column values in place, instead of using SQL. The `OracleDataRow` implements indexes on the row, which returns an `OracleData` object.

To create an object query on a table, implement `OracleDataReader`, an Oracle extension of the ADO.Net `DataReader` object, to return `OracleDataRow` objects. The following example uses the `GetDataRow` method of the `OracleDataReader` object to retrieve the desired rows. To set up the query more efficiently, set the `RowQuery` attributes of `Table` and `Filter` before executing the query, as follows:

Note: For more information on the ADO.Net classes, see Chapter 13, "Oracle Database Lite ADO.NET Provider".

```
OracleConnection conn =
    new OracleConnection("DSN=POLITE;UID=SYSTEM;PWD=MANAGER");
conn.Open();
OracleCommand cmd = (OracleCommand)conn.CreateCommand();
//set properties for row query of table name and where clause
cmd.RowQuery.Table = "T1"; //table name
cmd.RowQuery.Filter = "C1 < 10"; //where clause

//execute the query and retrieve the desired rows
OracleDataReader rd = (OracleDataReader)cmd.ExecuteQuery();

//While there are rows, process each row
while(rd.Read())
{  //Retrieve each row with the GetDataRow method into the OracleDataRow object
    OracleDataRow row = rd.GetDataRow();
    //query and modify in place columns C1 and C2 of the row.
    //implicit conversion to and from OracleData
    // Retrieve the integer in column C1
    int i = row["C1"];
    // Add 5 to the value in C1 and store it in column C2
    row["C2"] = i + 5;
    //convert the value to a string and write it out
    string s = row["C2"];
    Console.WriteLine(s);
}
rd.Close();
conn.Close()
```

You can retrieve and modify the row by performing the following:

1. Retrieve the row with the `GetDataRow` method of the `OracleDataReader` class.
2. Query and modify the retrieved row within the `OracleDataRow` object.

11.9.1.3 Access and Modify Database Using OracleSPManager Inside Triggers

When you have a stored procedure that is executed by a trigger, the actual row that caused the trigger is accessible through the `OracleSPManager`. Thus, you do not have to create a SQL statement to retrieve the desired row. Instead, use the `GetDataRow` method of the `OracleSPManager` object. Also, you can use the `GetConnection` method of this object to retrieve the current `Connection` object.

The `OracleSPManager` class contains the following static methods, which you can ues to retrieve the connection or the row:

```
public static OracleConnection GetConnection();
public static OracleDataRow GetDataRow();
```

The following example uses the `OracleSPManager` static methods to retrieve the connection and row:

```
public static void log1(int a, int b, int c)
{
    //get current connection from .Net procedure manager
    OracleConnection conn = OracleSPManager.GetConnection();

    //get current row for trigger
    OracleDataRow r = OracleSPManager.GetDataRow();

    if (r[0] != a || r[1] != b || r[2] != c)
        throw new OracleException("Invalid row");

    if (ia == 0 && ib == 0 && ic == 0)
       throw new OracleException("Invalid row");

    OracleCommand cmd = (OracleCommand)conn.CreateCommand();
    cmd.CommandText = "INSERT INTO T1_LOG VALUES(?, ?)";
    cmd.Parameters.Add(new OracleParameter(a));
    cmd.Parameters.Add(new OracleParameter(a + b + c));
    cmd.ExecuteNonQuery();
    //do not close connection here
 }
}
```

11.9.2 Building Your .Net Stored Procedures

You can either build your stored procedure using Visual Studio .Net or by using the `olsp.mak` makefile. See the Visual Studio documentation for how to build using Visual Studio .Net.

If you want to build using the `olsp.mak` file, follow the directions as given for the C++ stored procedures in Section 11.8.2.2, "Automatically Build Your Stored Procedure". There are a few directions in that section that are specific to the .Net environment, as follows:

1. If you are building .Net procedures, then set the `NETFRKDIR` environment variable in the makefile to point to your .Net Framework directory
2. If building for the Compact Framework, then define the `CFK` macro and set the `CFSDKDIR` environment variable in the makefile to point to your Compact Framework SDK directory.
3. When building the `SPClass.dll` C# class, use the following syntax for the make:

   ```
   nmake -f olsp.mak SPClass.dll
   ```

4. Move the resulting DLL into the appropriate place, which is either the application directory or in the global assembly cache. Use the `gacutil.exe` executable if you want to install this DLL in the global assembly cache.

11.9.3 Publish Your .Net Stored Procedure

When you want to publish your .Net stored procedure, perform the following:

- Section 11.9.3.1, "Create the .Net Class Object in the Oracle Lite Database"
- Section 11.9.3.2, "Publish Methods With a Call Specification"

11.9.3.1 Create the .Net Class Object in the Oracle Lite Database

before you can create the call specification for the .Net stored procedure, you must first create the class within the Oracle Lite database. Use the following syntax:

```
CREATE [OR REPLACE] DOTNET CLASS USING BFILE('AssemblyName', 'ClassName');
```

Where:

- `AssemblyName` is the assembly file name, such as `SPClass.DLL`.
- `ClassName` is the name of the class, such as `SPClass`. However, if the class is defined within a namespace, prefix the namespace name before the classname, as follows: `MyNameSpace.SPClass`.

For example, the following creates the `SPClass` within the Oracle Lite database.

```
create dotnet class using bfile('SPClass.dll', 'SPClass');
```

11.9.3.2 Publish Methods With a Call Specification

Publish the methods in the class that you want to call from SQL with a call specification, which is created with either the `CREATE FUNCTION` or `CREATE PROCEDURE` commands.

Perform the following to publish your .Net stored procedures:

1. Publish any methods in the .Net class that you want to make accessible to SQL by creating call specifications for these methods. By publishing a method, you associate a SQL name to the method. SQL applications use this name to invoke the method.
2. Invoke the stored procedure through a SQL DML statement.

Publish any static method in the class that you want to call from SQL by creating a call specification for it. The call spec maps the method's name, parameter types, and return types to SQL counterparts.

To create a call spec, use the SQL commands `CREATE FUNCTION` for methods that return a value or `CREATE PROCEDURE` for methods that do not return a value. The `CREATE FUNCTION` and `CREATE PROCEDURE` statements have the following syntax:

```
CREATE [OR REPLACE]
   { PROCEDURE <proc_name> [([<sql_parms>])] |
   FUNCTION <func_name> [([<sql_parms>])]
   RETURN <datatype> }
   AS LANGUAGE DOTNET NAME
   '<class_name>.<method_name>)';
   /
```

Where:

- `<proc_name>` is a SQL procedure name; `<func_name>` is the name of the function in the DLL used for this procedure.
- `<sql_parms>` can be a maximum of 32 arguments. All arguments passed to the procedures are given as DBData values to the function, which must cast the arguments to the appropriate data type. The syntax has the following format:

  ```
  <arg_name> [IN | OUT | IN OUT] <datatype>
  ```

- `<datatype>` is the datatype.
- `<class_name>.<method_name>` is the name of the class and method that is used for the procedure or function.

For example:

The following call specification publishes the method to SQL:

```
CREATE PROCEDURE bu1 (
    oc1 int,
    nc1 int,
    oc2 int,
    nc2 int)
  AS LANGUAGE DOTNET
  NAME 'SPClass.bu1';
   /
```

11.9.4 Dropping .Net Stored Procedures

To drop a .Net class object from the database, delete it with the following drop statement:

```
drop dotnet class 'ClassName';
```

11.9.5 .Net Stored Procedure Example

The following examples show how to create, build and publish the stored procedures.

- Section 11.9.5.1, ".Net Stored Procedure and Trigger Example One"
- Section 11.9.5.2, ".Net Stored Procedure and Trigger Example Two"

11.9.5.1 .Net Stored Procedure and Trigger Example One

The following example does the following:

1. Creates the .Net `SPClass`.
2. Creates the `t1` table.
3. Creates the call specification of `bu1` for the .NET stored procedure `bu1` in the class.method: `SPClass.bu1`.
4. Creates a `BEFORE UPDATE` trigger, `foo`, which calls the `bu1` .Net stored procedure before the values of `c1` and `c2` in the table are updated.

```
create dotnet class using bfile('SPClass.dll', 'SPClass');

create table t1(c1 int, c2 int);

create procedure bu1(oc1 int, nc1 int, oc2 int, nc2 int)
   as language dotnet name 'SPClass.bu1';

create trigger foo before update of c1,c2 on t1
  for each row bu1(old.c1, new.c1, old.c2, new.c2);
```

The following demonstrates how the trigger is executed, which in turn invokes the .Net stored procedure:

```
insert into t1 values(1,2);
insert into t1 values(10,2);

--trigger fired here
update t1 set c1 = 10, c2 = 20 where c1 = 1;
update t1 set c1 = 100 where c1 = 10;
```

11.9.5.2 .Net Stored Procedure and Trigger Example Two

The following example does the same as Example 1, but with a more complicated trigger. The trigger and procedure are dropped at the end of this example.

```
create table t3(c1 int, c2 int);

create procedure bc2(tabref varchar, tranid int, opseq int, c1 int, c2 int) as
language dotnet name 'SPClass.bc2';

--special trigger columns here
create trigger foo2 before insert on t3 for each row bc2(OL__TABLEREF, OL__
TRANSID, OL__OPSEQ, new.c1, new.c2);

--trigger fired here
insert into t3 values(1,2);
insert into t3 values(10,20);
insert into t3(c1) values(100);
insert into t3(c2) values(100);

drop dotnet class ’SPClass’;
```

12 Using Simple Object Data Access (SODA) for PocketPC Platforms

SODA is an interface used for Oracle Database Lite C++ development that provides object-oriented data access using method calls, relational access using SQL and object-relational mapping to bridge the gap between the two. Object functionality is roughly three times faster than ODBC for simple operations. It enables rich datatypes—such as arrays and object pointers—as well as standard SQL columns. A programmer can store any data structure in the database and not think about relational design or performing joins.

A C++ developer can also use an interface for executing SQL statements. The resulting code is shorter and cleaner than ODBC. SQL queries can return objects to be examined and modified directly through the object-oriented layer, without calling any additional SQL statements.

Object-relational mapping enables the application to access relational data as if it was object hierarchy. Thus, your application can replicate rich data types or object pointers to the Oracle database server.

Oracle Database Lite includes SODA Forms, which is a library that simplifies the development of GUI applications for PocketPC devices. See Section 12.6.2, "Develop Your GUI Using the SODA Forms Library" for more details.

The SODA API method calls are documented in the *SODA: Simple Object Data Access API Reference*, which is located off the `<ORACLE_HOME>/Mobile/index.htm` page. The full sample code that is demonstrated in this chapter is located in the `<ORACLE_HOME>/Mobile/doc/soda/sodadoc/html/sodasimple_8cpp-source.html` file.

- Section 12.1, "Getting Started With SODA"
- Section 12.2, "Using SQL Queries in SODA Code for PocketPC Platforms"
- Section 12.3, "Virtual Columns and Object-Relational Mapping"
- Section 12.4, "Behavior of Reference-Counted and Copy-By-Assignment Objects"
- Section 12.5, "Another Library for Exceptions (ALE)"
- Section 12.6, "Building a SODA Forms Application"
- Section 12.7, "SODA Forms Edit Modes"
- Section 12.8, "Customizing Your SODA Forms Application"
- Section 12.9, "Displaying a List Of Objects in a Table"
- Section 12.10, "SODA Forms UI Controls"

12.1 Getting Started With SODA

In order to get started with SODA quickly, the following sections discuss the most frequently used classes:

- Section 12.1.1, "Overview of the SODA Classes"
- Section 12.1.2, "Demonstrating Frequently-Used SODA Classes"

12.1.1 Overview of the SODA Classes

When developing your C++ application, you would use the following classes the most:

- `DBSession` connects to the database and find and create classes.
- `DBClass` creates new database objects.
- `DBObject` modifies existing objects.
- `DBData` wraps an attribute value and is used for type conversion.
- `DBQueryExpr` builds single-table queries supported by SODA.
- `DBString (olString)` is a C string wrapper used by SODA.
- `DBList (olList)` is a template to store lists of values.
- `DBColList` and `DBDataList` instantiate these objects.

For example, implement the `DBObject` method, as follows:

```
DBObject obj;
...
obj["NAME"] = "Jack"
```

For full documentation for the SODA API method calls, see the SODA APIs, which you can find off the `<ORACLE_HOME>/Mobile/index.htm` page.

12.1.2 Demonstrating Frequently-Used SODA Classes

The following example demonstrates most of the SODA object-oriented functionality, as discussed in Section 12.1.1, "Overview of the SODA Classes".

```
void helloSODA() {
  puts("Hello SODA");
  try {
   DBSession sess("POLITE"); // Connect to the DSN, creating it if necessary

   // Find or create our class
   DBClass cls;
   try {
     cls = sess["PEOPLE"];
   } catch(DBException e) {
   cls = sess.createClass("PEOPLE", DBAttrList() <<
   DBAttr("ID", DB_INT) << DBAttr("NAME", DB_STRING));
  }
  // Create several objects. We can identify columns by name or positions
  DBObject o = cls.create("ID", 10, "NAME", "Alice");

  // The previous syntax of col1, val1, col2, val2 ... works for up to
// 32 columns. This version works for any number of columns
   cls.create(DBSetList() << "ID" << 10 << "NAME" << "Alice");
   cls.create(0, 20, 1, "Bob");
```

```
    // Note the automatic type conversion
    cls.create("ID", "314", 1, 3.14159265358);

    // Execute a query (will return two objects)
    DBCursor c = cls.createCursor(DBColumn("ID") == 10 ||
           DBColumn("NAME") == "Bob");

   DBObject ob;

   while (ob = ++c) {
    DBString s = ob["NAME"];
    puts(s);
    o["ID"] = (int)o["ID"]+1;
   }

   // Delete an object
   o.remove();

   // Clean up so that create class is successful next time
   sess.rollback();
  } catch(DBException e) {
  DBString s = e.getMessage();
  printf("Error: %s\n", (const char *)s);
  }
}
```

12.2 Using SQL Queries in SODA Code for PocketPC Platforms

To add SQL queries to your SODA code for PocketPC platforms, do the following:

1. Include `sodasql.h`, instead of `soda.h`.
2. Link your program with `sodasql.lib`.
3. Install `sodasql.dll` at runtime.
4. Create a `DBSqlSession` object instead of the `DBSession` object. Execute relational queries and other SQL statements with the help of `DBSqlStmt` and `DBSqlCursor` classes. Query results can be returned either as column values or as `DBObjects` for matching rows.

The following is a sample that uses the SODA relational interface:

```
void helloSQL() {
 try {
  puts("Hello SQL");
  DBSqlSession sess("POLITE");
  sess.execute("create table odtest(c1 int, c2 varchar(80))");
  DBSqlStmt stmt = sess.prepare("insert into odtest values(?,?)");

  // The values stand in for two ?'s in the statement above
  stmt.execute(5, "John");
  stmt.execute(10, "Mike");
  stmt.execute(15, "Alice");

  // Execute a single-table query that will return DBObject's for matching
  // rows. Use a standard SODA interface to access the objects
  DBSqlCursor c = sess.execute("odtest", "c1 < 15");
  while (++c) {
    DBObject o = c.getObject();
    DBString s = o["c2"];
    int val = o["c1"];
    printf("%d %s\n", val, (const char *)s);
    o["c1"] = val+1; // Can modify objects in addition to just reading them
  }

  // Execute a usual relational query
  c = sess.execute("select * from odtest");
  while (++c) {
    DBString s = c["c2"];
    printf("%d %s\n", (int)c["c1"], (const char *)s);
  }
 } catch(DBException e) {
 DBString s = e.getMessage();
 printf("Error: %s\n", (const char *)s);
 }
}
```

12.3 Virtual Columns and Object-Relational Mapping

A programmer does not view the data as column values that are stored in the database. For example, a master-detail relationship might be expressed as matching values in two tables, but for a program it is more natural to access a column in the master object, which contains an array of pointers to details.

The `DBVirtualCol` class enables the translation between the conceptual view of the data and the actual data in the tables. You can create a column that is completely under programmer's control through the `get`, `set` and `remove` methods and adding it to a class at runtime.

In fact, SODA contains a specialized `DBValueRel` class that extends the `DBVirtualCol` class to map master-detail relationships to object pointers. The following sample builds a binary search tree in the database using object-relational mapping:

```
struct HelloVirtual {
 int lpos, rpos, vpos;
 void visit(DBObject o);
 HelloVirtual();
};

void HelloVirtual :: visit(DBObject o) {
 while (o) {
 visit(o[lpos]);
 printf("%d\n", (int)o[vpos]);
 o = o[rpos];
 }
}

HelloVirtual :: HelloVirtual() {
 try {
   DBSession sess("POLITE");
   // TreeNode represent a binary tree with left and right pointers
   // that point back to parent's id column
   DBClass cls = sess.createClass("TreeNode",
   DBAttrList() << DBAttr("val", DB_INT) << DBAttr("id", DB_INT)
     << DBAttr("lchild", DB_INT) << DBAttr("rchild", DB_INT));
   // Create an index on id to speed up search
   cls.createIndex("i1", DBColList() << "id", true);
   // Set a sequence as a default value of id, so that we don't have to set it
   // explicitely
```

```
      DBSequence seq = sess.createSequence("s1");
      cls.defaultVal("id", seq);
      // Create the virtual columns
      DBValueRel lref("left", DBSrcCol(cls, "lchild") -> DBDstCol(cls, "id"),
           DB_UPD_DETAIL);
      DBValueRel rref("right", DBSrcCol(cls, "rchild") -> DBDstCol(cls, "id"),
           DB_UPD_DETAIL);
      // Cache column positions for frequent access
      lpos = cls["left"], rpos = cls["right"], vpos = cls["val"];
      // Root of the binary tree
      DBObject root;
      // Insert some random numbers into our binary search tree
      for (int i = 0; i < 50; i++) {
       int v = rand();
       DBObject o = cls.create(vpos, v); // Note automatically generated ids
       DBObject par; int dir;
       for(DBObject cur = root; cur; par = cur, cur = cur[dir])
       dir = (int)cur[vpos] >= v ? lpos : rpos;
       if (par) par[dir] = o; else root = o;
      }
      // Do in-order traversal of the tree, printing out numbers in sorted order
      visit(root);
    } catch(DBException e) {
    DBString s = e.getMessage();
    puts(s);
   }
 }
```

12.4 Behavior of Reference-Counted and Copy-By-Assignment Objects

Most C++ classes in SODA are reference-counted, which means that assigning one variable of one type to another cannot copy an object, but creates another way to refer to the same object. For example, the `DBData` class represents values that can be stored in persistent objects.

The following example demonstrates reference-counting:

```
DBData d = 5; // Create a new object containing value 5
                 // and make a reference to it
DBData d2 = d; // Both reference the same object
d << 20; // Add another value to existing object.
          // Both d and d2 reference the new array
d2 = 10; // d references the array, d2 is reassigned to the new data.
d.clear(); // Clear the last reference to the array of 5 and 20
            //and free the array
```

The programmer does not need to free objects when they are no longer used. However, this method is relatively expensive and not practical for objects that are created and destroyed often, such as when new lists of values, such as `DBSetList`, are allocated for each SODA call. Various lists in SODA, such as `DBSetList`, `DBDataList` and so on, are copy-on-assignment rather than reference-counted objects.

The following example demonstrates copy-by-assignment:

```
DBSetList ls << "cost" << 1000;
DBSetList ls2 = ls; // Created a copy of ls
ls << "value" << "priceless" // Only ls is changed
ls2.clear(); // Just set this copy to size 0
```

Note: SODA includes non-database classes and templates for your convenience, which have names that start with `ol` rather than `DB`—such as `olHash`. Avoid making unnecessary copies of these classes.

You can optimize your implementation by not creating an unnecessary copy by passing a reference or a `const` reference to the object, rather than an object, for both reference-counted and copy-on-assignment classes when calling a function. For example, `void func(const DBData &v)` avoids creating an unnecessary copy.

Note: The `clear` method only nullifies a particular reference to reference-counted objects. `DBSession` and `DBCursor` classes provide a `close` method that releases the underlying database resources, even while the objects are still referenced. Using anything that relies on a closed cursor or database connection throws a `DBException`.

12.5 Another Library for Exceptions (ALE)

Many embedded compilers, such as Visual C++ for PocketPC, do not support C++ exceptions. Oracle Database Lite includes ALE, which is a library that closely mimics C++ exceptions. The following sections describe how to use ALE:

- Section 12.5.1, "Decorating Classes With ALE"
- Section 12.5.2, "New Operator and ALE"
- Section 12.5.3, "Global Variables"
- Section 12.5.4, "Exceptions and Inheritance"
- Section 12.5.5, "Using ALE with PocketPC ARM Compilers"
- Section 12.5.6, "Troubleshooting ALE Runtime Errors"
- Section 12.5.7, "Compiling Your Program With ALE"
- Section 12.5.8, "ALE Code on Systems That Support Exceptions"

12.5.1 Decorating Classes With ALE

A decorated class is one that uses ALE for handling its exceptions. If your embedded compiler does not support exception handling, then use ALE, which relies on careful accounting of all objects that are already constructed or are being constructed. ALE supports stack unwinding and catching exceptions based on object type.

To use ALE, C++ source code needs to be modified where the try, catch and throw blocks are replaced with the ALE macros, which is known as decorating classes. The following is an example of a decorated class:

```
#include "ale.h"
struct Error {
    const char *msg;
    Error(const char *msg) :msg(msg) {}
};

aleTry {
    olArray<char> a(5);
    aleThrow(Error,Error("Adios"));// Or aleThrowObj(Error,("Adios"));
} aleCatch(Error,e) {
    puts(e.msg);
} aleCatch(aleBadAlloc,e) {
    puts("Tough!");
} aleCatchAll {
    puts("Some other exception happened\n");
    aleReThrow;
} aleEnd;
```

Table 12–1 ALE Macros for C++ Exceptions

Macro	Action
`aleTry`	Equivalent to C++ try. Use to enclose the code that might encounter any exception.
`aleCatch (type, varName)`	Catch exception of a given type and store it in the variable `varName`, which is local to the block. Unlike the regular C++ exceptions, the `type` name string must match the argument of the throw exactly.
`aleThrow (type, obj)`	Throw an exception contained in `obj` of `type`. ALE supports single inheritence of exceptions, as described in Section 12.5.4, "Exceptions and Inheritance".
`aleThrowObj (type, arg1, arg2, ....)`	Construct a new object of `type` with the specified arguments and throw it as an exception.
`aleCatchAll`	Catch any exceptions that are not handled explicitly.
`aleReThrow`	Rethrow the exception that is caught in the innermost `aleCatch` or `aleCatchAll` block.
`aleEnd`	Close the exception handling construct. Add a semi-colon `;` after `aleEnd`.

If your embedded compiler does not support exception handling, then use ALE, which relies on careful accounting of all objects that are already constructed or are being constructed. Your class is involved in exception handling if the class is on the stack when an exception is thrown, its constructor may throw an exception, or it has a decorated superclass and member—even if the class does not do any additional exception-related processing.

When you decorate one class with ALE, you must decorate all classes involved with this class. If you omit a decoration in one of the classes, then the program might fail. Therefore, it is best to decorate all your classes except for plain C-style structures that do not have any constructors or destructors.

If your class does not have any constructors with a body that throws exceptions (it is OK if the superclass or member constructor does), then you can use a simple form of class decoration by adding `ALELAST(ClassName)`, without a semicolumn, at the end of class declaration. `ALELAST` informs the library to register the class and clean-up if an error occurs. The following demonstrates how to use `ALELAST`:

```
template<class T> class PtrHolder {
   T *ptr;
public:
   ~PtrHolder() { delete ptr; }
   operator T *() { return ptr; }
   ALELAST(PtrHolder)
};
```

If any of the constructors throw exceptions, then you need to do the following:

1. Add `ALECLAST(ClassName)`, rather than `ALELAST` to the end of the class body.
2. Add `ALECONS(ClassName)` in the beginning of the body of each constructor.

This is demonstrated, as follows:

```
template<class T> class Array {
    T *a;
    size_t len;
public:
    Array(size_t len=0} : len(len) {
        ALECONS(Array);
        a = new T[len];
    }
    Array(const Array &arr) : len(arr.len) {
        ALECONS(Array);
        for (size_t i = 0; i < len; i++)
            a[i] = arr.a[i];
    }
    ...
    ALECLAST(Array)
};
```

In this example, the class contains an explicit copy constructor. If your class does not contain an explicit copy constructor and instances can be copied, then you need to explicitly write a copy constructor and add `ALECONS(ClassName);` rather than using a copy constructor that is generated by the compiler. If you need a more complicated initialization than a default or copy constructor, then use a global pointer—which can be initialized by another global object, rather than a global instance.

12.5.2 New Operator and ALE

Decorated classes can be safely used with the `new` and `delete` functions, including using `new` and `delete` for `array` and `placement new`. However, systems that do not support exceptions usually do not declare `std::bad_alloc` and `std::no_throw` types. Use `aleBadAlloc` and `aleNoThrow` instead of using `std::bad_alloc` and `std::no_throw` types.

One design decision is whether to decorate classes with constructors that only throw `bad::alloc` if they run out of memory using `ALELAST` or `ALECLAST`. If you are writing classes for a single application that does not allocate much memory, you might dispense with error checking and just use `ALELAST`. Your program might crash because of incorrect cleanup calls if it runs out of memory. If your class allocates a lot of memory or you are writing a highly-reusable framework, then it is best to use `ALECLAST` and decorate all the constructors.

12.5.3 Global Variables

If you need a global or static variable to be decorated with ALE, declare it using `aleGlobal` template, as follows:

```
aleGlobal<MyType> myGlobal; // Initialized with default constructor
aleGlobal<MyType> myGlobal1(MyType("Hello", 5")); // Initialized with copy
constructor.
...
MyType *t = myGlobal; // Declared variables behave as pointers
```

Declare all global or static decorated instances using `aleGlobal` or you may receive runtime errors.

12.5.4 Exceptions and Inheritance

Unlike regular C++ exception handling, ALE requires that class names in `aleThrow` and `aleCatch` match exactly. Typedef names, throwing a subclass, and catching a superclass will not work. To build a hierarchy of exceptions, add `ALEPARENT` declaration to the subclass, as follows:

```
class BaseE {
    ALELAST(BaseE)
};
class DerivedE : public BaseE {
    ALELAST(DerivedE)
    ALEPARENT(BaseE)
};
```

`DerivedE` can be caught as `BaseE`. If you use multiple inheritance, then the first base class must be declared as a parent.

12.5.5 Using ALE with PocketPC ARM Compilers

The Microsoft Embedded Visual C++ for ARM has a bug that is triggered when an ALE-decorated object (or any object with embedded pointers) is passed by value to a function or method. The affected code receives an ALE fatal error message at runtime. To avoid this problem, always pass SODA objects and instances of other classes that use ALE as a constant reference rather than value. For example, modify the following code:

```
void createName(DBClass cls, DBString name) {
cls.create("name", name);
}
```

to the following implementation:

```
void createName(const DBClass &cls, const DBString &name) {
cls.create("name", name);
```

12.5.6 Troubleshooting ALE Runtime Errors

If your classes are not decorated properly, then you will receive runtime errors. On PocketPC, ALE displays a message box explaining the problem, and then the program terminates. In addition, the error and the dump of the ALE stack is appended to `aleDump.txt`, which exists in the root directory of the device. In simple cases, the error message pinpoints the exact problem; for example `ALECONS is missing for class MyArray`. Usually, one of the classes found near the top of the ALE stack is not decorated properly. If you do not decorate a class and its superclasses or members are decorated, then you may receive a runtime error and see the superclasses/members on the stack.

12.5.7 Compiling Your Program With ALE

To build a program that uses ALE, include `ale.h` from the Oracle Database Lite SDK and link with the `olStdDll.lib` library. You need `olStdDll.dll` at runtime.

12.5.8 ALE Code on Systems That Support Exceptions

For systems that already support C++ exceptions, like Win32, Oracle Lite includes a dummy `ale.h` that defines the same macros, but uses regular C++ exceptions to implement them. If you are writing code that must execute on both Win32 and

PocketPC, remember to test the code with the actual ALE library to ensure that all your classes are decorated correctly. `ALELAST`, `ALECLAST` or `ALECONS` have no effect on the Win32 platform.

12.6 Building a SODA Forms Application

The following sections describe how to create a SODA Forms Application for PocketPC platforms:

- Section 12.6.1, "Development Environment Requirements"
- Section 12.6.2, "Develop Your GUI Using the SODA Forms Library"
- Section 12.6.3, "Designing the UI for PocketPC"
- Section 12.6.4, "Customizing the Database Schema"
- Section 12.6.5, "Binding UI to Data in the PocketPC Environment"
- Section 12.6.6, "Setting List Choices for Status Contol on PocketPC"
- Section 12.6.7, "Customizing the Table in OrderForm"
- Section 12.6.8, "Monitoring the Logic"

12.6.1 Development Environment Requirements

SODA Forms relies on SODA, which is an easy-to-use C++ interface for the Oracle Lite database engine. Read SODA documentation before continuing with this manual. Make sure you understand objects, queries, cursors and virtual columns as a way to customize database schema for a particular application.

When developing for the PocketPC environment, refer to the Microsoft Development Network. Focus on how to use Embedded Visual C++ to create resources and compile programs for PocketPC. Also, familiarize yourself with Windows UI controls, including their formats and styles.

If you want to use SQL in your application, understand the SODA SQL support. The `FormsOrder` demo uses SODA SQL to support custom queries.

Find and open `FormOrders.vcp` in `SodamFormCE\FormOrders` directory under the samples for SODA on the PocketPC. This is the demo for PPC2003.

12.6.2 Develop Your GUI Using the SODA Forms Library

SODA Forms is a quick way to create data entry GUI.

12.6.2.1 Traditional Way to Develop Native Data Entry Applications

The GUI for data entry applications performs the same actions, as follows:

- Copy values from different columns of a database record into controls, such as text fields and checkboxes.
- Track what the user is doing by handling various UI events, like "field entered" or "button clicked". The user typically examines or edits the data on the screen and presses a button or picks a menu item to designate the next action.

To perform the action requested by the user, the application performs database and UI calls. For example, to save changes, the application retrieves the values in the UI controls and eventually executes a SQL update statement. An application needs to be able to discard changes, delete records, create new records, go to the previous or the next entry, and search for data that satisfies user-selected conditions.

Normally, you duplicate the UI code for each screen supported by the program, which can result in a bulky program. Instead, use Soda Forms, as described in Section 12.6.2.2, "Trimming Your PocketPC UI Code With SODA Forms", to streamline your UI code.

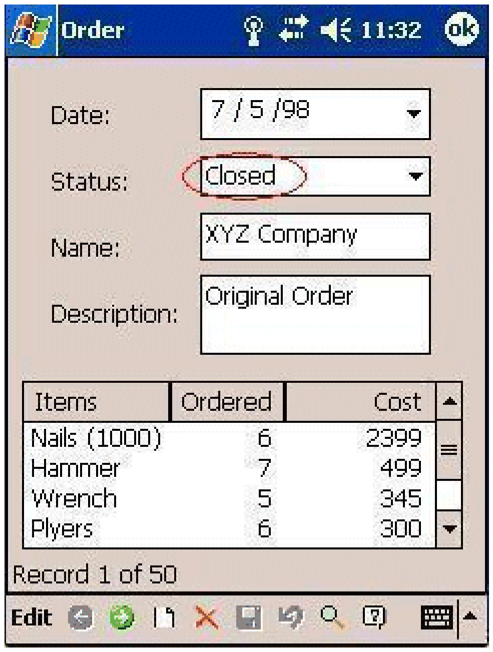
12.6.2.2 Trimming Your PocketPC UI Code With SODA Forms

We moved the boilerplate UI code to a library and to enable the programmer to concentrate on application logic, not the tasks of copying values from database rows to UI controls. Oracle Database Lite provides this library for PocketPC.

The following example creates a form:

```
DBSession sess("OrdersODB"); // Open a database connection
DBClass cls = sess["ORD_MASTER"]; // Locate a table in the database
DBForm frm(OrderForm, cls, orderCols, OL_COUNTOF(orderCols)); // Initialize a form
frm.edit(cls.createCursor()); // Let the user edit all records in ORD_MASTER table
```

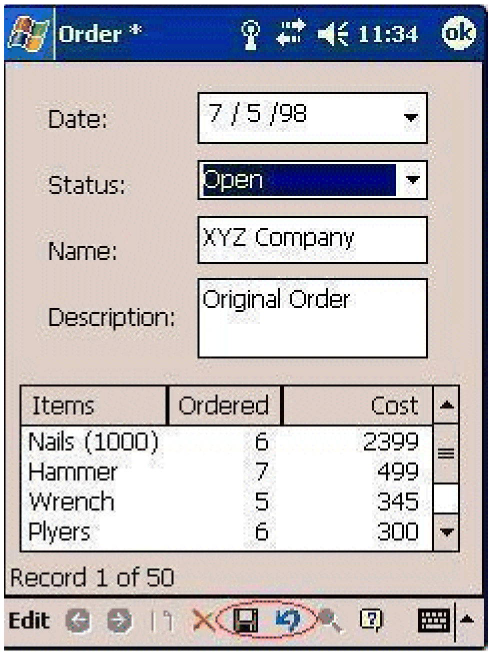
When designing an application for PocketPC, the form is created with Embedded Visual C++ resource editor with values from the first row of `ORD_MASTER` table1, as follows:



Date:	7 / 5 /98
Status:	Closed
Name:	XYZ Company
Description:	Original Order

Items	Ordered	Cost
Nails (1000)	6	2399
Hammer	7	499
Wrench	5	345
Plyers	6	300

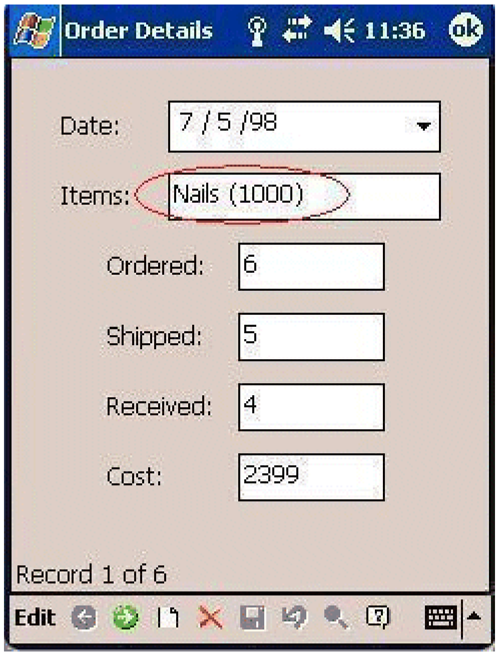
Record 1 of 50



Date:	7 / 5 /98
Status:	Open
Name:	XYZ Company
Description:	Original Order

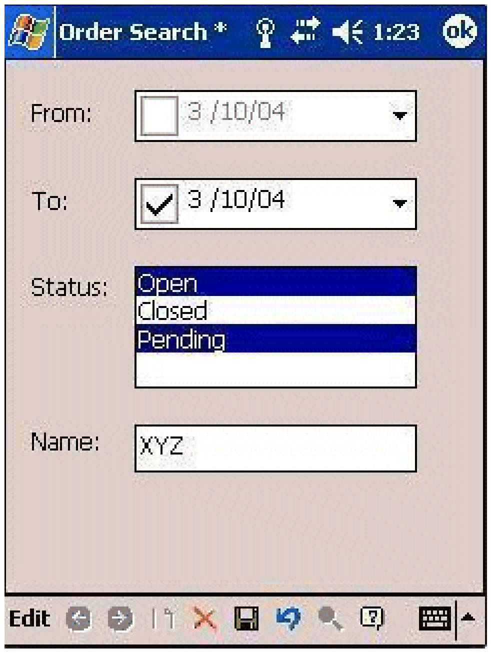
Items	Ordered	Cost
Nails (1000)	6	2399
Hammer	7	499
Wrench	5	345
Plyers	6	300

Record 1 of 50



Date:	7 / 5 /98
Items:	Nails (1000)
Ordered:	6
Shipped:	5
Received:	4
Cost:	2399

Record 1 of 6



From: ☐ 3 /10/04

To: ✓ 3 /10/04

Status: Open, Closed, Pending

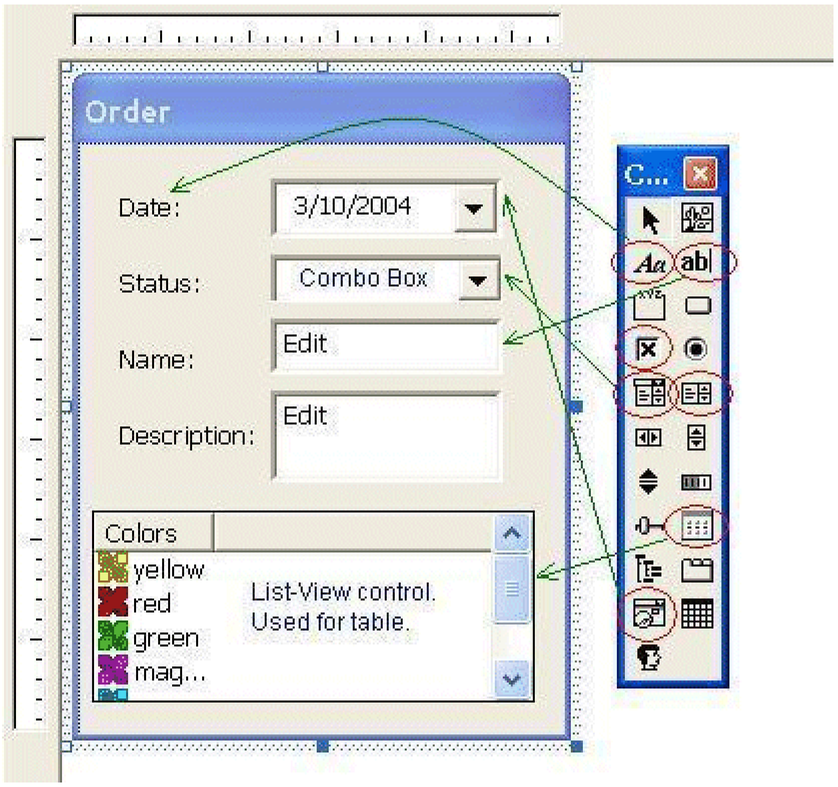
Name: XYZ

The user can update the current record and then either save the changes or discard them and reload from the database. He or she can also search data using any values visible on the screen, scroll through all the records as well as delete existing entries or create new ones. No special code is needed to support these operations.

On PocketPC, you can also see a lens button on the toolbar. When this button is clicked, the application can launch a cusomtized dialog for entering search conditions, retrieve their values, an dexecute custom queries. For PocketPC, we only support queries implemented by the application.

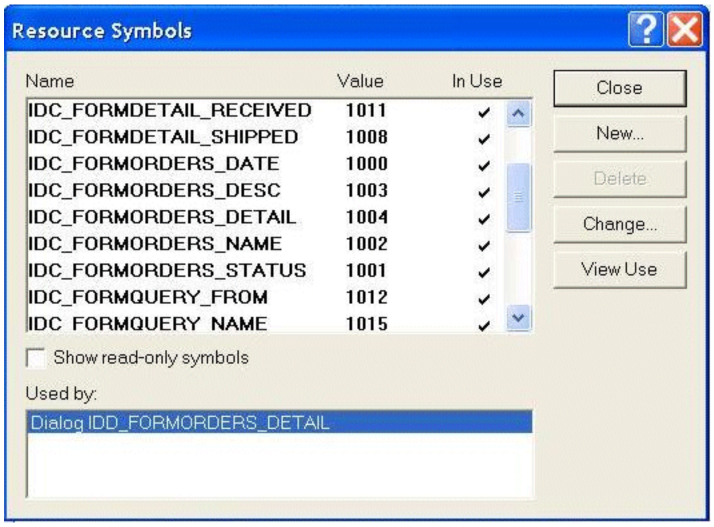
12.6.3 Designing the UI for PocketPC

For the PocketPC platform, a SODA form is a Windows dialog. Thus, the first step is to design dialogs for every form using the Embedded Visual C++ resource editor. In the `FormOrders` project, click on the resource tab, open the dialog folder and click on the `IDD_FORMORDERS_MASTER` dialog. This shows the dialog for the master form, as follows:



The controls on the master form, labels (also called static controls, such as "Date:", "Status:", and so on), edit controls (name and description fields), Date-Time picker (for date field), ComboBox (for status field), and list view control for the detail table. There are corresponding icons on the toolbox used to create these controls.

Each control has separate formats and styles that can be customized through the resource editor to find the appearance and behavior you want. For every control there is a resource symbol with numeric id assigned to it. Right-click on the `FormOrders.rc` and select `Resource Symbols`, as follows:



Name	Value	In Use
IDC_FORMDETAIL_RECEIVED	1011	✓
IDC_FORMDETAIL_SHIPPED	1008	✓
IDC_FORMORDERS_DATE	1000	✓
IDC_FORMORDERS_DESC	1003	✓
IDC_FORMORDERS_DETAIL	1004	✓
IDC_FORMORDERS_NAME	1002	✓
IDC_FORMORDERS_STATUS	1001	✓
IDC_FORMQUERY_FROM	1012	✓
IDC_FORMQUERY_NAME	1015	✓

☐ Show read-only symbols

Used by:

Dialog IDD_FORMORDERS_DETAIL

You can either create your own identifiers or let the resource editor create them for each symbol. Look at the header file, `resource.h`, which is generated by the resource editor, to see how identifiers are assigned to resource symbols using `#define` statements.

12.6.4 Customizing the Database Schema

The following SODA code connects the application to an Oracle Lite database and retrieves the `DBClass` objects for two tables used in the demo—`ORD_MASTER` and `ORD_DETAIL`:

```
DBSession sess("OrdersODB"); // Open a database connection
DBClass mCls = sess["ORD_MASTER"];
DBClass dCls = sess["ORD_DETAIL"];
```

However, values stored in the database do not exactly match what should be shown to the user. SODA allows each application to customize how the schema is shown. Both the `ORD_MASTER` and `ORD_DETAIL` tables have ID columns, which are primary keys and should be initialized to some unique value when a new record is created. To avoid asking the user to pick unique values, set the default value of each column to a sequence, as follows:

```
mCls.defaultVal("ID", sess.findSequence("OrderSeq"));
dCls.defaultVal("ID", sess.findSequence("DetailSeq"));
```

In addition, the UI shows the order status as "Open", "Closed" or "Pending"; however, the "STATUS" column in `ORD_MASTER` stores numbers, such as 0, 1 or 2. The following line creates a virtual column named `strStatus` that contains a mapped value:

```
DBMapCol mc("strStatus", mCls, "STATUS", DBDataList() << 0 << "Open" << 1 <<
"Closed" << 2 << "Pending" << DBNULL << DBNULL);
```

Finally, modify the Order screen to contain a table of items included in the order. SODA Forms populates a table by reading a column that stores a list of `DBObject` pointers and copying values from each object to a table row. In our case, `ORD_DETAIL` rows relate to a row in `ORD_MASTER` through the values of the `KEY` column, which matches the value of the master ID. The following declaration creates a pseudo-column that contains an array of pointers and updates `KEY` values as assigned:

```
DBValueRel rDet("detail", DBSrcCol(mCls, "ID") -> DBDstCol(dCls, "KEY"),  DB_UPD_
CASCADE);
```

If a mapping you are looking for is not part of SODA, then you can create your own by subclassing the `DBVirtualCol` class. Default values and virtual columns are transient and not stored in the database. Thus, declare them in every application that requires access to your abstractions.

12.6.5 Binding UI to Data in the PocketPC Environment

Once you complete customizing the database schema, you can map columns in the database to values that appear on the user screen when editing a particular record. You can map the columns by binding UI to data in the PocketPC environment. For each column, you need to define a `DBFormCols` structure that has the following fields.

- The resource id assigned to a particular UI control.
- The column name to which the UI control will be bound.
- The optional argument that specifies how the control should be edited. For PocketPC, the only relevant value is `DBFormEditListIndex`, which asks to use the index of the selection, rather than its string value, for `ListBox` and `ComboBox` controls. If omitted, the default method is used.

Here is the mapping for our two forms:

```
//Mapping for ORD_MASTER table
static const DBFormCols orderCols[] = {
 { IDC_FORMORDERS_DATE, "DDATE", DBFormEditDate},
 { IDC_FORMORDERS_NAME, "NAME"},
 { IDC_FORMORDERS_DESC, "DESCRIPTION"},
 { IDC_FORMORDERS_STATUS, "strStatus"},
 { IDC_FORMORDERS_DETAIL, "detail"}
};
//Mapping for ORD_DETAIL table
static const DBFormCols detailCols[] = {
    { IDC_FORMDETAIL_DATE, "DDATE", DBFormEditDate },
    { IDC_FORMDETAIL_ITEMS, "DESCRIPTION" },
    { IDC_FORMDETAIL_ORDERED, "QTYORDERED", DBFormEditDigits },
    { IDC_FORMDETAIL_SHIPPED, "QTYSHIPPED", DBFormEditDigits },
    { IDC_FORMDETAIL_RECEIVED, "QTYRECEIVED", DBFormEditDigits },
    { IDC_FORMDETAIL_COST, "COST", DBFormEditDigits }
};
```

Once the mapping is in place, create the `DBForm` objects by specifying the resource ID of the form itself, the `DBClass` of objects that the form will be used to edit and the mapping table with the number of elements it contains, as follows:

```
DBForm mFrm(OrderForm, mCls, orderCols, OL_COUNTOF(orderCols));
DBForm dFrm(DetailForm, dCls, detailCols, OL_COUNTOF(detailCols));
```

12.6.6 Setting List Choices for Status Contol on PocketPC

For listbox and ComboBox controls in the PocketPC environment, set the string list of choices. You cannot set the list choices for the listbox in the resource editor, and it can be difficult to set for the combobox. Thus, set the list choices for both the listbox and the combobox programmatically through the `setListItems` function of `DBFormCol`, as follows:

```
DBList<DBString> stList;
stList << "Open" << "Closed" << "Pending";
mFrm[IDC_FORMORDERS_STATUS].setListItems(stList);
```

12.6.7 Customizing the Table in OrderForm

Specify which columns of the `ORD_DETAIL` table appear in the table and how the values are aligned within a cell and in the title of each column. The following array contains the information:

```
static const DBFormTblCols detailTblCols[] = {
    { "DESCRIPTION", "Items" },
    { "QTYORDERED", "Ordered", DBFormColRight },
    { "COST", "Cost", DBFormColRight }
};
```

Retrieve a handle to the table column and set its format, as follows:

```
DBFormCol tCol = mFrm[OrderDetailTable];
tCol.setTableInfo(dCls, detailTblCols, OL_COUNTOF(detailTblCols));
```

12.6.8 Monitoring the Logic

The following few lines of code constitute the logic of `FormOrders` application:

```
// Load all objects of ORD_MASTER into a form
```

```
DBCursor c = mCls.createCursor();
mFrm.load(c);
// Handle table select event in master form to launch the detail form
while (mFrm.edit() == OrderDetailTable)
        dFrm.edit(mFrm[OrderDetailTable]);
```

From this, the following occurs:

1. A list of objects from `ORD_MASTER` to edit is loaded into the order form.
2. The `mFrm.edit` function displays the UI and handles many of the user actions internally. Before the call returns, the user could have made changes, created new records, searched through the data, and so on.
3. The return values from the function call are an identifier of the UI control that was activated, which was not handled internally. In the case of the Orders demo, it would either be the `OrderDetailTable`—meaning that a row in that table was clicked—or the `DBFormItemExit`—if the exit icon on the toolbar was clicked.
4. If the user clicked the table, the demo asks the `DBForm` detail to edit the list of objects contained in the table.
5. Once the edit is completed, any pointers to new objects are saved in the `detail` virtual column of the `ORD_MASTER` row. The `DBMapCol` updates the `KEY` column of the objects to the identifier value that matches the master.

You can use SODA Forms to write concise UI code without replicating a boilerplate. The next sections explore more advanced capabilities of the library.

12.6.9 Compiling Your SODA Application

SODA can be compiled on Windows and PocketPC environments. To build a SODA application on Win32, perform the following:

1. Add the Oracle Database Lite SDK to the include and library path in your compiler options.
2. Include the appropriate `.h` files and link with appropriate library files, as follows:
   - For most applications, include `<soda.h>` and link with `olStdDll.lib` and `sodadll.lib`.
   - If you are using Visual Studio.Net and you are building an application using SODA SQL binding, include the `<sodasql.h>` file instead of `soda.h` and link with `olStdDll.lib`, `sodadll.lib`, and `sodasql.lib`.
   - If you are using Visual C++ 6.0 and building and application using SODA SQL binding, then link with `olStdDll6.lib`, `sodadll6.lib`, and `sodasql6.lib` libraries. The `olStdDll` library contains utility classes that are not related to database, such as `olString` and `olHash`. It does not depend on the rest of Oracle Database Lite runtime, except `ceansi.dll`, on PocketPC platforms.
   - To build a SODA Forms application for PocketPC platform, include `SodaForm.h` into your program and link with the following import libraries: `sodadll.lib` (soda library), `sodasql.lib` (library for soda sql) and `sodaform.lib` (sodaform runtime). Also, install the Oracle Database Lite runtime and `sodadll.cab` in order to run your application. `Sodadll.cab` contains `sodadll.dll`, `sodasql.dll`, `sodaform.dll` and `SodaFormHelp.html` (default help file for SODA Forms). Currently, we support SODA Forms for two Pocket PC platforms: Pocket PC 2002 and

Pocket PC 2003. The soda libraries and `sodadll.cab` files are provided in the SDK for each platform—both for the device and the emulator.

SODA includes a software emulation library that requires some changes in syntax when using C++ exceptions, but keeps the program structure intact. See Section 12.5, "Another Library for Exceptions (ALE)" on how to support C++ exceptions.

12.7 SODA Forms Edit Modes

The following sections describe the SODA Forms edit modes:

- Section 12.7.1, "Editing a Single Object"
- Section 12.7.2, "Editing a List of Objects"
- Section 12.7.3, "Creating a New Object"
- Section 12.7.4, "Popping Up A Dialog"
- Section 12.7.5, "Custom Queries for PocketPC Environment"

12.7.1 Editing a Single Object

The code below enables a user to edit a specific object. To edit a list of objects, see Section 12.7.2, "Editing a List of Objects":

```
DBObject o;
DBFormItem id = frm.edit(o);
```

In this case, toolbar will not have arrows to scroll or "new" icon to create a new object. Query is disabled. If the user saves the changes or deletes the object, `DBFormItemSave` and `DBFormItemDelete` are returned respectively. With list edit, save or delete do not return, since the user can still make additional changes to another record.

12.7.2 Editing a List of Objects

The following example loads a list of objects into a form, and enables the user to insert, delete, update or query:

```
olList<DBObject> ls;
ls << obj1 << obj2 << obj3;
DBFormItem id = frm.edit(ls);
```

When the edit returns, the `DBObject` list is updated with the new list of objects that reflects creation and deletion. The `DBFormItem` variable has one of the following values:

Table 12–2 DBFormItem Identifier Values

Value	Explanation
DBFormItemExit	On PocketPC, the user clicked an OK button on the navigation bar
Identifier of a table column	User clicked on a row in a table. Use the frm[id].getSelectedRowfunction to determine which one it is.
Identifier of a button	User clicked on a button in the form.
Identifier of a menu item	User clicked on a menu item not handled internally by SODA forms.

Table 12–2 (Cont.) DBFormItem Identifier Values

Value	Explanation
Identifier of a pop-up list or checkbox	The UI control was changed and you called frm[resId].setChangeNotify(true) on that column.
DBFormItemRevert	You set DBFormRevertExit edit flag and the user reverted a change.

There are several ways to customize the editing. The following example demonstrates a customization:

```
DBCursor cur = cls.createCursor(DBColumn("status") == "Active");
frm.load(cur, DBFormNew|DBFormUpdate|DBFormDirtyExit, DBSetList() << "zip" <<
94403 << "status" << "Active");

DBFormItem id;
while ((id = frm.edit()) != DBFormItemExit)
    if (id == CustomerBelmontButton) {
        frm[CustomerZipField] = 94002;
        frm.dirty();
    }
olList<DBObject> ls;
frm.getList(ls);
```

The `DBFormNew` or `DBFormUpdate` edit modes are specified instead of the default `DBFormListEdit` mode, which disallows the deletion of records while creating and updates are OK. `DBFormReadOnly` enables users to view and search the data.

The `DBFormDirtyExit` flag means that a button press or menu selection would exit edit even if the form is dirty. The default is to beep and wait until the user saves or reverts the change.

Specifying a `DBSetList` provides initial values that are given when a new object is created. For bound, enabled columns, the user can change that value. For unbound database columns or columns bound to read-only UI controls, this is the final value.

You can specify objects to edit in two ways – by giving an explicit list of objects or by providing a `DBCursor`. In the later case, the results of a query are loaded into the form.

In this case, load the list of objects once, call edit one or more times, and then retrieve the final edited list. SODA Forms provides you a choice between calling the edit method with all the arguments or calling load, edit without arguments and then `getList` or `getObject` functions. Examples in this document only show one possibility and do not discuss each edit flag.

12.7.3 Creating a New Object

The example below loads initial values into a form and asks the user to modify the values. The user then either clicks **Save** to create a new record or clicks **Delete** to cancel the creation.

```
DBObject o = frm.create(DBSetList() << "zip" << 94403 << "status" << "Active");
```

This call returns the new object or `DBNULL`, if the creation was canceled. However, if you use the load/edit sequence, then call the `getObject` method to retrieve the object handle, if `DBFormItemSave` is returned by the `edit` method.

12.7.4 Popping Up A Dialog

Although SODA Forms is designed for editing database records, you can use the same interface to retrieve input from the user. The following example enables the user to edit two fields and then retrieve the result:

```
DBForm frm(CustomerForm);
frm[CustomerZipField] = 94002;
frm[CustomerStatusField] = "Active";
DBFormItem id = frm.dialog();
if (id == DBFormItemSave) {
       DBString status = frm[CustomerStatusField];
       int zip = frm[CustomerZipField];
       ...
}
```

The values set before the `frm.dialog` call are default values. The user can make changes and then click **Revert** to restore the defaults and try again. Finally, the user clicks **Save** to send the changes to the program.

12.7.5 Custom Queries for PocketPC Environment

If you want to search on the list of objects loaded into a form, you can perform a custom query, which is the query logic implemented by the application. Execute the `FormOrders` demo and click on the lens toolbar button on the main form. It pops up a form—a dialog—to enter the search parameters for a custom query. Examine the `IDD_FORMORDERS_QUERY` dialog under the `FormOrders` resources to see how it enters the following search criteria: from- and to- dates, multiple list choices for the order status, and keyword search on the company name.

The following code sets up the query form:

```
//orders query form
//this constructor will automatically put the form into a dialog mode
DBForm qFrm(IDD_FORMORDERS_QUERY);
//set list choices
qFrm[IDC_FORMQUERY_STATUS].setListItems(stList); //same list as for the master
form
qFrm[IDC_FORMQUERY_STATUS].setEditType(DBFormEditListIndex); //cannot use virtual
column here
```

In order to enable the lens toolbar button, specify `DBFormCustomQuery` mode when loading the master form, as follows:

```
// Load all objects of ORD_MASTER into a form
DBCursor c = mCls.createCursor();
long mMode = DBFormListEdit | DBFormCustomQuery;
mFrm.load(c, mMode);
```

The full logic of the `FormOrders` application is as follows:

```
for(;;) {
    DBFormItem i = mFrm.edit();
    if (i == IDC_FORMORDERS_DETAIL)
        dFrm.edit(mFrm[i]);
    else if (i == DBFormItemQuery) {
        DBFormItem j = qFrm.edit();
        if (j == DBFormItemSave) {
           mFrm.load(doQuery(sess, qFrm), mMode);
           mFrm.setTitle("Search Results");
        }
```

```
            else if (j == DBFormItemDelete) {
               mFrm.load(mCls.createCursor(), mMode);
               mFrm.setTitle("Order");
            }
        }
        else //DBFormItemExit
            break;
}
```

Call the `edit` method on the master form and look at the return value. If the row in the table was clicked (`i == IDC_FORMORDERS_DETAIL`), then call the `edit` method on the detail form and then return to the master form through the loop as shown earlier in the document. If the user clicks on the query(lens) button (`i == DBFormItemQuery`), then pop up a query dialog (`qFrm.edit`). Once the user sets up the query parameters, the user can click on the **Save** button to execute the query (`j == DBFormItemSave`), **Delete** button to cancel the query (**j == DBFormItemDelete**) or exit the form to come back to the previous screen. The `doQuery` function creates the `DBSqlCursor` based on the search parameters in the query form. To execute the query, load the master form using the `DBSqlCursor` and change the form title to `Search Results`. If the query was canceled, reload the master form with the original list of objects (`mCls.createCursor`) and change its title back to the original.

The `doQuery` function retrieves the search parameters from the query form as `DBData` from its columns. Then it creates a SQL where-clause string and the binding list for it. The multiple choices for the status list are stored inside `DBData` as an array of integers. For the company name field, the search is performed using a `LIKE` expression—`LIKE %s%`—so that any substring matches.

12.8 Customizing Your SODA Forms Application

You can customize the UI of your application by configuring your resource file. Also, edit the `sodares.rsrc` file to modify the UI resources used by the SODA Forms library.

12.8.1 Customizing Help Messages

On PocketPC, when the user clicks on the help button on the toolbar, SODA Forms launches the Windows CE help editor and loads the help file. The default help file is `SodaFormHelp.html`, which describes editing the database record. This help file is copied under the `\Windows` directory on PocketPC during installation. If you want to display different helpfiles or customize help for every screen in your application, then SODA Forms calls the `frm.setHelpFile` method where you can pass in your help file name. Your custom help file should be located under the `\Windows` directory on your PocketPC. You may include images and hyperlinks in your help file.

12.8.2 Menus

For PocketPC, SODA Forms uses the PocketPC Menubar, which is a combination of menus and the toolbar. The default Menubar comes with the SODA Forms and includes the "Edit" menu and tool bar buttons ("Left", "Right", "New", "Delete", "Save", "Revert", "Query", "Help"). The menubar can be created in the resource editor, so you can create your own or modify the default one. Read the MSDN library on how to create Menubar resources.

Remember to keep the same identifiers for the Menubar itself (`IDR_DBFORM_MENUBAR`) and the predefined menu items. You can locate these identifiers in the file `SodaRes.h`, which comes with the SODA Forms source code. You should have all the menu and toolbar items that are in the default menubar with the identifiers defined in `SodaRes.h`. You can modify toolbar icons, caption, and place toolbar items in your menus, as long as you keep the same identifiers.

To create or modify the existing menubar within SodaForms, then rebuild the project and copy the new `SodaForm.dll` on the device for the changes to take effect.

12.9 Displaying a List Of Objects in a Table

SODA Forms allows one table control on a form to be bound to a list of objects. This list can be specified by the user, using one of the overloaded `setObjects` methods in the `DBFormCol` class. You can also bind a table to a database column that contains a list of objects. SODA supports the object pointer array datatype directly, but only normalized relational data can be replicated to the Mobile Server. Use `DBValueRel` virtual columns to map master-detail relationships into pointers.

To bind a table to the database column, perform the following:

- Add the column name for the PocketPC environment by adding the column name and the `ListView` resource identifier to the `DBFormCols` array that you pass to the `DBForm` constructor.
- Call the `setTableInfo` method in the corresponding `DBFormCol` object to specify which columns of the objects are to be displayed in the table, what are the column titles, and how the data is to be aligned.

Users can scroll through the table, click on the headers to sort on the column value and click a row to select it. In this case, the `edit` call returns with the object identifier of the selected row and executing the `getSelectedPos` method on the `DBFormCol` returns the row selected or `DB_NEW_POS`, if the user clicked on an empty space. These events are suppressed if the form is dirty.

The application handles any table click. However, you should load the list of objects into another `DBForm`, let user edit it, and then save the changes back to the table and to the corresponding database column. `DBForm` contains an overloaded `edit(DBFormCol)` method that does all the work automatically for you if you only have a single level of a master-detail relationship. For more complicated cases, you could do the following:

1. Maintain a stack of `DBForm` objects.
2. Call the `DBForm::load(DBFormCol)` method when you push a new form on the stack.
3. Call `DBFormCol::saveTo(DBFormCol)` before the pop.

12.10 SODA Forms UI Controls

SODA Forms supports multiple kinds of UI resources. For each, the values stored in the database and editing behavior can be customized by specifying an edit type in the third field of `DBFormCols`, as described in Section 12.6.5, "Binding UI to Data in the PocketPC Environment", and optionally calling methods on the corresponding `DBFormCol` object.

The form developed on PocketPC can have controls that are not bound to database, such as push buttons. Clicking on a button returns from the edit method with the button Id. When developing on PocketPC, use the following control types:

Table 12–3 Control Types for PocketPC

Control Type	Edit Behavior	Value stored in database
Checkbox	The checkbox can be checked on or off. If Tri-state style is specified, then the checkbox can have a greyed-out appearance if the value is unknown.	True, false or null (for unknown value) is stored in the database.
ComboBox	This is a combination of a listbox and the edit control. `Drop List` is a popup list. `Dropdown` is a popup list that contains edit control for editing the selection. Multiple selections are not supported for the Combobox.	Same as listbox. For "Dropdown" format, with the default edit type, the strings which are not in the original list can be loaded into the edit area of the combo box or stored in the database.
Edit	This is a text field for entering characters. Different styles can be specified in the resource editor, to change the appearance and the set of characters that can be entered into the field. For example, the `Number` style only allows numbers to be entered.	Any trailing spaces or newlines will be stripped from user's input. If the result is an empty string, null value will be stored in the database. Otherwise, the trimmed result will be stored. If the control is bound to a numeric column (control should have "Number" style), the entered number will be stored.
Date-Time Picker	The control stores and shows date and/or time values. It displays a month calendar to modify the value. You can use any of the following formats: `Short Date`, `Long Date`, `Time`, or a custom format. To set the custom format, call the `tbl[resId].setDateFormat` method with the custom format string. See the MSDN library for the `DateTime_SetFromat` method on how to construct the custom format. Use the `Show None` style to null the date/time. This places a check box on the left and the control has an unspecified (null) value when the check box is not checked. The `Allow Edit` style enables manual editing of the date string.	Date/time value is stored in the database. If the control has the `Show None` style and is in unspecified state, then a null value is stored.
Listbox	Displays a list of text items. Set iether single or multiple selection type in the resource editor. The latter is useful for the query mode when you want to query by multiple choices within the list.	Single-selection listbox: If the column edit type is default, the string of the selected item is stored in the database. On input, unknown values or null are shown as an empty string. To allow user to choose a null value, include an empty string as one of the list selections. If the column edit type is DBFormEditListIndex, 0-based index of the selection is stored in the database. This mode is useful where the selection strings are language specific but the database column should always have the same values. Multiple-selection listbox: List selections are stored as array of strings or numbers (when edit type is DBFormEditListIndex). Remember that DBData can store arrays of values of certain type. If there are no selections, null value is stored.

13 Oracle Database Lite ADO.NET Provider

The following sections discuss the Oracle Database Lite ADO.NET provider for Microsoft .NET and Microsoft .NET Compact Framework. The Oracle Database Lite ADO.NET provider resides in the `Oracle.DataAccess.Lite` namespace.

- Section 13.1, "Discussion of the Classes That Support the ADO.NET Provider"
- Section 13.2, "Limitations for the ADO.NET Provider"
- Section 13.3, "Developing an ADO.NET Application on WinCE"

13.1 Discussion of the Classes That Support the ADO.NET Provider

The Oracle Database Lite ADO.NET driver is implemented in the following assembly:

- Windows—`Oracle.DataAccess.Lite.dll`
- Windows CE—`Oracle.DataAccess.Lite_wce.dll`

If you are building an application that uses the ADO.Net driver, you must package the driver with the application files. The assembly DLL must be located in the same directory as your application executable (`*.exe` file). Alternatively, you can add the driver to the Windows global assembly cache. See the Microsoft documentation on how to add this DLL to the Global Assembly Cache.

The following sections describe classes for the Oracle Database Lite ADO.NET provider:

- Section 13.1.1, "Establish Connections With the OracleConnection Class"
- Section 13.1.2, "Transaction Management"
- Section 13.1.3, "Create Commands With the OracleCommand Class"
- Section 13.1.4, "Maximize Performance Using Prepared Statements With the OracleParameter Class"
- Section 13.1.5, "Large Object Support With the OracleBlob Class"
- Section 13.1.6, "Data Synchronization With the OracleSync Class"
- Section 13.1.7, "Creating a Database for Testing"

13.1.1 Establish Connections With the OracleConnection Class

The `OracleConnection` interface establishes connections to Oracle Database Lite. This class implements the `System.data.IDBConnection` interface. When constructing an instance of the `OracleConnection` class, implement one of the following to open a connection to the back-end database:

- Pass in a full connection string as described in the Microsoft ODBC documentation for the `SQLDriverConnect` API, which is shown below:

  ```
  OracleConnection conn = new OracleConnection
        ("Data_Directory=\\orace;Database=polite;DSN=*;uid=system;pwd=manager");
  conn.Open();
  ```

- Construct an empty connection object and set the `ConnectionString` property later.

With an embedded database, we recommended that you open the connection at the initiation and leave it open for the life of the program. When you close the connection, all of the `IDataReader` cursors that use the connection are also closed.

13.1.2 Transaction Management

By default, Oracle Database Lite connection uses the autocommit mode. If you do not want the autocommit to be on, then you can start a transaction with the `BeginTransaction` method in the `OracleConnection` object. The `BeginTransaction` method returns a reference to the `IDbTransaction` object. Then, when finished, execute either the `Commit` or `Rollback` methods on the returned `IDbTransaction`, which either commits or rolls back the transaction. Once the transaction is completed, the database is returned to autocommit mode.

Whenever you rollback a transaction, it rolls back all the operations that you have performed before. Sometimes you need to undo the transaction to a certain point. For this scenario, you can use the Save Points functionality. Save Points allows you to rollback a transaction to a certain point. Oracle Lite supports Save Points. Within a transaction, you can set up, remove or undo any number of Save Points using SQL statements. Using save points gives you better granular control over the transaction. Within the transaction, use SQL syntax to set up, remove and undo savepoints.

For WinCE devices, Oracle Database Lite supports only one process to access a given database. When a process tries to connect to a database that is already in use, the `OracleConnectionOpen` method throws an `OracleException`. To avoid this exception being thrown, close a connection to allow another process to connect.

The following is an example in turning off autocommit for a C# application:

```
OracleConnection conn = new OracleConnection ("DSN=consroot;uid=system");
conn.Open();
IDbTransaction trans = conn.BeginTransaction(); // Turn off AUTOCOMMIT
OracleCommand cmd = (OracleCommand)conn.CreateCommand();
cmd.CommandText = "create table TEST1 (c0 number)";
cmd.ExecuteNonQuery();
trans.Commit(); // AutoCommit is 'ON'
cmd.Dispose();
conn.Close();
```

The following is an example in turning off autocommit for a VB.NET application:

```
conn = New Oracle.DataAccess.Lite.OracleConnection(("DSN=consroot;uid=system")
conn.Open()
IDbTransaction trans = conn.BeginTransaction()
OracleCommand cmd = New OracleCommand (conn)
cmd.CommandText = "create table TEST1 (c0 number)"
cmd.ExecuteNonQuery()
trans.Commit()
cmd.Dispose()
conn.Close()
```

13.1.3 Create Commands With the OracleCommand Class

The `OracleCommand` class implements the `System.Data.IDBCommand` interface. Create any commands through the `CreateCommand` method of the `OracleConnection` class. The `OracleCommand` has constructors recommended by the ADO.NET manual, such as `OracleCommand(OracleConnection conn, string cmd)`.

However, if you use the `OracleCommand` constructors, it is difficult to port the code to other platforms, such as the ODBC provider on Windows 32. Instead, create the connection and then use interface methods to derive other objects. With this model, you can either change the provider at compile time or use the reflection API at runtime.

13.1.4 Maximize Performance Using Prepared Statements With the OracleParameter Class

Parsing a new SQL statement can take significant time; thus, use prepared statements for any performance-critical operations. Although, `IDbCommand` has an explicit `Prepare` method, this method always prepares a statement on the first use. You can reuse the object repeatedly without needing to call `Dispose` or change the `CommandText` property.

13.1.4.1 SQL String Parameter Syntax

Oracle Database Lite uses ODBC-style parameters in the SQL string, such as the `?` character. Parameter names and data types are ignored by the driver and are only for the programmer's use.

For example, assume the following table:

```
create table t1(c1 int, c2 varchar(80), c3 data)
```

You can use the following parameters in the context of this table:

```
IDbCommand cmd = conn.CreateCommand();
cmd.CommandText  = "insert into t1 values(?,?,?);"
cmd.Parameters.Add("param1", 5);
cmd.Parameters.Add("param2", "Hello");
cmd.Parameters.Add("param3", DateTime.Now);
cmd.ExecuteNonQuery();
```

Note: The relevant class names are `OracleParameter` and `OracleParameterCollection`.

13.1.5 Large Object Support With the OracleBlob Class

The `OracleBlob` class supports large objects. Create a new `OracleBlob` object to instantiate or insert a new BLOB object in the database, as follows:

```
OracleBlob blob = new OracleBlob(conn);
```

Since the BLOB is created on a connection, you can use the `Connection` property of `OracleBlob` to retrieve the current `OracleConnection`.

Functions that you can perform with a BLOB are as follows:

- Section 13.1.5.1, "Using BLOB Objects in Parameterized SQL Statements"
- Section 13.1.5.2, "Query Tables With BLOB Columns"

- Section 13.1.5.3, "Read and Write Data to BLOB Objects"

13.1.5.1 Using BLOB Objects in Parameterized SQL Statements

You can use the BLOB object in parameterized SQL statements, as follows:

```
OracleCommand cmd = (OracleCommand)conn.CreateCommand();
cmd.CommandText = "create table LOBTEST(X int, Y BLOB)";
cmd.ExecuteNonQuery();
cmd.CommandText = "insert into LOBTEST values(1, ?)";
cmd.Parameters.Add(new OracleParameter("Blob", blob));
cmd.ExecuteNonQuery();
```

13.1.5.2 Query Tables With BLOB Columns

You can retrieve the `OracleBlob` object using the data reader to query a table with a BLOB column, as follows:

```
cmd.CommandText = "select * from LOBTEST";
IDataReader rd = cmd.ExecuteReader();
rd.read();
OracleBlob b = (Blob)rd["Y"];
```

Or you can write the last line of code, as follows:

```
OracleBlob b = (OracleBlob)rd.getvalue(1);
```

13.1.5.3 Read and Write Data to BLOB Objects

The `OracleBlob` class supports reading and writing to the underlying BLOB, and retrieving and modifying the BLOB size. Use the `Length` property of `OracleBlob` to get or to set the size. Use the following functions to read and write to the BLOB, as follows:

```
public long GetBytes(long blobPos, byte [] buf, int bufOffset, int len);
public byte [] GetBytes(long blobPos, int len);
public void SetBytes(long blobPos, byte [] buf, int bufOffset, int len);
public void SetBytes(long blobPos, byte [] buf);
```

For example, the following appends data to a BLOB and retrieves the bytes from position five forward:

```
byte [] data = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };
 blob.SetBytes(0, data); //append data to the blob
byte [] d = blob.GetBytes(5, (int)blob.Length - 5); //get bytes from position 5 up
to the end
blob.Length = 0; //truncate the blob completely
```

Use the `GetBytes` method of the data reader to read the BLOB sequentially, but without accessing it as a `OracleBlob` object. You should not, however, use the `GetBytes` method of the reader and retrieve it as a `OracleBlob` object at the same time.

13.1.6 Data Synchronization With the OracleSync Class

You can perform a synchronization programatically with one of the following methods:

- Section 13.1.6.1, "Using the OracleSync Class to Synchronize"
- Section 13.1.6.2, "Using the OracleEngine to Synchronize"

13.1.6.1 Using the OracleSync Class to Synchronize

Note: A `DataException` is thrown if synchronization fails. Also, you must close all database connections before doing a synchronization.

To programmatically synchronize databases, perform the following:

1. Instantiate an instance of the `OracleSync` class.
2. Set relevant properties, such as username, password and URL. The username and password are limited to 28 characters each.
3. Call the `Synchronize` method to trigger data synchronization.

This is demonstrated in the following example:

```
OracleSync sync = new OracleSync();
sync.UserName = "JOHN";
sync.Password = "JOHN";
sync.ServerURL = "mobile_server";
sync.Synchronize();
```

The attributes that you can set are described in Table 13–1.

Table 13–1 OracleSync Attributes

Attibute	Description
`UserName`	Assign a string in quotes with the name of the user for synchronization.
`Password`	Assign a string in quotes with the password for the user.
`ServerURL`	Assign a string in quotes with the Mobile Server host name.
`ProxyHost`	Assign a string in quotes with the host name of the proxy server.
`ProxyPort`	Assign a string in quotes with the port of the proxy server.
`Secure`	Set to true if using SSL; false if not.
`PushOnly`	If true, upload changes from the client to the server only, as download is not allowed. This is useful when the data transfer is a one way transmission from the client to server.
`HighPriority`	Set to true if requesting a high priority synchronization.
`SetTableSyncFlag`	Three arguments are required for `SetTablesyncFlag`, as follows: `sync.SetTableSyncFlag (String pub_name, String tbl_name, boolean remove)` Passing `pub_name`, null `tbl_name`, `remove` = 0 turns off `syncFlag` for everytable in that publication. Passing `pub_name`, `tbl_name`, `remove` = 1 turns on `syncFlag` for that specific table. Thus, you can set synchronization off for all tables, then turn on each individual table that you want to synchronize.

If you want to retrieve the synchronization progress information, set the `SyncEventHandler` attribute of the `OracleSync` class before your execute the `sync.synchronize` method, as follows.

```
sync.SetEventHandler (new OracleSync.SyncEventHandler
                            (MyProgress), true);
```

You pass in your implementation of the `MyProgress` method, which has the following signature:

```
Void MyProgress(SyncStage stage, int Percentage)
```

13.1.6.2 Using the OracleEngine to Synchronize

You can synchronize with the same engine that performs the synchronization for the `msync` tool. You can actually launch the GUI to have the user enter information and click **Synchronize** or you can enter the information programmatically and synchronize without launching the GUI.

13.1.6.2.1 Launch the msync Tool for User Input You can launch the `msync` tool, so that the user can modify settings and initialize the synchronization, by executing the following:

```
OracleEngine.Synchronize(false)
```

Providing the `false` as the input parameter tells the engine that you are not providing the input parameters, but to bring up the `msync` GUI for the user to input the information.

13.1.6.2.2 Set the Environment and Synchronize With the OracleEngine You can set the information and call for a synchronization through the `OracleEngine` class without bringing up the GUI.

If you accept the default synchronization settings, provide `true` as the input parameter to automatically synchronize, as follows:

```
OracleEngine.Synchronize(true)
```

You can execute the synchronize method with three input parameters that define a specific server: the server name, username and password.

```
OracleEngine.Synchronize("S11U1", "manager", "myserver.mydomain.com")
```

Alternatively, you can configure a string that contains the options listed in Table 13–2 with a single `String` input parameter and synchronize, as follows:

```
OracleEngine.Synchronize(args)
```

In the above example, the `String args` input parameter is a combination of the options in Table 13–2.

```
String args = "S11U1/manager@myserver.mydomain.com /save /ssl /force"
```

Include as many of the options that you wish to enable in the `String`.

Table 13–2 Command Line Options

Option	Description
`username/password@server[:port][@proxy:port]`	Automatically synchronize to the specified server.
`/a`	Automatically synchronize to saved preferred server.
`/save`	Save user info and exit.
`/proxy:(proxy_server)[:port]`	Connect by specific proxy server and port.
`/ssl`	Synchronize with SSL encryption.

Table 13–2 (Cont.) Command Line Options

Option	Description
`/force`	Force refresh.
`/noapp:(application_name)`	Do not synchronize specific Web-to-Go application data. Synchronize with other applications.
`/nopub:(publication_name)`	Do not synchronize specific publication data. Synchronize with other publications.
`/notable:(table_name)` `/notable:(odb_name).(table_name)`	Do not synchronize specific table data. Synchronize with other tables.
`/onlyapp:(application_name)`	Synchronize only specific Web-to-Go application data. Do not synchronize with other applications.
`/onlypub:(publication_name)`	Synchronize only specific publication data. Do not synchronize with other publications.
`/onlytable:(table_name)` `/onlytable:(odbc_name).(table_name)`	Synchronize only specific table data. Do not synchronize with other tables.
`/hp`	Enable high priority data synchronization.

13.1.7 Creating a Database for Testing

In a non-production environment, you may want to create a database to test your ADO.NET application against. In the production environment, the database is created when you perform the `OracleEngine.Synchronize` method (see Section 13.1.6.2, "Using the OracleEngine to Synchronize" for more information). However, to just create the database without synchronization, you can use the `CreateDatabase` method of the `OracleEngine` class. To remove the database after testing is complete, use the `RemoveDatabase` method. These methods are only supported when you install the Mobile Development Kit (MDK).

Note: Use the CAB file provided with the MDK.

The following is the signature of the `CreateDatabase` method:

```
OracleEngine.CreateDatabase (string dsn, string db, string pwd)
```

13.2 Limitations for the ADO.NET Provider

The following are limitations to the Oracel Database Lite ADO.NET provider:

- Section 13.2.1, "Partial Data Returned with GetSchemaTable"
- Section 13.2.2, "Creating Multiple DataReader Objects Can Invalidate Each Other"
- Section 13.2.3, "Calling DataReader.GetString Twice Results in a DbNull Object"
- Section 13.2.4, "Thread Safety"

13.2.1 Partial Data Returned with GetSchemaTable

The Oracle Database Lite ADO.NET provider method—`GetSchemaTable`—only returns partial data. For example, it claims that all of the columns are primary key, does not report unique constraints, and returns null for `BaseTableName`, BaseSchemaName and BaseColumnName. Instead, to retrieve Oracle Database Lite meta information, use ALL_TABLES and ALL_TAB_COLUMNS instead of this call to get Oracle Database Lite meta information.

13.2.2 Creating Multiple DataReader Objects Can Invalidate Each Other

The Oracle Database Lite ADO.NET provider does not support multiple concurrent DataReader objects created from a single OracleCommand object. If you need more than one active DataReader objects at the same time, create them using separate OracleCommand objects.

The following example shows how if you create multiple DataReader objects from a single OracleCommand object, then the creation of reader2 invalidates the reader1 object.

```
OracleCommand cmd = (OracleCommand)conn.CreateCommand();
cmd.CommandText = "SELECT table_name FROM all_tables";
cmd.Prepare();
IDataReader reader1 = cmd.ExecuteReader();
IDataReader reader2 = cmd.ExecuteReader();
```

13.2.3 Calling DataReader.GetString Twice Results in a DbNull Object

Calling the GetString method of DataReader twice on the same column and for the same row results in a DbNull object. The following example demonstrates this in that the second invocation of GetString results in a DbNull object.

```
 IDataReader dr = cmd.ExecuteReader();
        String st = null;
        while(dr.Read())
        {
                st = dr.GetString (1);
                st = dr.GetString (1);
        }
```

13.2.4 Thread Safety

To build a thread-safe program, make sure that different threads use separate IDbCommand and IDataReader objects. The OracleConnection and IDbTransaction methods can be called concurrently, except for when used to open and close the connection.

13.3 Developing an ADO.NET Application on WinCE

For an example of how to develop an ADO.NET application, see Chapter 20, "Tutorial for Building Mobile Applications for Windows CE".

14 Using Symbian Devices

Symbian support is a relatively new addition to the supported devices in Oracle Database Lite. As such, not all functionality that exists for other devices is available for Symbian devices. This chapter helps to show what is supported. Use the rest of the documentation for more details on those areas.

- Section 14.1, "Support Symbian Devices in Oracle Database Lite"
- Section 14.2, "Invoke Synchronization from Applications on Symbian Devices"
- Section 14.3, "Using a JDBC Driver for J2ME CLDC to Connect to the Database"
- Section 14.4, "Use the Utility Tools on Symbian Devices"

14.1 Support Symbian Devices in Oracle Database Lite

When you are developing applications for the Symbian environment, you can use the following:

- For your development language, you can use either C or C++ APIs.
- Symbian applications that need a standard interface and work with multiple database engines can use either the Open Database Connectivity (ODBC) interface or some other interface built on top of ODBC.

14.2 Invoke Synchronization from Applications on Symbian Devices

The following sections describes how to set up your application to use the synchronization APIs for use on a Symbian device. Also, see Section 4.1, "Synchronization APIs For C or C++ Applications" for information on how to use the C or C++ APIs available to start synchronization programmatically within your application.

- Section 14.2.1, "How To Write A Program Using Oracle Database Lite 10g"
- Section 14.2.2, "Prepare Your Application for Synchronization"
- Section 14.2.3, "How to Use the Synchronization API for Symbian Devices"

14.2.1 How To Write A Program Using Oracle Database Lite 10g

Note: For an example, see the CSQL example in the `<EPOCROOT>\OliteEx\CoreDB\CSQL` directory.

Perform the following to enable database (ODBC) functionality:]

1. Include `sql.h` and `sqlext.h` in your source code, as follows:

   ```
   #include <sql.h>
   #include <sqlext.h>
   ```

2. Add the include path `SYSTEMINCLUDE \epoc32\include\olite` in the `.mmp` file.
3. Add the library `LIBRARY olod2040.lib` in the `.mmp` file.
4. Oracle Database Lite 10*g* uses STDLIB resources. You need to call `CloseSTDLIB()` after all database operations to free up resources.
5. Character data stored in Oracle Database Lite 10*g* must be in UTF-8 encoding.

   If you write APP application, then you might need to convert between UCS and UTF-8 encodings back and forth. For more information, refer the Symbian API reference.

   You can use the following two functions to convert between encodings:

   Note: To use these functions, include `utf.h` and link `charconv.lib`.

   - `CnvUtfConverter::ConvertFromUnicodeToUtf8()`
   - `CnvUtfConverter::ConvertToUnicodeFromUtf8()`

6. Oracle Database Lite 10*g* uses `STDLIB`; thus, you must release all resources after you finish any ODBC operations. To release all resources, perform the following:

   a. Add `#include <sys/reent.h>`.

   b. Invoke the `CloseSTDLIB()` method after each `SQLFreeEnv()` call.

14.2.2 Prepare Your Application for Synchronization

Note: For an example, see the mSync example in the `<EPOCROOT>\OliteEx\Sync\mSync` directory.

---

1. Include `ocapi.h` in your source code, as follows: `#include <ocapi.h>`.
2. Add the include path `SYSTEMINCLUDE \epoc32\include\olite` in your `.mmp` file.
3. Add the library `LIBRARY ocapi.lib` in your `.mmp` file.

14.2.3 How to Use the Synchronization API for Symbian Devices

The Synchronization API does not run under the `eshell.exe.` For starting synchronization, the application performs the following:

1. Invoke the `ocSessionInit()` method.
2. Invoke the `ocDoSynchronize()` method, which will return before the synchronization completes.

3. To determine if the synchronization is complete, the GUI application continues to invoke the `ocGetLastError()` method. If it returns -1, then synchronization is still executing. With any other value, the synchronization is complete.
4. Once synchronization completes, then invoke the `ocSessionTerm()` method.

For an example, see the `msync.cpp` sample code.

14.3 Using a JDBC Driver for J2ME CLDC to Connect to the Database

You can use a JDBC driver for J2ME CLDC—in a limited capacity—for Java applications to connect and update the database. Full details on the JDBC driver for J2ME CLDC, which can be used on Symbian devices is documented in Section 10.8.1.2, "JDBC Driver for J2ME CLDC".

14.4 Use the Utility Tools on Symbian Devices

To use the database utility tools on the emulator, perform the following:

1. Open a command prompt window.
2. Change directory to the `<EPOCROOT>\epoc32\release\wins\udeb` directory.
3. Type the tool name with appropriate arguments. See the Oracle Database Lite 10*g* documentation for more information.

To use the database utility tools on the device, perform the following:

1. Open `eshell.exe` on the device. Consult with the device manufacturer for the `eshell.exe` program.
2. Type the tool name with appropriate arguments. See the Oracle Database Lite 10*g* documentation for more information.

15 Oracle Database Lite Transaction Support

When an application connects to the local client database—Oracle Database Lite—it begins a transaction with the database. There can be a maximum of 64 connections to Oracle Database Lite. Each connection to Oracle Database Lite maintains a separate transaction, which conform to ACID requirements.

A transaction can include a sequence of database operations, such as `SELECT`, `UPDATE`, `DELETE`, and `INSERT`. All operations either succeed and are committed or are rolled back. Oracle Database Lite only updates the database file when the commit is executed. If an event, such as a power outage, interrupts the commit, then the database is restored during the next connection.

- Section 15.1, "Locking"
- Section 15.2, "What Are the Transaction Isolation Levels?"
- Section 15.3, "Configuring the Isolation Level"
- Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types"

15.1 Locking

Oracle Database Lite supports row-level locking. Whenever a row is read, it is read locked. Whenever a row is modified, it is write locked. If a row is read locked, then different transactions can still read the same row. However, a transaction cannot access a row if it is a write locked row by another transaction.

15.2 What Are the Transaction Isolation Levels?

Each transaction is isolated from another. Even though many transactions run concurrently, transaction updates are concealed from other transactions until the transaction commits. You can specify what level of isolation is used within the transaction, as listed in Table 15–1:

Table 15–1 Isolation Levels

Isolation Level	Description
Read Committed	In Oracle Database Lite, a `READ COMMITTED` transaction first acquires a temporary database level read lock, places the result of the query into a temporary table, and then releases the database lock. During this time, no other transaction can perform a commit operation. No data objects are locked. All other transactions are free to perform any DML operation—except `commit`—during this time. Since a commit operation locks the database in `intent` exclusive mode, a read committed transaction, while gathering the query result, will block another transaction that is trying to commit or vice versa. A `READ COMMITTED` transaction provides the highest level of concurrency, as it does not acquire any data locks and does not block any other transaction from performing any DML operations. In addition, the re-execution of the same query (`SELECT` statement) may return more or less rows based on other transactions made to the data in the result set of the query. **Note**: A `SELECT` statement containing the `FOR UPDATE` clause is always executed as if it is running in a `REPEATABLE READ` isolation level. A `SELECT` statement can execute Java stored procedures. If the transaction executing the Java stored procedure is in the `READ COMMITTED` isolation level and the Java stored procedure updates the database, then the `SELECT` statement that executes the Java stored procedure must have a `FOR UPDATE` clause. Otherwise, Oracle Database Lite issues an error. **Note**: If you are retrieving a large object, such as a BLOB, within a READ COMMITTED transaction, see the "Select Statement Behavior When Retrieving BLOBs in a READ COMMMITTED transaction" section in the *Oracle Database Lite SQL Reference*.
Repeatable Read	In this isolation level, a query acquires read locks on all of the returned rows. More rows may be read locked because of the complexity of the query itself, the indexes defined on its tables, or the execution plan chosen by the query optimizer. The `REPEATABLE READ` isolation level provides less concurrency than a `READ COMITTED` isolation level, transaction because the locks are held until the end of the transaction. A "phantom" read is possible in this isolation level, which can occur when another transaction inserts rows that meet the search criteria of the current query and the transaction re-executes the query. If a `FOR UPDATE` clause is used in a query, a short-term update lock is acquired on the current row(s) being selected. If a row is updated, the lock is converted into an exclusive lock. An exclusive lock prevents any other transaction running in an isolation level other than `READ COMMITTED` to access this row. If the row is not updated but the next row is fetched, the update lock is downgraded to a read lock, permitting other transactions to read the row.
Serializable	This isolation level acquires shared locks on all tables participating in the query. The same set of rows is returned for the repeated execution of the query in the same transaction. Any other transaction attempting to update any rows in the tables in the query is blocked.
SingleUser	In this isolation level only one connection is permitted to the database. The transaction has no locks and consumes less memory.

Refer to the documentation for ODBC for more information on isolation levels.

15.3 Configuring the Isolation Level

The default isolation level is `READ COMMITTED`. You can modify the isolation level for a data source name (DSN) by using the ODBC Administrator—which you can bring up by executing `odbcad32`—or by manually editing the `ODBC.INI` file. We recommend that you use the `odbcad32` tool, as it will inform you if you have an incorrect combination of isolation level and cursor type. See Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types" for more information.

When you bring up the ODBC Administrator, under the User DSN tab, double-click the Oracle Lite 40 ODBC driver for which you want to modify the isolation level. Select the default cursor type from the pull-down list.

If you decide to edit the `ODBC.INI` file by hand, then set the isolation level as follows:

```
IsolationLevel = XX
```

where the value for `XX` is Read Committed, Repeatable Read, Serializable, or Single User.

Note: The `ODBC.INI` file is available in Windows under `%WINDIR%` and in Linux under `$OLITE_HOME/bin`. For the Linux platform, you must have write permissions on the directory where this is located to be able to modify them.

Alternatively, you can define the isolation level of a transaction by using the following SQL statement:

```
SET TRANSACTION ISOLATION LEVEL <ISOLATION_LEVEL>;
```

where `ISOLATION_LEVEL` is `READ COMMITTED`, `REPEATABLE READ`, `SERIALIZABLE`, or `SINGLE USER`.

See Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types", for information on how certain isolation levels and scrollable cursors sometimes cannot be used in combination.

15.4 Supported Combinations of Isolation Levels and Cursor Types

If you use the ODBC Administrator—which you can bring up by executing `odbcad32`—then this tool informs you if you are using an incorrect combination of isolation level and cursor type.

We support these types of cursors

- Forward only cursors allow you to only move forward through the returned result set. You cannot go backwards, nor can you view any additional modifications. To return to a row, you would have to close the cursor, reopen it and then move to the row you wanted to see. However, it is the fastest cursor for moving through a result set.
- Scrollable cursors are the most flexible as they allow you to go forward and backward through the returned result set, but are also expensive. The other advantage of using a scrollable cursor is you can see modifications directly after they occur.

The three supported types of scrollable cursors are as follows:

- Static—The result set appears to be static; that is, it does not detect modifications made to the membership, order, or values of the result set after the cursor is opened. This cursor can detect its own modifications, just not the modifications of others.
- Dynamic—Any modifications to the result set can be detected and viewed when the row is re-fetched.
- Keyset Driven—The abilities of this cursor is between the static and dynamic. It can detect modifications to the values in the rows of the result set; however, it cannot detect changes to the membership and order of the result set.

Refer to the documentation for ODBC for more information on cursor types.

For some cursors, you cannot combine them with certain isolation levels. Table 15–2 shows the supported combinations of isolation levels and cursor types. Unsupported combinations generate error messages.

Table 15–2 Supported Combinations

	Forward Only Cursor	**Scrollable Static Cursor**	**Scrollable Keyset Driven Cursor**	**Scrollable Dynamic Cursor**
Isolation Level				
Read Committed	Supported	Supported	Unsupported	Unsupported
Repeatable Read	Supported	Unsupported	Supported	Supported
Serializable	Supported	Unsupported	Supported	Supported
Single User	Supported	Supported	Supported	Supported

16 Oracle Database Lite Security

The following sections detail security issues for Oracle Database Lite:

- Section 16.1, "Authenticating Users With Your Own User Management System"
- Section 16.2, "Providing Your Own Encryption Module for the Client Oracle Lite Database"

16.1 Authenticating Users With Your Own User Management System

You can provide an external authenticator for the Mobile Server to authenticate users with passwords as well as their access privileges to applications. For example, in an enterprise environment, you may have your user data, such as employee information, stored in a LDAP-based directory service. The Mobile Server can retrieve the user information from the LDAP directory—or from any custom User Management System—if configured with your own implementation of an external authenticator. The Mobile Server links the external user information to the Mobile Server repository.

16.1.1 Implementing Your External Authenticator

In order to use an external authenticator, you must implement the `oracle.lite.provider.Authenticator` JAVA interface and configure the implementation in the `webtogo.ora` file.

Note: Sample code for an external authenticator can be found in `<`*`ORACLE_HOME`*`>\Mobile\Server\samples\devmgr\java\SampleAuthenticator.java`

Implement the following methods in your external authenticator. The Mobile Server invokes each of these methods as appropriately.

- Section 16.1.1.1, "The Initialization Method for the External Authenticator"
- Section 16.1.1.2, "The Destruction Method for the External Authenticator"
- Section 16.1.1.3, "The Authentication Method for the External Authenticator"
- Section 16.1.1.4, "The User Instantiation Method for the External Authenticator"
- Section 16.1.1.5, "Retrieve the User Name or the User Global Unique ID"
- Section 16.1.1.6, "Log Off User"
- Section 16.1.1.7, "Change User Password"

16.1.1.1 The Initialization Method for the External Authenticator

Mobile Server invokes the `initialize` method before calling any other method of provider class. This method will be called only once when the provider is initialized.

```
Method: void initialize (String metaData) throws Exception
Parameter: String metaData (Reserved for future use)
```

16.1.1.2 The Destruction Method for the External Authenticator

Mobile Server invokes the `destroy` method when the system shutdowns. Provider implementation should implement all the cleanup code in this method.

```
Method: void destroy() throws Exception
Parameter: None
```

16.1.1.3 The Authentication Method for the External Authenticator

Authenticate a user and return a session handle. The returned session handle is passed to the `logOff` method when the user logs off from the system. Note that the `logOff` method may not be called for each successful `authenticate` method call. Some of the Mobile Server clients may use the `authenticate` method to verify the user credential and not for logging on to the system.

```
Method: Object authenticate (String uid, String pwd) throws SecurityException
Parameter: User Id (or User Name) and password string
Return: Session handle or null
```

16.1.1.4 The User Instantiation Method for the External Authenticator

If the user has not been instantiated in the Mobile Server repository, then the Mobile Server invokes the `getInitializationScripts` method—after authenticating the user—to retrieve the initialization scripts for the user. The Mobile Server uses the initialization scripts to instantiate the user in the Mobile Server and assign access rights to applications and data. See Section 16.1.3, "User Initialization Scripts" for more information.

```
Method: StringBuffer getInitializationScripts (Object sid)
Parameter: Session handle returned by 'authenticate' method
Return: 'StringBuffer' containing User's initialization scripts
```

16.1.1.5 Retrieve the User Name or the User Global Unique ID

Return the user name or GUID (Globally Unique Id) of the user if there is one. Usually, LDAP-based User Management systems maintain a GUID for each user. In case your authentication mechanism does not support GUID, then the `getUserGUID` method returns `NULL`.

```
Method: String getFullName (Object sid)
Parameter: Session handle returned by 'authenticate' method
Return: User's full name

Method: String getUserGUID (Object sid)
Parameter: Session handle returned by 'authenticate' method
Return: User's GUID or null
```

16.1.1.6 Log Off User

Log off the User from the back-end system. Note that the `logOff` method may not be called for each successful `authenticate` method call. Some of the Mobile Server clients may use the `authenticate` method to verify the user credential and not for logging on to the system.

```
Method: void logOff (Object sid) throws SecurityException
Parameter: Session handle returned by 'authenticate' method
```

16.1.1.7 Change User Password

```
Method: void changePassword (Object sid, String pwd) throws SecurityException
Parameter: Session handle returned by the authenticate method and new password
string
```

16.1.2 Registering External Authenticator

To register your external authenticator, modify the `webtogo.ora` file and set your external `Authenticator` JAVA class name in the `EXTERNAL_AUTHENTICATION` section, as follows:

```
[EXTERNAL_AUTHENTICATION]
CLASS  = SampleAuthenticator
EXPIRATION = 1800
```

The Mobile Server caches the user instantiated through the external authenticator for a period of time in order to improve efficiency. The default expiration time for the cached user object is 30 minutes (or 1800 seconds). Customize this value by setting a new value for the `EXPIRATION` parameter.

16.1.3 User Initialization Scripts

Mobile Server invokes the `getInitializationScripts` method to retrieve the user initialization script that instantiates user-specific objects in the Mobile Repository. The external authenticator can perform the following actions during the initialization process:

1. Assign access rights to applications
2. Set data subscription parameters.
3. Optionally, add the user to a user group.

The syntax of the initialization script is based on the `INI` format. The first section in the script is as follows.

```
[MAIN]
VERSION=2
```

The following example performs these actions for a user whose id is `USER1`.

1. Assigning access rights to applications.

   Assign access rights to `Application1` and `Application2` for `USER1`, where `Application1` has two publication items and three subscription parameters.

   ```
   # List the applications we want access to
   #
   [ACL]
   Application1
   Application2
   # List Access details for 'Application1'
   #
   [ACL.Application1]
   NAME=USER1
   TYPE=USER
   DATA=LOCATION, ITEMS
   # List Access details for 'Application2'
   ```

```
#
[ACL.Application2]
NAME=USER1
TYPE=USER
```

2. Setting data subscription parameters.

```
[SUBSCRIPTION.USER1.Application1.LOCATION]
NAME=ZIP, USR_ID
VALUE=12345, USER1
[SUBSCRIPTION.USER1.Application1.ITEMS]
NAME=WEIGHT
VALUE=20
```

3. Adding a User to a User Group

```
[GROUP]
User's Group
[GROUP.User's Group]
USER=USER1
```

16.2 Providing Your Own Encryption Module for the Client Oracle Lite Database

The database on the Mobile client—also known as the Oracle Lite database—uses Advanced Encryption Standard (AES) for encrypting the database. However, you can provide your own encryption module for the client database.

The following sections describe how to implement and plug-in your own encryption module.

- Section 16.2.1, "Encryption Module APIs"
- Section 16.2.2, "Plug-In Custom Encryption Module"

16.2.1 Encryption Module APIs

Oracle Database Lite invokes your encryption APIs when performing encryption duties, instead of the internal AES encryption module. Thus, you must develop and include the following APIs in your customized encryption module:

- Section 16.2.1.1, "Initialize the Encryption Module"
- Section 16.2.1.2, "Delete Encryption Context"
- Section 16.2.1.3, "Create the Encryption Key"
- Section 16.2.1.4, "Encrypt Data"
- Section 16.2.1.5, "Decrypt Data"

Note: All of the functions in this section are in Windows format. Adjust appropriately if developing on a UNIX environment.

16.2.1.1 Initialize the Encryption Module

Implement the `encCreateCtxt` function to initialize the external encryption module. Oracle Database Lite invokes this function when initializing encryption. This function returns an encryption context handle to Oracle Database Lite, which it passes back on all subsequent API calls. The context handle is displayed as a `void*`, so that you can make it any type of structure you desire.

```
__declspec(dllexport) void* encCreateCtxt()
```

16.2.1.2 Delete Encryption Context

When Oracle Database Lite is finished with the encryption module, it invokes the `encDeleteCtxt` function to delete the encryption context—which was created with the `encCreateCtxt` function.

```
__declspec(dllexport) void encDeleteCtxt(voie * ctx);
```

16.2.1.3 Create the Encryption Key

Oracle Database Lite invokes your `encCreateKey` function to create the encryption key within the encryption context, as follows:

```
__declspec(dllexport) void encCreateKey (void* ctx, const unisgned char* key, int
len, int dir);
```

Where the input parameters are as follows:

- `ctx`—The encryption context, which is created in the `encCreateCtxt` function.
- `key`—Pointer to the key to be created.
- `len`—Length of the encryption key.
- `dir`—Encryption direction or type, where 1: encryption, 2: decryption, 3: both encryption and decryption.

16.2.1.4 Encrypt Data

Oracle Database Lite invokes your `encEncryptData` function to encrypt the data that is to be sent, as follows:

```
__declspec(dllexport) void encEncryptData (void* ctx, const unisgned char* data,
int len, unsigned char* out);
```

Where the input parameters are as follows:

- `ctx`—The encryption context, which is created in the `encCreateCtxt` function.
- `data`—Pointer to the data to be encrypted.
- `len`—Length of the data in bytes.
- `out`—Output buffer.

This function returns the number of bytes copied to the output buffer.

16.2.1.5 Decrypt Data

Oracle Database Lite invokes your `encDecryptData` function to decrypt the data that it receives. This function copies the result to the output buffer.

```
__declspec(dllexport) void encDecryptData (void* ctx, const unisgned char* data,
int len, unsigned char* out);
```

Where the input parameters are as follows:

- `ctx`—The encryption context, which is created in the `encCreateCtxt` function.
- `data`—Pointer to the data to be decrypted.

- `len`—Length of the data in bytes.
- `out`—Output buffer.

This function returns the number of bytes copied to the output buffer.

16.2.2 Plug-In Custom Encryption Module

Once implemented, you can plug-in your custom encryption module by adding the `[All Databases]` section to the `POLITE.INI` configuration file. You must either implement your encryption module into a DLL for the Windows environment or into a Shared Object (.SO) for the UNIX environment.

For example, if you created the encryption module as a DLL called `my_enc.dll`, which is located in the `C:\my_dir` directory, then you would add this module as the default encryption module in the `POLITE.INI` configuration file, as follows:

```
[All Databases]
EXTERNAL_ENCRYPTION_DLL=C:\my_dir\my_enc.dll
```

17 Tutorial for Building Mobile Web-to-Go Applications

There are two types of Web-to-Go applications:

- The original Oracle Database Lite Web-to-Go application that uses an Oracle Database Lite Servlet stack. You can still use this type of application, but the Oracle Database Lite Server stack is not J2EE 1.3 compatible.
- A Web-to-Go application built upon the OracleAS OC4J stack. Since the OC4J product is continually updated, then building your Web-to-Go application using the J2EE standards is better if you want to use future J2EE standards. This application is known as the OC4J Web-to-Go application.

  To build the OC4J Web-to-Go application, follow the J2EE standards specified by Sun Microsystems and then create the snapshot with MDW and publish the application with the EAR or WAR file within the Packaging Wizard.

This tutorial demonstrates how to build, package and publish the original Oracle Database Lite Web-to-Go application with the "To Do List" demo. For details on how to create an OC4J Web-to-Go application, refer to the OC4J documentation or to the Sun Microsystems J2EE specification.

The "To Do List" application maintains a list of "To Do" items with status for each item indicating its completion. All items are stored in the Oracle database. Multiple users can access the "To Do List" application to display their corresponding To Do items.

Note: For more information on developing Web-to-Go applications, see Chapter 6, "Developing Mobile Web-to-Go Applications".

The following sections in this tutorial guide you through the phases of implementing a Web-to-Go application for Mobile devices:

- Section 17.1, "Develop the Application"
- Section 17.2, "Create Publication for Application"
- Section 17.3, "Package the Application Using the Packaging Wizard"
- Section 17.4, "Administer the Application"
- Section 17.5, "Execute the Application on the Mobile Client for Web-to-Go"

17.1 Develop the Application

The first step is to develop and test the "To Do List" application using the Mobile Development Kit for Web-to-Go. Table 17–1 shows the components for the "To Do List" application:

Table 17–1 "To Do List" Application Components

Component	Function
Java Servlet	Accesses the database and inserts To Do items.
Java Server Page (JSP)	Provides the "To Do List" application user interface in HTML.
JavaBean	Provides database access to the JSP.

The source code for the "To Do List" application is installed along with the Mobile Development Kit. It can be found at the following location.

```
<ORACLE_HOME>\Mobile\Sdk\wtgsdk\src\tutorial
```

- The JavaServer Page—The `ToDoList.jsp` generates an HTML page which displays the list of items that must be completed.
- The JavaBean—The `ToDoBean.java` JSP uses a JavaBean to perform operations with the Oracle database. Y
- The Java Servlet—The `InsertToDo.java` Java Servlet inserts a new To Do Item in the Oracle database, and uses the "To Do List" JSP to regenerate the HTML page.

In this section, the following tasks are discussed.

- Section 17.1.1, "Create Database Objects in the Oracle Server"
- Section 17.1.2, "Compile the Application"

The Mobile Development Kit for Web-to-Go always uses Oracle Database Lite as the development database and a Web-to-Go server, which is known as the Mobile Client Web Server.

17.1.1 Create Database Objects in the Oracle Server

During deployment, the Mobile Server automatically creates the Oracle Database Lite database in the client device along with the requisite tables and data. To publish the application, users must create the database objects used by the application in the back-end Oracle database.

The "To Do List" application uses the following database objects.

- The `TODO_ITEMS` table—The application stores To Do Items in this database table. Table 17–2 shows the To Do Items table columns.

Table 17–2 The TODO_ITEMS Table

Column	Function
`ID`	Primary key
`TODO_ITEM`	Text describing the To Do item
`USERNAME`	Owner of the To Do item
`DONE`	Indicates whether or not the To Do item has been completed

- The TODO_SEQ sequence—Each time a user inserts a new record in the TODO_ITEMS table, the TODO_SEQ sequence generates a primary key value for the new record.

17.1.1.1 Create the Table Owner Account

Create the database user who will own the "To Do List" application objects in the Oracle database. If you have installed the samples during your Mobile Server installation, you can skip this step and continue with the next step. If you have not installed the samples, enter the following commands using the Command Prompt.

> **Note:** The CONNECT_STRING is the entry where the database resides, as defined in the tnsnames.ora file, which is used to locate the back-end Oracle database.

```
sqlplus system/<sys_password>@<CONNECT_STRING>
create user master identified by master;
grant CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE, CREATE VIEW, CREATE SESSION,
CREATE INDEXTYPE to master;
```

17.1.1.2 Create the Database Objects in the Oracle Database

In order to execute the To Do List demo, set up the schema and the database objects. We have provided a SQL script that creates the database objects in the back-end database.

To create the database objects, run the tutorial.sql SQL script against the back-end Oracle database, as follows:

```
> cd <ORACLE_HOME>\Mobile\Sdk\wtgsdk\src\tutorial
> msql system/<sys_pwd>@jdbc:oracle:thin:@<host>:<port>:
      <oracle_sid> @tutorial.sql
> msql master/master>@jdbc:oracle:thin:@<host>:<port>:
      <oracle_sid> @tutorial.sql
```

When you execute the tutorial.sql command against the system schema, it creates the TODO_ITEMS and TODO_SEQUENCE in the back-end server database. When you execute the tutorial.sql command against the master schema, then it creates the TODO_ITEMS table and the TODO_SEQUENCE sequence in the user schema also, which in this example is master. This is necessary in order to create the publication item in the master schema.

> **Note:** While entering the above command to create database objects, you must include a mandatory space between "<oracle_sid>" and "@tutorial.sql".

Where:

- <sys_pwd> is the system password. This is required if you are creating the master schema. However, if you have eliminated the statements that create the schema, you can use master/master for username/password.
- <host>:<port> refers to the name and listening port of the machine where the back-end Oracle database is installed.

This script creates the TODO_ITEMS table and the TODO_SEQUENCE sequence on the Oracle database.

17.1.2 Compile the Application

Compile the application by performing the following tasks:

1. Set the CLASSPATH.

   You must set the CLASSPATH to include the required Java Servlet Development Kit and Mobile Server libraries. To include these libraries, this tutorial provides a script called setenv.bat. Using the Command Prompt, enter the following commands.

   ```
   cd <ORACLE_HOME>\Mobile\Sdk\wtgsdk\bin
   setenv.bat
   ```

2. Compile the application.

   You can compile the application manually or by running the compile.bat script. To run the script, start the Command Prompt and enter the following commands.

   ```
   <ORACLE_HOME>\Mobile\Sdk\wtgsdk\src\tutorial
   compile.bat
   ```

   To compile the application manually, perform the following tasks.

   a. Compile the Java Servlet—Using the command prompt, enter the following commands:

      ```
      cd <ORACLE_HOME>\Mobile\Sdk\wtgsdk\src\tutorial
      javac -d ..\..\root\tutorial\WEB-INF\classses\InsertToDo.java
      ```

      This creates the following servlet class file.

      ```
      <ORACLE_HOME>\Mobile\Sdk\wtgsdk\root\tutorial\
      WEB-INF\classses\InsertToDo.class
      ```

   b. Compile the Java Bean—Using the command prompt, enter the following command:

      ```
      javac -d ..\..\root\tutorial\WEB-INF\classes ToDoBean.java
      ```

   c. Install the JSP—Using the command prompt, enter the following command:

      ```
      copy ToDoList.jsp
          <ORACLE_HOME>\Mobile\Sdk\wtgsdk\root\tutorial\ToDoList.jsp
      ```

17.2 Create Publication for Application

As described fully in Chapter 5, "Using Mobile Database Workbench to Create Publications", you can use MDW to create your publication. Launch MDW by executing oramdw from *<ORACLE_HOME>*/Mobile/Sdk/bin. The following sections detail how to use MDW to create a publication for the application in this tutorial.

Note: While creating this publication, use Chapter 5, "Using Mobile Database Workbench to Create Publications" for a deeper understanding of how to use MDW and the type of information that you must provide.

- Section 17.2.1, "Create a Project"
- Section 17.2.2, "Create Publication Items"

- Section 17.2.3, "Create Publication"

17.2.1 Create a Project

Create a new project for this application by selecting **File->New->Project**. This brings up a wizard where you enter the following information:

> **Note:** For more information, see Section 5.2, "Create a Project".

1. Define a name and location for the project.
2. Enter the username, password, JDBC driver type, database host, database port and database SID for the Mobile repository.

   Provide the Mobile Repository access information. Because you are interacting with the repository to create and manipulate synchronization objects, including the SQL scripts for the publication items, you need access to the Mobile Repository. For example, the Mobile Repository username and password is `mobileadmin/welcome123`. The JDBC driver type used is the Oracle Thin driver. The back-end Oracle database host, port, and SID are `mobile-qa11.oracle.com:1521:orcl`.
3. Specify schema username and password. Enter the user and password of the schema owner for the schema that you are using for the Mobile application. The Mobile application schema contains all database tables, views, synonyms used to build the snapshots for the application.
4. Verify the information that you entered and click **Finish**.

17.2.2 Create Publication Items

For this project, you need to create the `todo_items` publication item and a `todo_seq` sequence.

> **Note:** For more information, see Section 5.4, "Create a Publication Item".

The following sections describe how to create the publication item and the sequence for this publication:

- Section 17.2.2.1, "Create Publication Item"
- Section 17.2.2.2, "Create Sequence"

17.2.2.1 Create Publication Item

Perform the following to create the publication item:

1. Start the new publication item wizard by selecting **File->New->Publication Item**.
2. Enter the name as `todo_items` and the type as `Fast`. If you want this publication item to use automatic synchronization, make sure that the "Enable Automatic Synchronization" checkbox is checked. Uncheck to use manual synchronization. Click **Next**.
3. Select the schema name as `MASTER`, the object type as `Table`, and leave the object filter blank. Click **Search**. When the search ends, select `todo_items` from the object list. Click **Next**.

4. Click **>>** to select all of the columns in the `todo_items` table. Click **Next**.
5. In the Query tab, select **Edit** to edit the query, as follows:

   ```
   select * from master.todo_items where username=:username
   ```

   Click **Next**.
6. If you checked the 'Enable Automatic Synchronization' checkbox, then an additional screen comes up. This screen enables you to specify users included in the compose. By default, all users are included. Leave checkbox unchecked and click **Next**.
7. The Summary page displays. Click **Finish**.

17.2.2.2 Create Sequence

Create the `todo_seq` sequence for the To Do List demo, as follows:

1. Start the new sequence wizard by selecting **File->New->Sequence**.
2. Enter the name as `todo_seq` and that the sequence starts with 1, increment of 2, window size of 500, and threshold of 50.
3. Uncheck the **offline only** checkbox, if already checked.

17.2.3 Create Publication

When you have completed the creation of the publication items, create the publication within the project by selecting **File->New->Publication**.

1. In the General tab, enter the name as `todo`, which becomes part of the DSN for the client-side database.
2. In the Publication Item tab, click **Add** to add the publication item that you just created with the following configuration:

   ```
   Name: todo_items
   Updatability: Updatable
   Conflict Resolution: Server Wins
   DML Callback: BLANK
   Grouping Function: BLANK
   Priority Condition: BLANK
   My Compose: BLANK
   Weight: 1
   Description: Blank
   ```

3. In the Sequence tab, click **Add** to add the `todo_seq` that you just created and click **OK**.
4. Optionally, if you do want to set some of the event rules for the publication, then you can select the Events tab to configure the thresholds for Automatic Synchronization rules, such as the following:
   - Sync if number of modified records in database exceeds threshold value
   - Sync if number of modified records in out queue exceeds threshold value
5. Save the publication by selecting **File->Save** and exit MDW.

17.3 Package the Application Using the Packaging Wizard

Using the Packaging Wizard, you can package and publish the To Do List application into the Mobile Server. For more information on how to use the Packaging Wizard, see Chapter 7, "Using the Packaging Wizard".

You can select and describe the To Do List application by launching the Packaging Wizard, as follows:

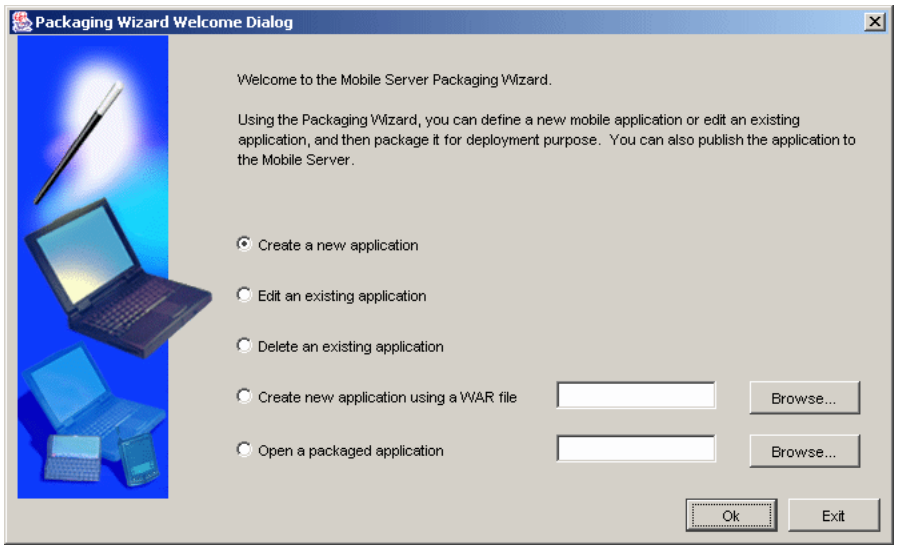
1. Start the Packaging Wizard, as follows:

```
cd <ORACLE_HOME>\Mobile\Sdk\bin
wtgpack
```

   The Packaging Wizard appears and provides you with the option to create a new application, edit an existing application, delete an existing application, or open a packaged application, as displayed in Figure 17–1.

   Note: Deleting an existing application merely deletes the application from the XML file and does not remove the application from the Mobile Server.

Figure 17–1 Make a Selection Dialog



Packaging Wizard Welcome Dialog

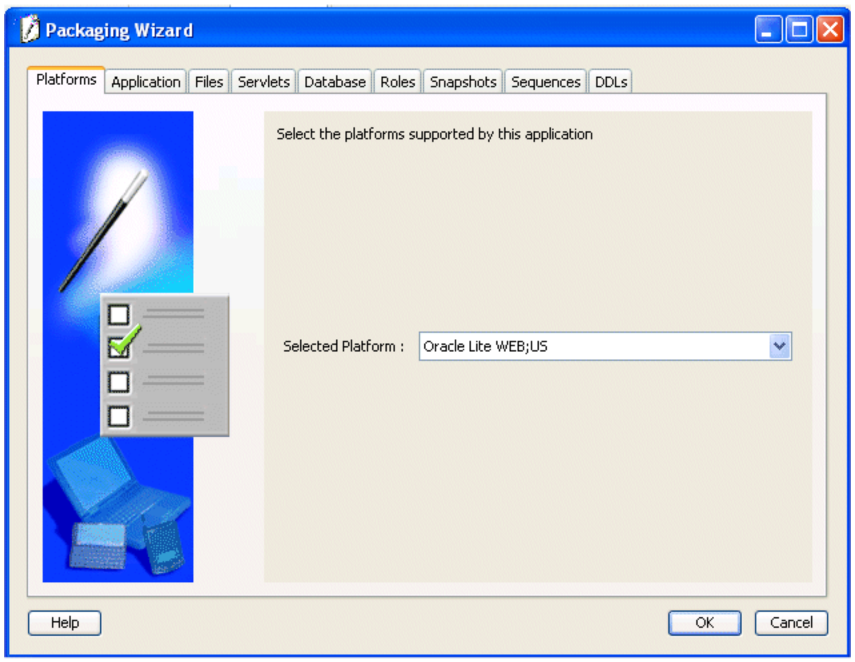
Welcome to the Mobile Server Packaging Wizard.

Using the Packaging Wizard, you can define a new mobile application or edit an existing application, and then package it for deployment purpose. You can also publish the application to the Mobile Server.

[✓ Create a new application]
Edit an existing application
Delete an existing application
Create new application using a WAR file
Open a packaged application

2. Select the **Create a new application** option and click **OK**.
3. The Select a Platform panel appears. As Figure 17–2 displays, this panel enables you to specify the platform for your application. Select **Oracle Lite WEB;US** from the **Available Platform** list. Click **Next**.

Figure 17–2 Selecting a Platform

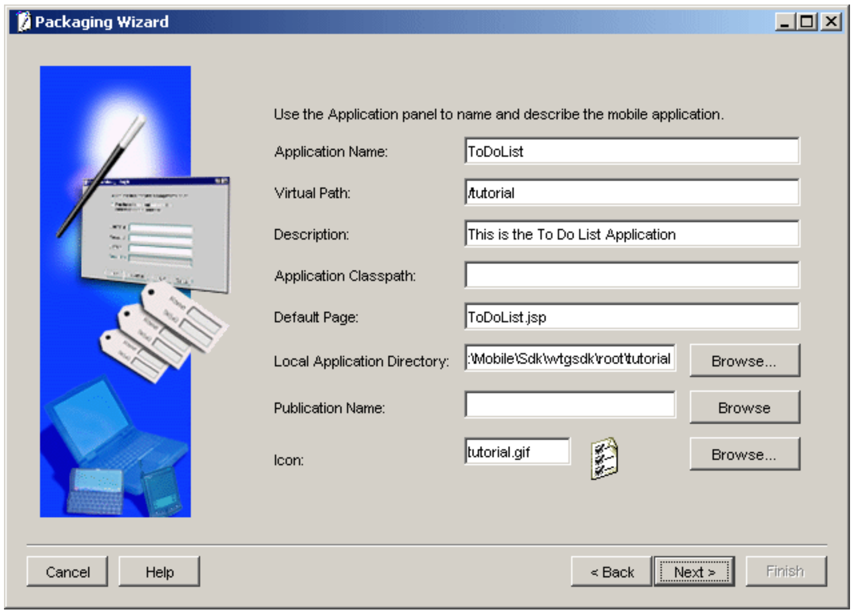


Select the platforms supported by this application

Selected Platform : Oracle Lite WEB;US

4. As Figure 17–3 displays, the Application panel appears. Use the Application panel to modify "To Do List" application settings. As Table 17–3 describes, enter the specified values in the corresponding fields.

Figure 17–3 Application Panel



Use the Application panel to name and describe the mobile application.

Application Name:	ToDoList
Virtual Path:	/tutorial
Description:	This is the To Do List Application
Application Classpath:	
Default Page:	ToDoList.jsp
Local Application Directory:	:\Mobile\Sdk\wtgsdk\root\tutorial
Publication Name:	
Icon:	tutorial.gif

Table 17–3 The "To Do List" Application Values

Field	Value
Application Name	ToDoList
Virtual Path	`/tutorial`

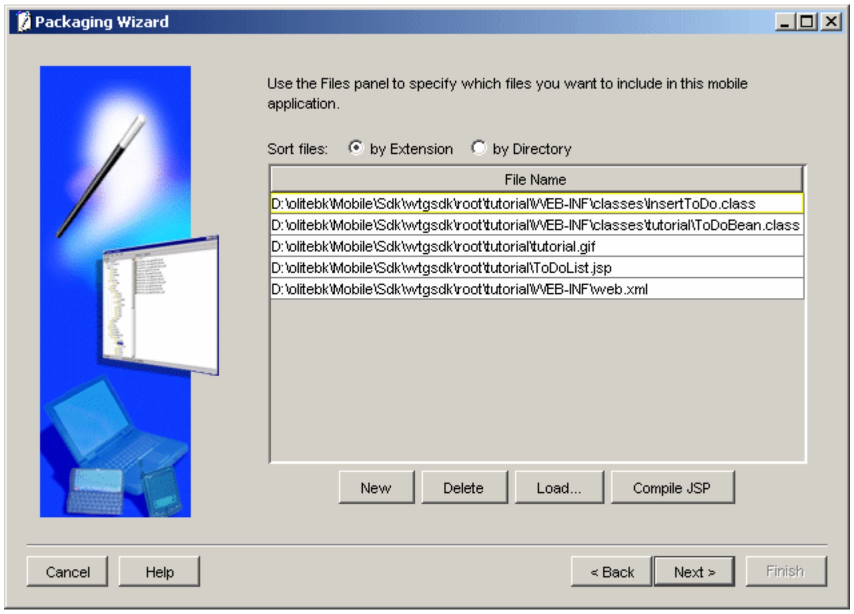
Table 17–3 (Cont.) The "To Do List" Application Values

Field	Value
Description	This is the To Do List Application
Application Classpath	(Leave this field blank)
Default page	`ToDoList.jsp` (this is case sensitive)
Local Application Directory	`<`*`ORACLE_HOME`*`>\mobile\sdk\wtgsdk\root\tutorial`
Publication Name	Click on Browse. The 'Connect to database' window appears. Enter the following: ■ username: `mobiladmin` ■ password: `welcome123` ■ database URL: `jdbc:oracle:thin:@<hostname>:<port>:<SID>` The next window shows the available publications. Select `todo`.
Icon	`tutorial.gif`

5. Click **Next**. As Figure 17–4 displays, the Files panel appears. Using the Files panel, you can select files that are part of the application. The Packaging Wizard uploads the selected files from the local application directory to the application repository on the Mobile Server.

   The Files panel identifies files that the Packaging Wizard uploads from the local application directory to the application repository on the Mobile Server.

Figure 17–4 Uploading Application Files



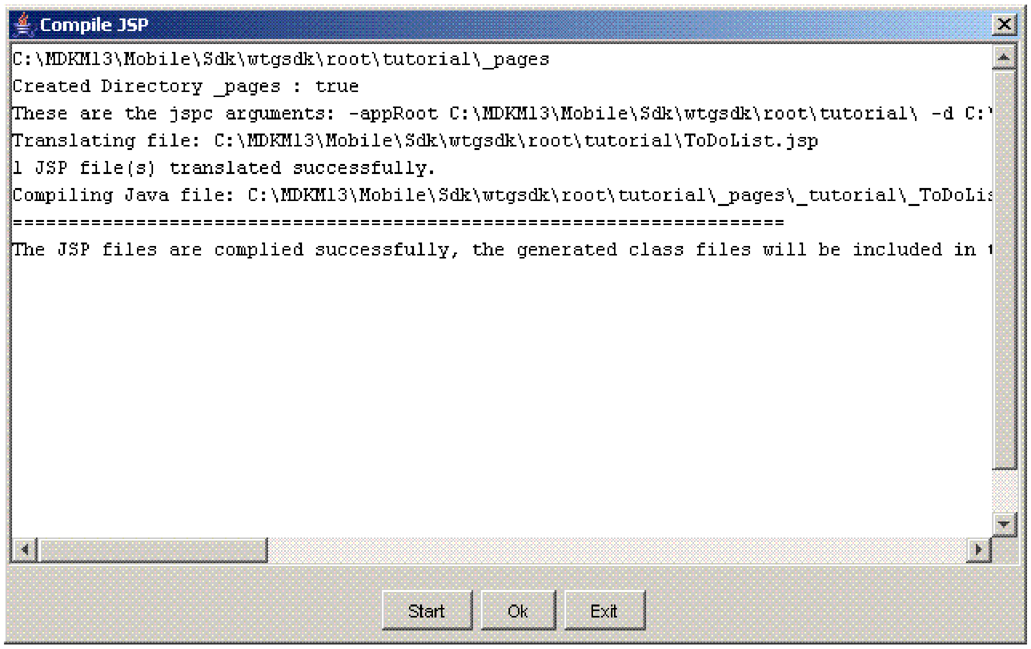
Use the Files panel to specify which files you want to include in this mobile application.

Sort files: [✓ by Extension] by Directory

File Name
D:\olitebk\Mobile\Sdk\wtgsdk\root\tutorial\WEB-INF\classes\InsertToDo.class
D:\olitebk\Mobile\Sdk\wtgsdk\root\tutorial\WEB-INF\classes\tutorial\ToDoBean.class
D:\olitebk\Mobile\Sdk\wtgsdk\root\tutorial\tutorial.gif
D:\olitebk\Mobile\Sdk\wtgsdk\root\tutorial\ToDoList.jsp
D:\olitebk\Mobile\Sdk\wtgsdk\root\tutorial\WEB-INF\web.xml

6. Click **Compile JSP**. The Packaging Wizard compiles all your JSP files to Java Servlet classes. As Figure 17–5 displays, the following confirmation page appears. Click **OK**.

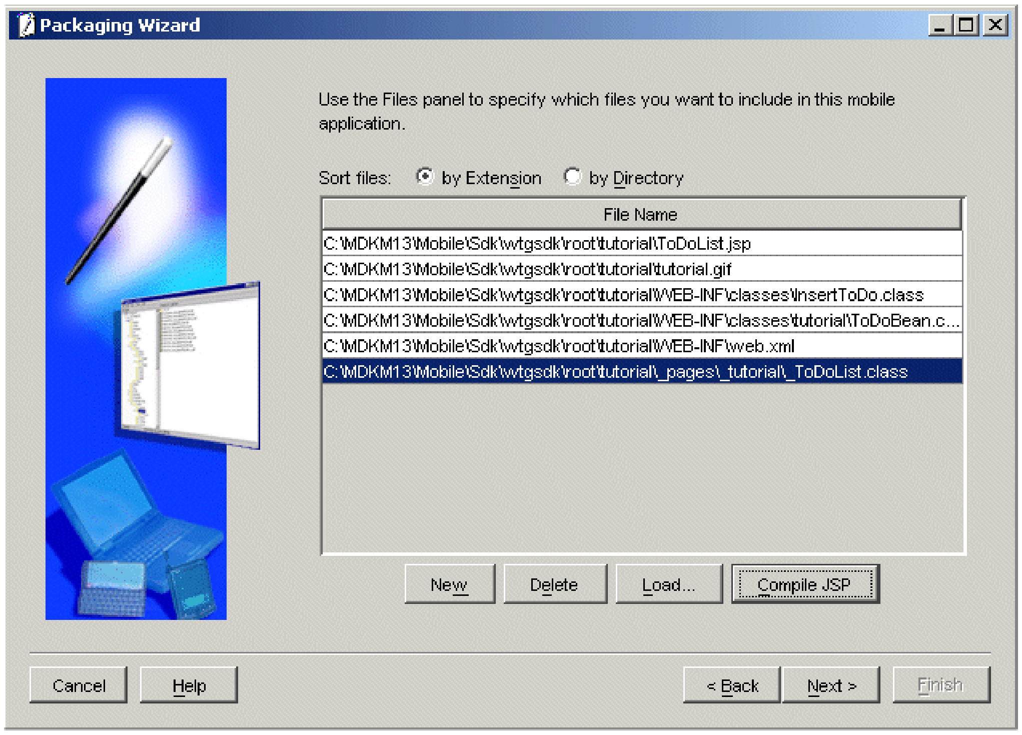
Figure 17–5   JSP Compilation Completion Message



```
C:\MDKM13\Mobile\Sdk\wtgsdk\root\tutorial\_pages
Created Directory _pages : true
These are the jspc arguments: -appRoot C:\MDKM13\Mobile\Sdk\wtgsdk\root\tutorial\ -d C:\
Translating file: C:\MDKM13\Mobile\Sdk\wtgsdk\root\tutorial\ToDoList.jsp
1 JSP file(s) translated successfully.
Compiling Java file: C:\MDKM13\Mobile\Sdk\wtgsdk\root\tutorial\_pages\_tutorial\_ToDoLis
=====================================================================
The JSP files are complied successfully, the generated class files will be included in t
```

7. The generated files are automatically added to the list of application files.

Figure 17–6   Including Generated Files to Application Files



Use the Files panel to specify which files you want to include in this mobile application.

Sort files: [✓ by Extension] by Directory

File Name
C:\MDKM13\Mobile\Sdk\wtgsdk\root\tutorial\ToDoList.jsp
C:\MDKM13\Mobile\Sdk\wtgsdk\root\tutorial\tutorial.gif
C:\MDKM13\Mobile\Sdk\wtgsdk\root\tutorial\WEB-INF\classes\InsertToDo.class
C:\MDKM13\Mobile\Sdk\wtgsdk\root\tutorial\WEB-INF\classes\tutorial\ToDoBean.c...
C:\MDKM13\Mobile\Sdk\wtgsdk\root\tutorial\WEB-INF\web.xml
C:\MDKM13\Mobile\Sdk\wtgsdk\root\tutorial_pages_tutorial_ToDoList.class

8. To view "To Do List" application servlets, click **Next**. To register with the Mobile Client Web Server, the Packaging Wizard automatically detects and selects servlets in your Local Application Directory. These servlets are registered with the Mobile Client Web Server.

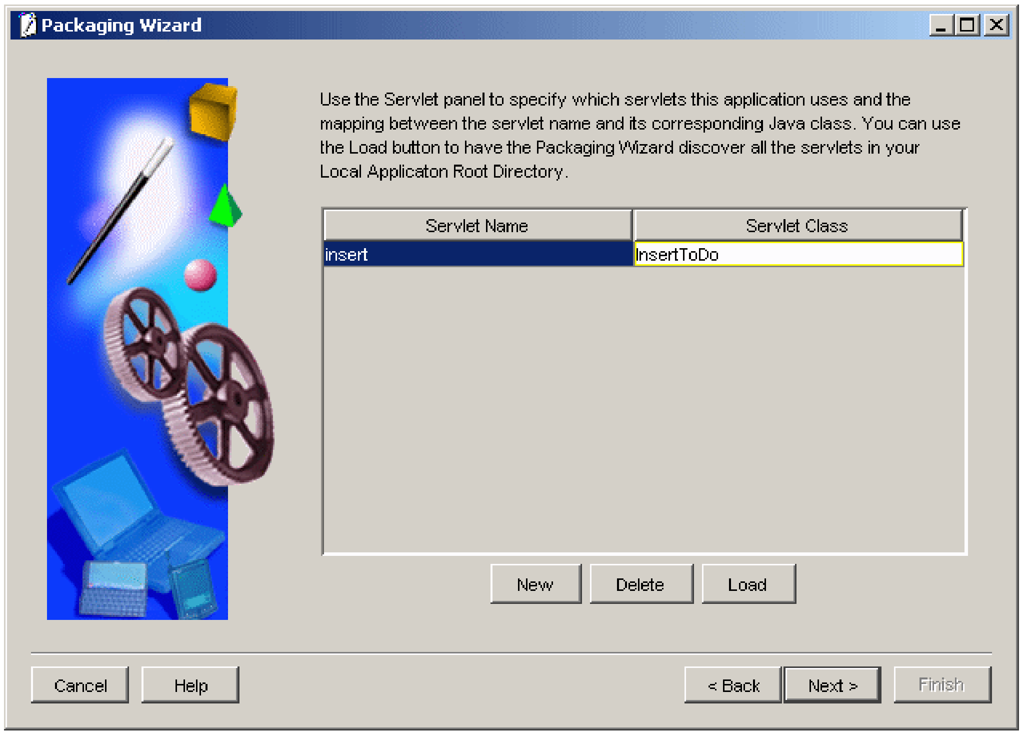
As Figure 17–7 displays, you can view the "To Do List" application servlet in the Servlets panel. Since the "To Do List" application contains only one servlet, the Servlets panel displays a single line.

The Servlets panel enables you to map virtual paths (servlet name) to the corresponding Java classes (servlet class).

Change the servlet name to **insert** by selecting the field, which turns white when selected. The servlet name is case sensitive, and must be in lower case. Leave the servlet class as `InsertTodo`.

Note: Ensure that you change the servlet name.

Figure 17–7 Registering Servlets

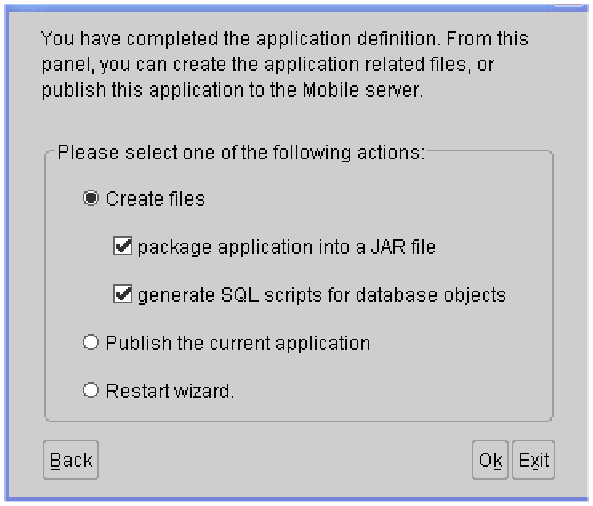


Use the Servlet panel to specify which servlets this application uses and the mapping between the servlet name and its corresponding Java class. You can use the Load button to have the Packaging Wizard discover all the servlets in your Local Applicaton Root Directory.

Servlet Name	Servlet Class
insert	InsertToDo

9. Click **Next** till you arrive at the Application Definition Completed Dialog as shown in Figure 17–8.

Figure 17–8 Application Definition Completed Dialog


You have completed the application definition. From this panel, you can create the application related files, or publish this application to the Mobile server.

Please select one of the following actions:

(●) Create files

☑ package application into a JAR file

☑ generate SQL scripts for database objects

( ) Publish the current application

( ) Restart wizard.

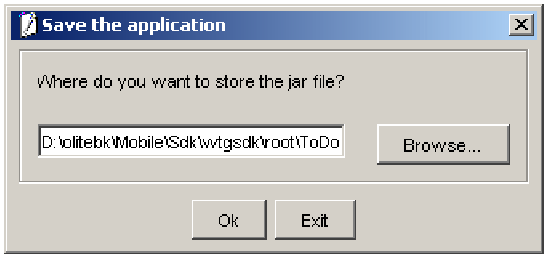
Using the Application Definition Completed panel, you can package the "To Do List" application into a JAR file. The Application Definition Completed Dialog remains open for you to initiate application packaging.

a. Select the **Create files** option and select both the **Package Application into a JAR file** and **Generate SQL scripts for database objects** boxes.

b. At this stage, the Save the Application dialog prompts you for the name of the JAR file, as Figure 17–9 displays. The default location is given below.

`<`*`ORACLE_HOME`*`>\Mobile\Sdk\wtgsdk\root\ToDoList.jar`

Figure 17–9 Save the Application Dialog


Save the application

Where do you want to store the jar file?

D:\olitebk\Mobile\Sdk\wtgsdk\root\ToDo

After choosing the JAR file, click **OK**. The JAR file is created and contains the application files and definition.

c. Back in the Application Definition Completed dialog, select the **Publish the Current Application** option and click **OK**.

The Publish the Application dialog appears. As Table 17–4 describes, enter the specified values.

Note: The Mobile Server must be up for successful publishing.

Table 17–4 Publish the Application Dialog Description

Field	Description	Value
Mobile Server URL	URL or IP Address of the machine where the Mobile Server is running.	`<Mobile Server>/webtogo`
Mobile Server User Name	User name of the Mobile Server user with administrative privileges.	Administrator
Mobile Server Password	Password of the Mobile Server user with administrative privileges.	admin
Repository Directory	Directory name where all files for this application will be stored inside the Mobile Server Repository.	`/tutorial`
Public Application	Do not select this check box unless you want to make this application available to all users.	Clear

d. To publish the application in the Mobile Server Repository, click **OK**. A dialog displays the application publishing status. You must wait until the application is published.

e. To confirm that the application is published successfully, click **OK**.

f. To exit the Packaging Wizard, click **Exit**.

You have now completed all development tasks that are required for packaging your application. Your application is packaged.

17.4 Administer the Application

This section describes how to administer the application that you created and deployed through the following tasks.

- Section 17.4.1, "Start the Mobile Server and the Mobile Manager"
- Section 17.4.2, "Using the Mobile Manager to Create a New User"
- Section 17.4.3, "Setting Application Properties"
- Section 17.4.4, "Granting User Access to the Application"
- Section 17.4.5, "Defining Snapshot Template Values for the User"

For more information about Mobile Manager tasks described in this tutorial, see the *Oracle Database Lite Administration and Deployment Guide*.

17.4.1 Start the Mobile Server and the Mobile Manager

The Mobile Manager is a Web-based application that enables you to administer Mobile Server applications. To start the Mobile Manager, perform the following steps.

1. Using the command prompt, go to the following directory.

   ```
   <ORACLE_HOME>\Mobile\Server\bin
   ```

2. To start the Mobile Server for the first time and subsequent occasions, execute the `runmobileserver` command.

3. Start your Web browser and connect to the Mobile Server by enter the following URL:

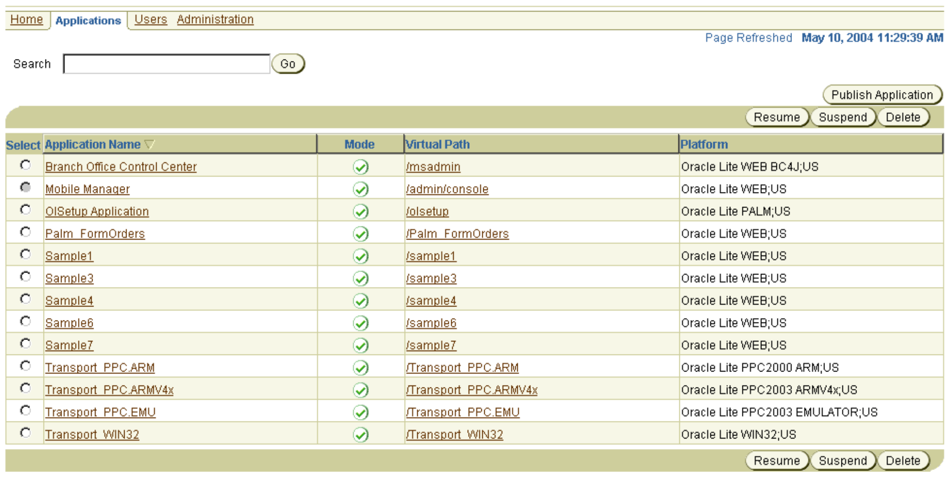
   ```
   http://<mobile_server>/webtogo
   ```

Note: Replace `<mobile_server>` with the host name of your Mobile Server.

---

4. Log on as the Mobile Server Administrator using `administrator` as the User Name and `admin` as the Password.
5. To launch the Mobile Manager, click the **Mobile Manager** link in the workspace.
6. Click the Mobile Server link.
7. Click the **Applications** link. As Figure 17–10 displays, the Applications page appears. Locate the To Do List application, which should be there since you published it.

Figure 17–10 Applications Page



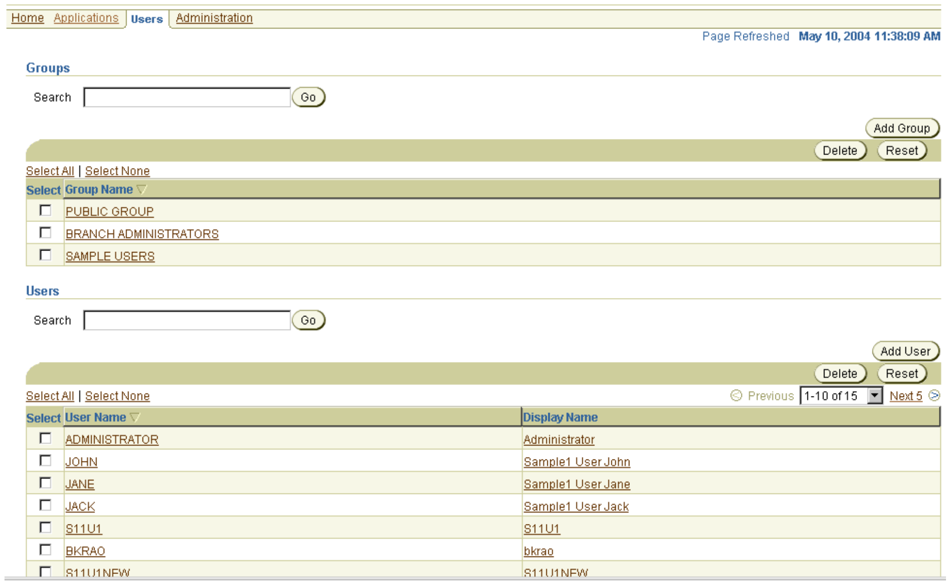
Select	Application Name	Mode	Virtual Path	Platform
○	Branch Office Control Center	✓	/msadmin	Oracle Lite WEB BC4J;US
○	Mobile Manager	✓	/admin/console	Oracle Lite WEB;US
○	OlSetup Application	✓	/olsetup	Oracle Lite PALM;US
○	Palm_FormOrders	✓	/Palm_FormOrders	Oracle Lite WEB;US
○	Sample1	✓	/sample1	Oracle Lite WEB;US
○	Sample3	✓	/sample3	Oracle Lite WEB;US
○	Sample4	✓	/sample4	Oracle Lite WEB;US
○	Sample6	✓	/sample6	Oracle Lite WEB;US
○	Sample7	✓	/sample7	Oracle Lite WEB;US
○	Transport_PPC.ARM	✓	/Transport_PPC.ARM	Oracle Lite PPC2000 ARM;US
○	Transport_PPC.ARMV4x	✓	/Transport_PPC.ARMV4x	Oracle Lite PPC2003 ARMV4x;US
○	Transport_PPC.EMU	✓	/Transport_PPC.EMU	Oracle Lite PPC2003 EMULATOR;US
○	Transport_WIN32	✓	/Transport_WIN32	Oracle Lite WIN32;US

17.4.2 Using the Mobile Manager to Create a New User

To create a new Mobile Server user, perform the following steps.

1. On the Mobile Manager home page, click the **Users** link. As Figure 17–11 displays, the Users page appears.

Figure 17–11 Users Page



Page Refreshed May 10, 2004 11:38:09 AM

Groups

Search

Select	Group Name
☐	PUBLIC GROUP
☐	BRANCH ADMINISTRATORS
☐	SAMPLE USERS

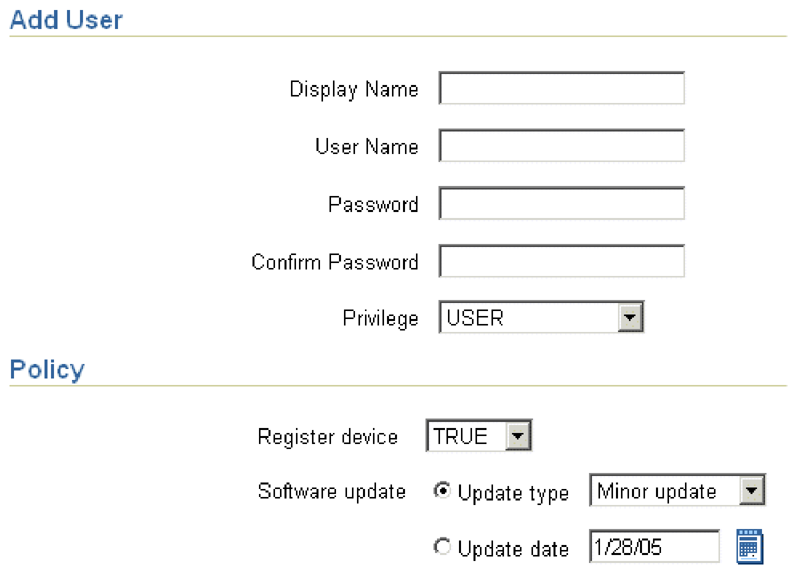
Users

Search

Select	User Name	Display Name
☐	ADMINISTRATOR	Administrator
☐	JOHN	Sample1 User John
☐	JANE	Sample1 User Jane
☐	JACK	Sample1 User Jack
☐	S11U1	S11U1
☐	BKRAO	bkrao
☐	S11U1NEW	S11U1NEW

2. Click **Add User**. As Figure 17–12 displays, the Add User page appears.

Figure 17–12 Add User Page



Add User

Display Name	
User Name	
Password	
Confirm Password	
Privilege	USER

Policy

Register device	TRUE	
Software update	● Update type	Minor update
	○ Update date	1/28/05

3. As described in Table 17–5, enter the following information in the Add User page and click **Save**.

Table 17–5 Add User Page Description

Field	Value
Display Name	`tutorial`
User Name	`tutorial`
Password	`tutorial`
Password Confirm	`tutorial`

Table 17–5 (Cont.) Add User Page Description

Field	Value
Privilege	User
Register Device	True
Software Update	Select all updates

17.4.3 Setting Application Properties

To set the "To Do List" application properties, perform the following steps.

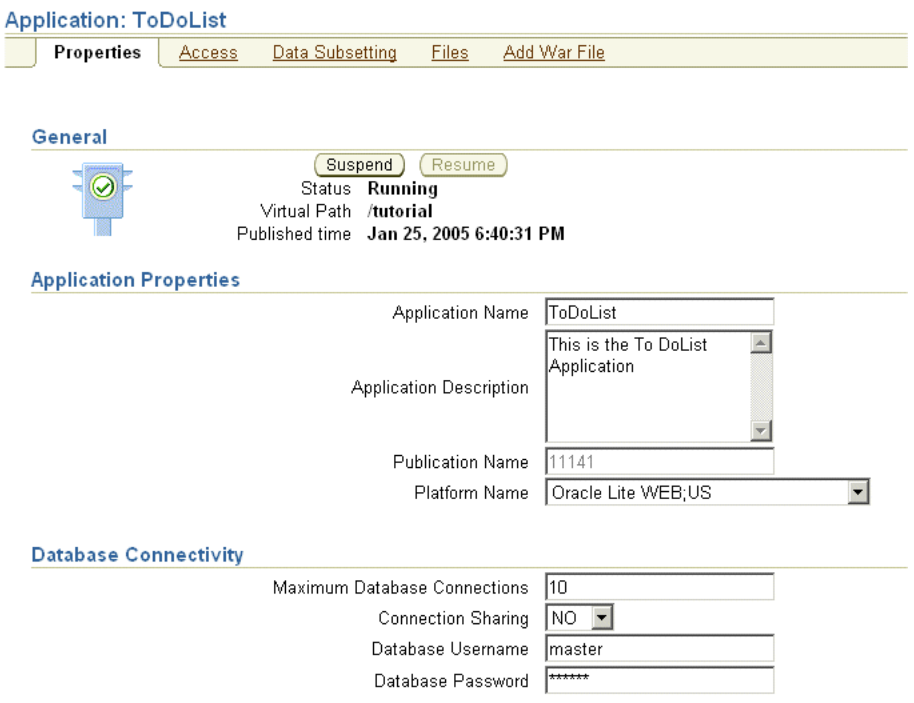
1. On Mobile Manager home page, click the **Applications** link. The Applications page appears.
2. To search for the application that you just published, enter To Do List in the **Application Name** field and click **Search**. The "To Do List" application appears in the workspace.

   Note: To display all the available applications, leave the search field blank and click **Search**. This action generates a list of all the available Mobile Server applications in the workspace.

   ---

3. Click the **To Do List** application link. As Figure 17–13 displays, the Application Properties page lists application properties and database connectivity details.

Figure 17–13 Application Properties Page



Application: ToDoList

Status	Running
Virtual Path	/tutorial
Published time	Jan 25, 2005 6:40:31 PM
Application Name	ToDoList
Application Description	This is the To DoList Application
Publication Name	11141
Platform Name	Oracle Lite WEB;US
Maximum Database Connections	10
Connection Sharing	NO
Database Username	master
Database Password	******

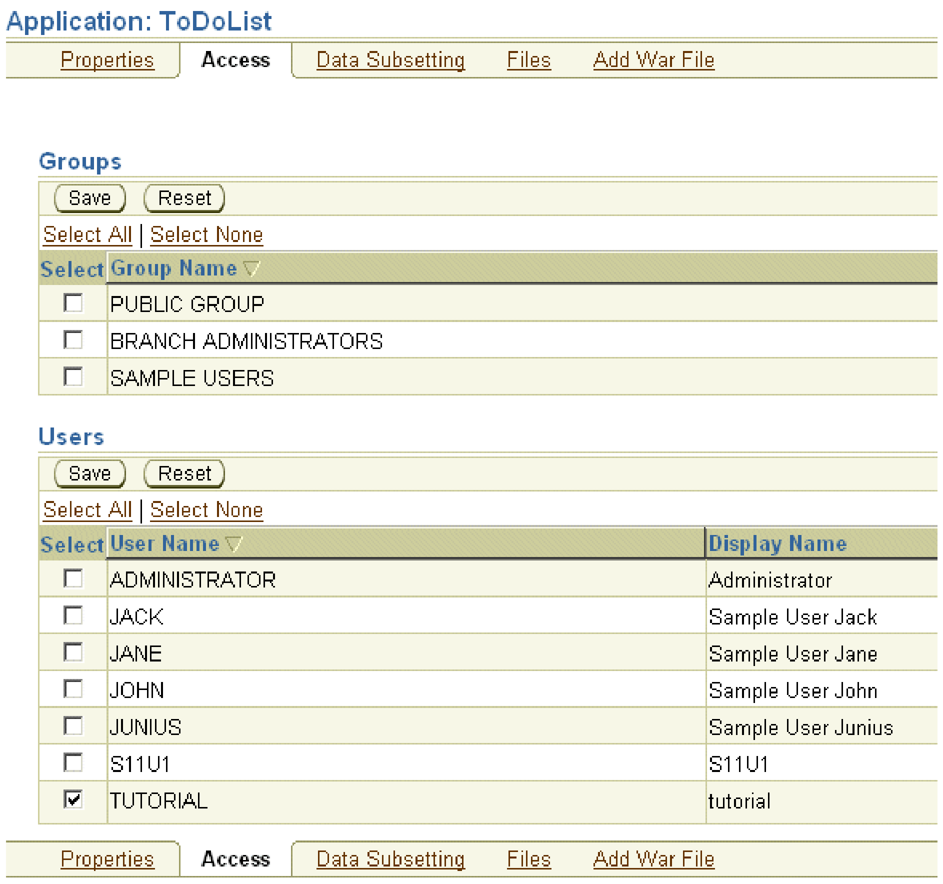
4. In the **Database Password** field type `master`. This is the default password for the Web-to-Go demo schema. Click **Apply**. The Mobile Manager displays a confirmation message.

17.4.4 Granting User Access to the Application

To grant the user TUTORIAL access to the "To Do List" application, perform the following steps.

1. Navigate to the Application Properties page and click the **Access** link. As Figure 17–14 displays, the Access page lists groups and users that are associated with the application. The check boxes on this page indicate whether or not the user or group has access to the application.

Figure 17–14 Access Page



Select	Group Name
☐	PUBLIC GROUP
☐	BRANCH ADMINISTRATORS
☐	SAMPLE USERS

Select	User Name	Display Name
☐	ADMINISTRATOR	Administrator
☐	JACK	Sample User Jack
☐	JANE	Sample User Jane
☐	JOHN	Sample User John
☐	JUNIUS	Sample User Junius
☐	S11U1	S11U1
☑	TUTORIAL	tutorial

2. Under the Users table, locate the user TUTORIAL and select the check box displayed against the user, TUTORIAL.

3. Click **Save**. The Mobile Manager displays a confirmation message. The user TUTORIAL has now been granted access to the "To Do List" application.

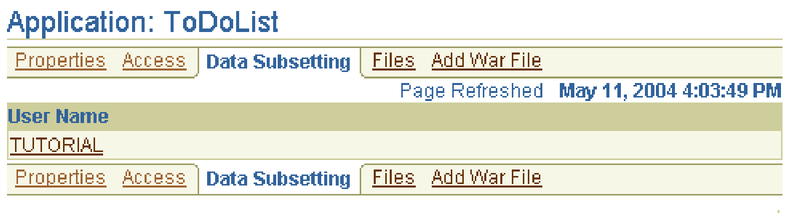
17.4.5 Defining Snapshot Template Values for the User

Define the snapshot template variable for the user, TUTORIAL. Each Mobile Client for Web-to-Go downloads the same application data when it synchronizes. In some cases, you may want to specify the data your application downloads for each user. You can accomplish this by modifying the user's snapshot template variable.

To modify a user's Data Subsetting parameters, perform the following steps.

1. Navigate to the Applications page and click the **ToDoList** application link. The Application Properties page appears. Click the **Data Subsetting** link. As Figure 17–15 displays, the Data Subsetting page appears.

Figure 17–15 Data Subsetting Page



Application: ToDoList

Properties Access **Data Subsetting** Files Add War File

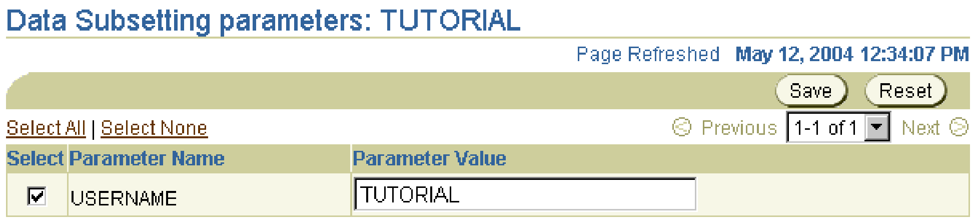
Page Refreshed **May 11, 2004 4:03:49 PM**

User Name
TUTORIAL

Properties Access **Data Subsetting** Files Add War File

2. Under the User Name column, click the user name link `tutorial`. As Figure 17–16 displays, the Data Subsetting Parameters page appears.

Figure 17–16 Data Subsetting Parameters Page



Data Subsetting parameters: TUTORIAL

Page Refreshed **May 12, 2004 12:34:07 PM**

Select	Parameter Name	Parameter Value
✓	USERNAME	TUTORIAL

3. Select the Username parameter and enter the value `tutorial`. Click **Save**.

For more information about snapshots, refer the *Oracle Database Lite Administration and Deployment Guide*.

17.5 Execute the Application on the Mobile Client for Web-to-Go

This section describes how to set up a Mobile client to use the application that you created and tested in the Development section, deployed in the Deployment section, and then administered in the Administration section. In this section, you will perform the following tasks.

- Section 17.5.1, "Install the Mobile Client for Web-to-Go"
- Section 17.5.2, "Log into the Mobile Client for Web-to-Go"
- Section 17.5.3, "Manually Synchronize the Mobile Client for Web-to-Go"

Note: You must install the application and test it on a separate machine from the Mobile Server.

17.5.1 Install the Mobile Client for Web-to-Go

You must install the Mobile client before you can use the application that you created and deployed.

Note: You must install the Mobile Client on a machine which does not host the Mobile Server installation.

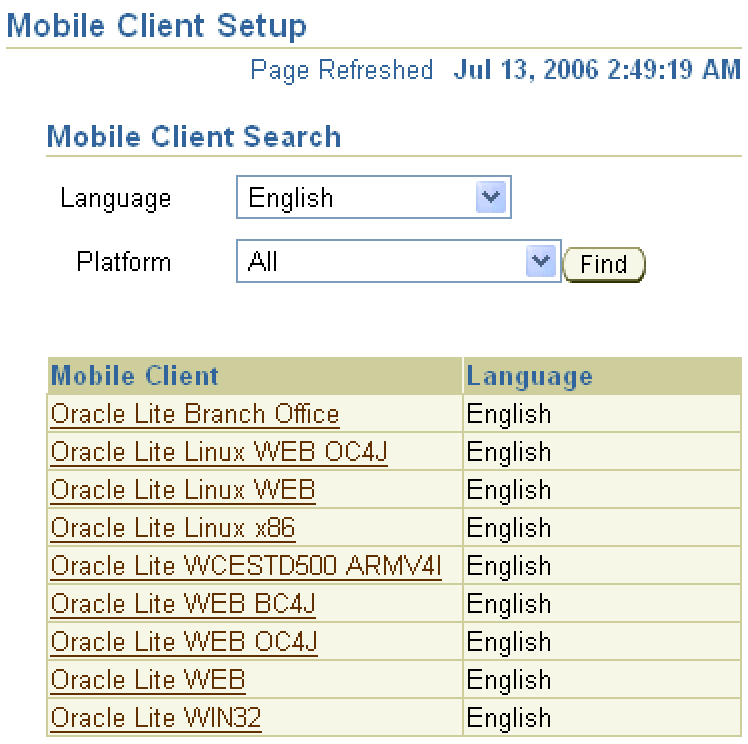
To install the Mobile Client for Web-to-Go, perform the following actions.

1. Start your Web browser and connect to the Mobile Server by entering the following URL.

   ```
   http://<mobile_server>/webtogo/setup
   ```

2. As Figure 17–17 displays, the Mobile Client Setup page lists a set of Mobile clients by platform. To download the Mobile Client for Web-to-Go setup program, click the corresponding Mobile Client link.

Figure 17–17 Mobile Client Setup Page



Mobile Client Setup

Page Refreshed Jul 13, 2006 2:49:19 AM

Mobile Client Search

Language: English

Platform: All

Mobile Client	Language
Oracle Lite Branch Office	English
Oracle Lite Linux WEB OC4J	English
Oracle Lite Linux WEB	English
Oracle Lite Linux x86	English
Oracle Lite WCESTD500 ARMV4I	English
Oracle Lite WEB BC4J	English
Oracle Lite WEB OC4J	English
Oracle Lite WEB	English
Oracle Lite WIN32	English

Note: While installing the Mobile Client, you will be prompted for the User name and Password. Enter `tutorial` as the user name and `tutorial` as the password.

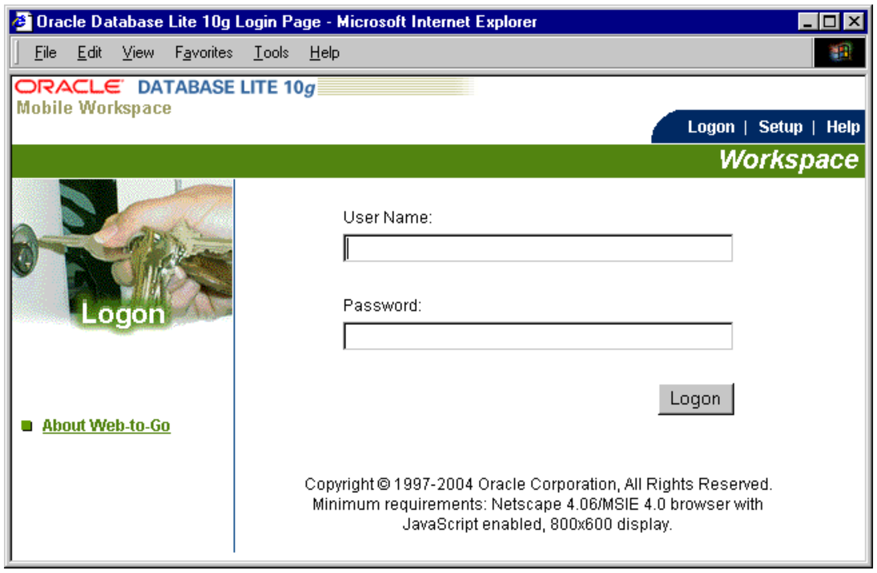
3. If you are using Netscape, choose a location to save the setup program and click **OK**. In Windows Explorer, double-click `setup.exe` to run the setup program.

   If you are using Internet Explorer, run the setup program from your browser window.

4. While installing the Mobile Client, you will be prompted for the user name and password. Enter `tutorial` as the user name and `tutorial` as the password.

5. The setup program prompts you to choose an installation directory such as `D:\mobileclient` and downloads all the required components and starts the Mobile Client for Web-to-Go on your machine. After completing the installation, the Mobile Manager login page appears as Figure 17–18 displays.

Figure 17–18 Mobile Manager Login Page



Copyright © 1997-2004 Oracle Corporation, All Rights Reserved.
Minimum requirements: Netscape 4.06/MSIE 4.0 browser with JavaScript enabled, 800x600 display.

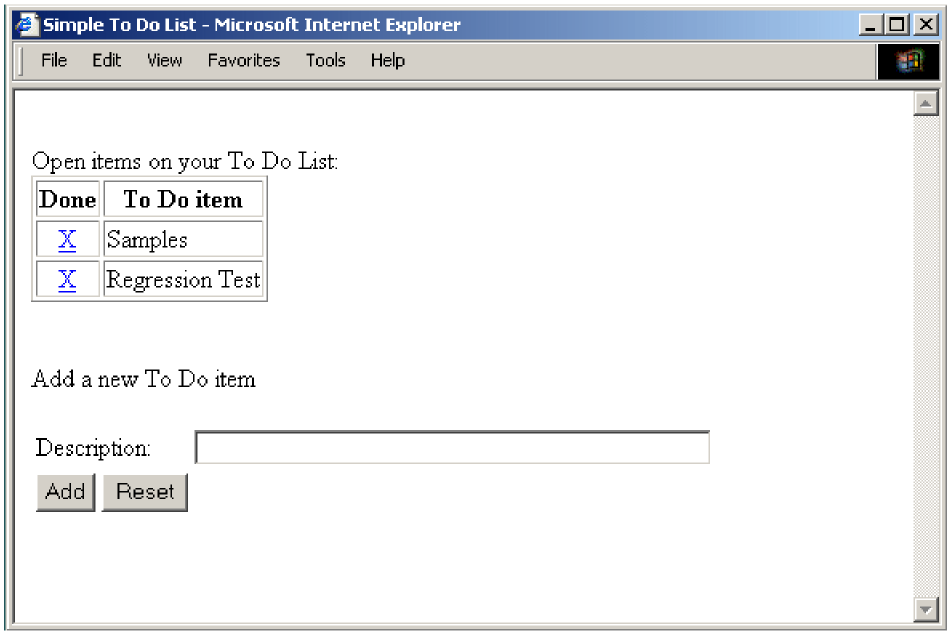
17.5.2 Log into the Mobile Client for Web-to-Go

Complete the Mobile Client for Web-to-Go setup process. Your browser displays the Web-to-Go logon page. If your browser does not display the Web-to-Go login page, enter the following URL.

```
http://localhost/webtogo
```

1. Log on to Web-to-Go using `tutorial` as the User Name and `tutorial` as the password.
2. As you are logging into the Mobile Client for Web-to-Go for the first time, you must complete the initial setup process. The client initialization page appears and displays a confirmation message. "The Web-to-Go Client was installed successfully! Web-to-Go client will now synchronize your computer with the Mobile Server."
3. To start downloading your applications and data, click **Next**. The data synchronization page appears. This page displays the data synchronization status.
4. Once the synchronization process is finished, the Mobile Client for Web-to-Go is restarted automatically. The Mobile Server displays the following message: "New or updated application files have been downloaded. Please wait while Mobile Client for Web-to-Go is being restarted."
5. After restarting the Mobile Client for Web-to-Go, the workspace portal appears with a single icon for the "To Do List" application and a link labeled **ToDoList**.
6. Click the **To Do List** application icon. As Figure 17–19 displays, Web-to-Go launches the "To Do List" application in your browser.

Figure 17–19 The "To Do List" Application


Simple To Do List - Microsoft Internet Explorer
File Edit View Favorites Tools Help

Open items on your To Do List:

Done	To Do item
X	Samples
X	Regression Test

Add a new To Do item

Description:

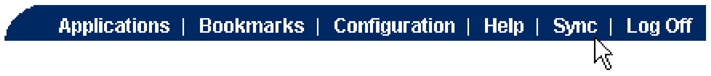
7. Enter a new To Do item and save it in the database. Click **Add**.
8. Exit the application by closing the browser window. This action returns you to the workspace.

17.5.3 Manually Synchronize the Mobile Client for Web-to-Go

If you set up automatic synchronization, you can skip this section. To manually synchronize the Mobile Client for Web-to-Go with the Mobile Server, perform the following steps.

1. As Figure 17–20 displays, click the Sync tab in the upper right corner of the workspace.

Figure 17–20 Sync Tab Location


Applications | Bookmarks | Configuration | Help | Sync | Log Off

The Mobile Client for Web-to-Go synchronizes the application and all of your data to the Oracle 10*g* Database. The workspace appears when the synchronization process has completed.

18

Tutorial for Building Mobile Web Applications Using ADF/BC4J

The following sections use a tutorial to describe how to create, deploy, and use an ADF/BC4J application:

18.1 Overview

The Oracle Application Development Framework (Oracle ADF) is an end-to-end a application framework that builds on J2EE standards and open-source technologies to simplify and accelerate implementing service-oriented applications. If you develop enterprise solutions that search, display, create, modify and validate data using Web, wireless, desktop or Web services interfaces, then Oracle ADF can simplify your job.

Oracle Business Components for Java (BC4J) is a part of the Oracle JDeveloper IDE (Integrated Development Environment), and provides Java developers with tools to create and manage reusable Java components.

ADF/BC4J offers a standards-based, server-side Java and XML framework for developers. You can build and deploy reusable business components for high performance Internet applications, such as e-commerce and business-to-business systems. Applications, which are created using ADF/BC4J, comprise five basic framework components: Entity Objects, Associations, View Objects, View Links, and Application Modules. Each of these components is interrelated to the other components, which enables you to establish views into database tables. You can combine, filter, and sort data as needed.

When used in application development, ADF/BC4J automatically generates database oriented components, so that you can focus on the business logic instead of on database related components.

The ADF/BC4J sample application used in this tutorial maintains employee details and stores all items in a relational database.

18.1.1 Before You Start

Ensure that the computer you are using for your development meets the requirements specified in this section. Table 18–1 lists configuration and installation requirements for the development computer.

Table 18–1 Development Computer Requirements

Requirement	Description
Windows NT/2000/XP User Login	The Windows NT/2000/XP login user must have Administrator privileges on the development computer.
Installed Java Components	■ Java Development Kit 1.4.2 or higher for the Mobile Server and Jdeveloper. ■ JRE 1.5.x or higher for the OC4J client.
Installed Oracle Components	Mobile Server or Mobile Development Kit (Oracle Database Lite CD-ROM) Oracle 9*i* or higher with the default Master schema installed. Oracle10*g* JDeveloper Release 3 (10.1.3.0.4) Studio. Studio Edition Version 10.1.3.0.4.3673 BUILD JDEVADF_10.1.3_NT_060125.0900.3673 **Note**: This tutorial is written and certified using the above-mentioned version of Jdeveloper.

18.2 Creating a Database Connection

When using ADF/BC4J, you need to define the database connection in both JDeveloper and Oracle Database Lite, which is shown in the following sections:

- Section 18.2.1, "Creating a Database Connection to Oracle Database"
- Section 18.2.2, "Specify The Connection To The Oracle Lite Database"

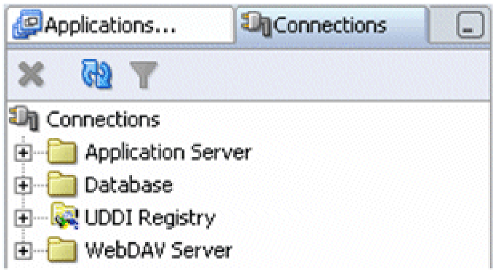
18.2.1 Creating a Database Connection to Oracle Database

Java Database Connectivity (JDBC) is a standard application-programming interface (API) that is used for connecting a Java application to relational databases. JDeveloper uses a connection navigator to maintain connection information for your application. The connection navigator makes it easy to create, manage, and test database connections. If you have not already established a connection to the database, then perform the following steps:

1. Connect to Oracle Database as the `master` user and execute the `adf_main.sql` script, which is located in the `<`*`ORACLE_LITE_HOME`*`>\Mobile\Sdk\wtgsdk\src\bc4jtutorial` directory.
2. Start Oracle10*g* JDeveloper Release 3 (10.1.3) Studio.
3. Select the **Connections** tab on the Applications Navigator.

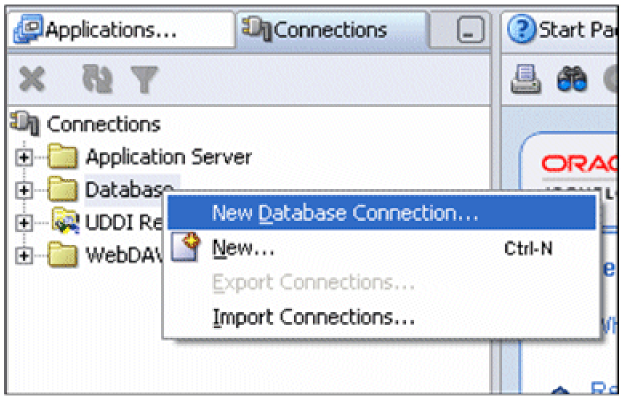
Note: If the Connections tab is not showing, choose **View -> Connection Navigator** from the JDeveloper main menu.

Figure 18–1 JDeveloper Connection tab on the Connection Navigator



4. Right-click the `Database` folder and select **New Database Connection**. This starts the Create Database Connection Wizard.

Figure 18–2 JDeveloper Connection tab on the Connection Navigator, New Database Connection

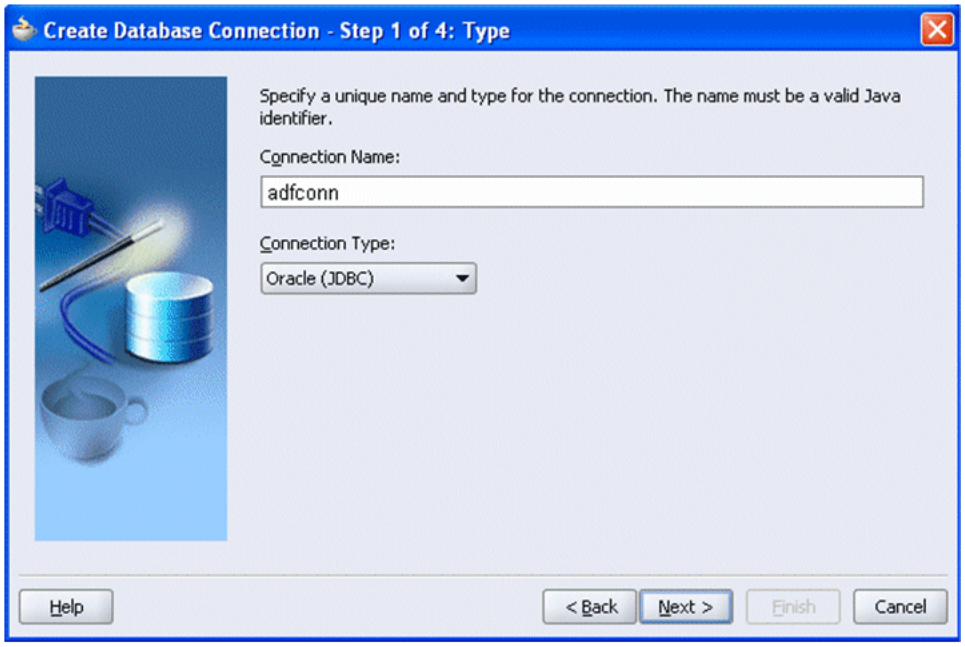


5. Perform the following in the Create Database Connection Wizard:

   a. Review the information on the Welcome page and click **Next**.

   b. In the Connection Name field, enter `adfconn`. Click **Next**.

Figure 18–3 Connection Wizard - Step 1 of 4: Type Panel



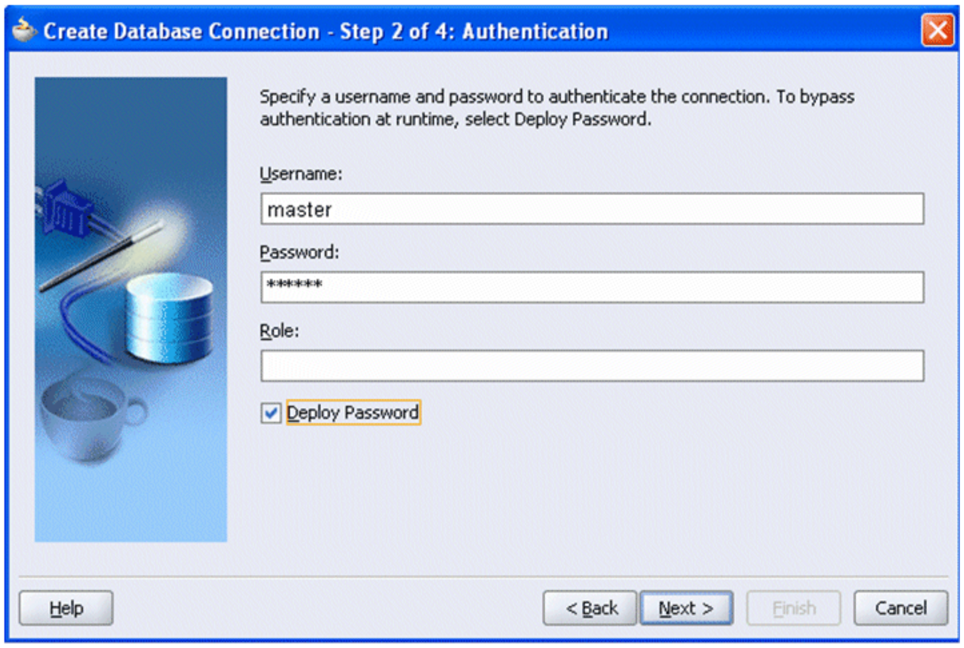
Specify a unique name and type for the connection. The name must be a valid Java identifier.

Connection Name: adfconn

Connection Type: Oracle (JDBC)

c. On the Authentication page, enter `master/master` for the username/password fields. Select **Deploy password**.

Figure 18–4 Connection Wizard - Step 2 of 4: Authentication Panel



Specify a username and password to authenticate the connection. To bypass authentication at runtime, select Deploy Password.

Username: master

Password: ******

Role:

[✓] Deploy Password

d. On the Connections page, the default values for the connection is as follows:

– Driver: thin

– Host name: `<mobileserver_host>`

– JDBC Port: `<mobileserver_port>`

- SID: `<repository_SID>`

e. Click **Next** and Test Connection. One of the following occurs:
   - If the database is available and the connection details are correct, then `Success!` displayed in the Status window.
   - If an error occurs, verify the connection settings. Click **Back** to make any necessary changes, and then retest the connection.
   - If the connection is successful, click **Finish** to complete the connection.

Note: Leave the fields set to these default values.

18.2.2 Specify The Connection To The Oracle Lite Database

The Oracle Database Lite connection is used for synchronization between the two databases—the back-end Oracle database and the local Oracle Lite database. Once you specify the connection within JDeveloper, then modify the application to use this connection.

For this example, we will create the `WTGJdbc` connection, which uses the `oracle.lite.web.WTGJdbcDriver`.

Note: The `WTGJdbc` connection must be used within the application as well as configured in the project settings. However, during development, you may have used the `adfconn` JDBC connection for testing. Once development is complete for the application, make sure that you modify your application to use the WTGJdbc connection before you deploy it.

To create the `WTGJdbc` connection, do the following:

1. In JDeveloper, configure the project settings and include the Oracle Database Lite user library named `webtogo.jar`, as follows:

   a. Copy the `olite40.jar` and `webtogo.jar` files from the Oracle Lite MDK into the *<JDEV_HOME>*`\bc4j\lib`.

   b. Click **Tools->Manage Libraries**.

   c. Select the **Libraries** tab.

   d. Select **User**.

   e. Create the library dialogs for both the `webtogo.jar` and `olite40.jar` files, as follows:
      - Click **New Button**. The "Create Library Dialog" displays.
      - Enter `webtogo.jar` for the library name.
      - Select **Deployed by default**.
      - Click **Add Entry** and browse for the `webtogo.jar` file.
      - Click **OK**.
      - The Manage Libraries screen displays. Click **OK**.

   f. Repeat step `e` for the `olite40.jar` file.

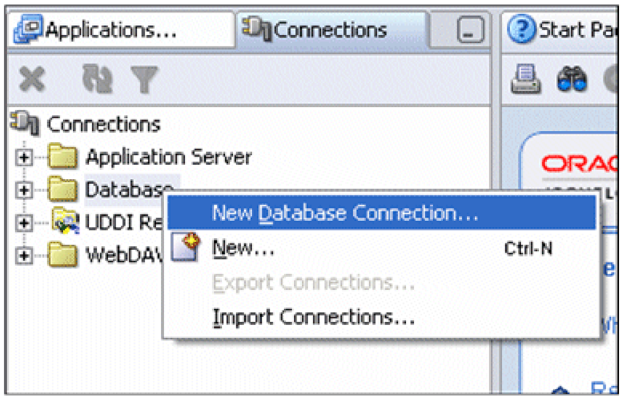
2. Select the Connections tab on the Application Navigator.

Note: If you do not see the Connections tab, select **View->Connection Navigator**.

---

3. Right-click the Database folder and select **New Database Connection**.

Figure 18–5 JDeveloper Connection tab on the Connection Navigator, New Database Connection



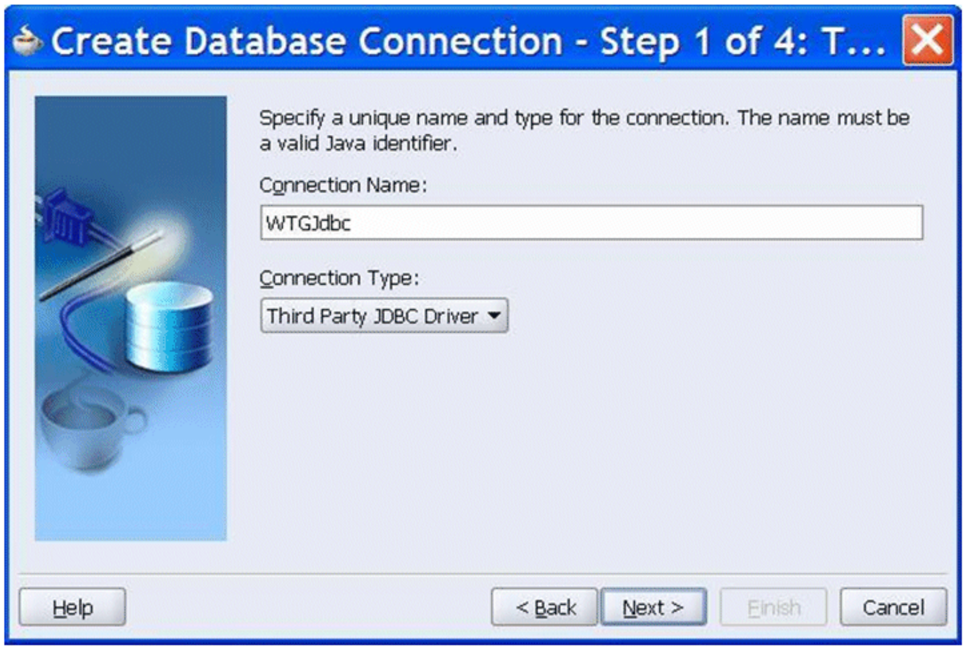
The Create Database Connection Wizard starts. Perform the following in creating a new database connection using this wizard:

a. Click **Next** on the Welcome screen.

b. Enter `WTGJdbc` as the Connection Name and choose **Third Party JDBC Driver** as the JDBC Connection Type.

c. Click **Next**.

Figure 18–6 Connection Wizard - Step 1 of 4: Type Panel


Create Database Connection - Step 1 of 4: T...

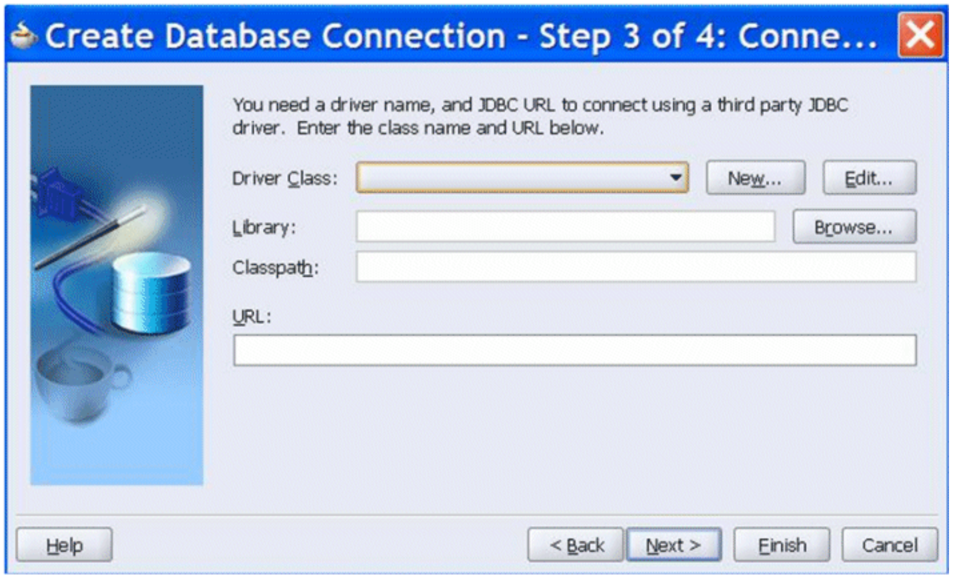
Specify a unique name and type for the connection. The name must be a valid Java identifier.

Connection Name: WTGJdbc

Connection Type: Third Party JDBC Driver

4. The Connection Wizard - Step 2 of 4: Authentication panel appears. Do not enter any values in this panel. Click **Next**.
5. The Connection Wizard - Step 3 of 4: Connection panel appears. Click **New**.

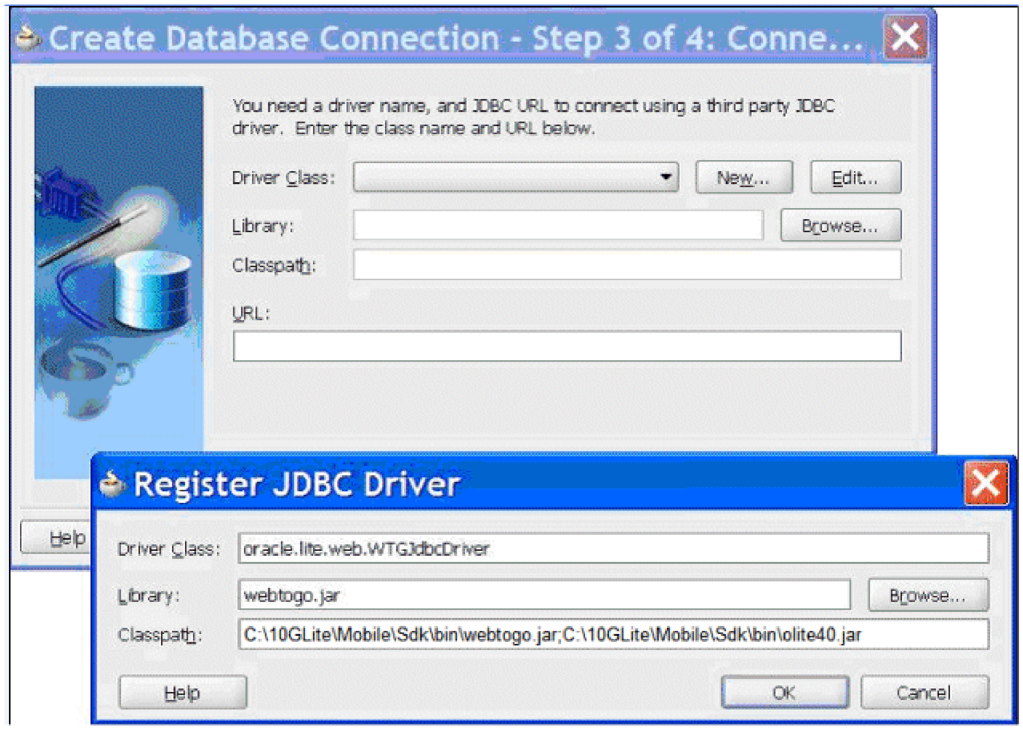
Figure 18–7 Connection Wizard - Step 3 of 4: Connection Panel


Create Database Connection - Step 3 of 4: Conne...
You need a driver name, and JDBC URL to connect using a third party JDBC driver. Enter the class name and URL below.
Driver Class:
New...
Edit...
Library:
Browse...
Classpath:
URL:
Help
< Back
Next >
Finish
Cancel

6. The Register JDBC Driver dialog appears, as Figure 18–8 displays. Perform the following:

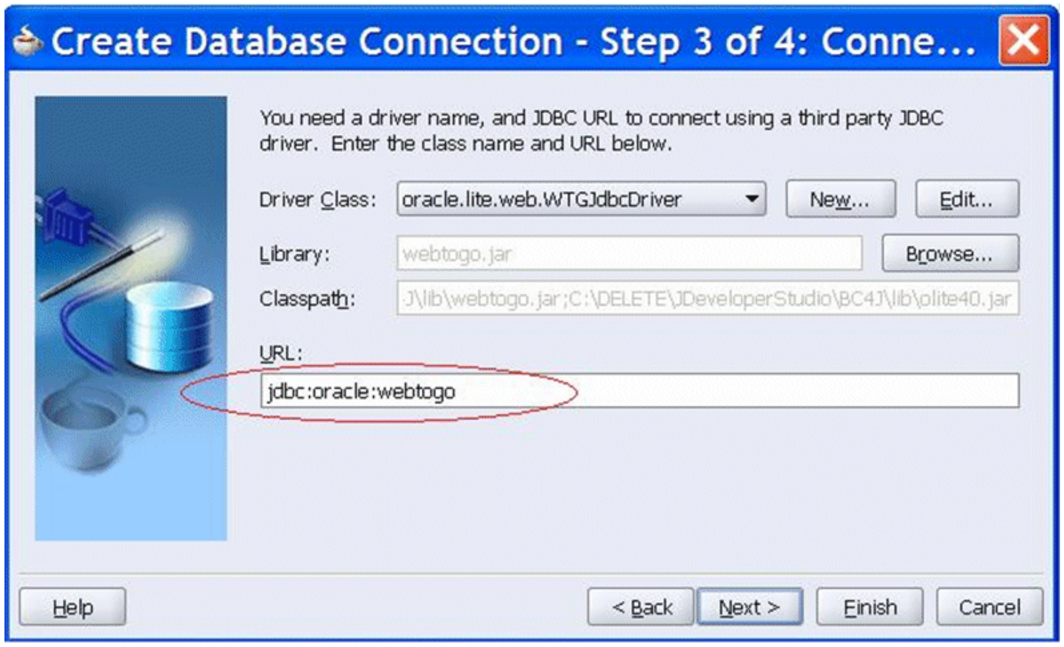
   a. Enter `oracle.lite.web.WTGJdbcDriver` as the Driver Class. Choose `webtogo` from the Library list and click **OK**.

Figure 18–8 JDBC Driver Dialog


Create Database Connection - Step 3 of 4: Conne...
You need a driver name, and JDBC URL to connect using a third party JDBC driver. Enter the class name and URL below.
Driver Class:
New...
Edit...
Library:
Browse...
Classpath:
URL:
Help
Register JDBC Driver
Driver Class: oracle.lite.web.WTGJdbcDriver
Library: webtogo.jar
Browse...
Classpath: C:\10GLite\Mobile\Sdk\bin\webtogo.jar;C:\10GLite\Mobile\Sdk\bin\olite40.jar
Help
OK
Cancel

b. Enter the following `jdbc:oracle:webtogo` URL and click **Next**.

Figure 18–9 Enter URL for Database Connection


Create Database Connection - Step 3 of 4: Conne...
You need a driver name, and JDBC URL to connect using a third party JDBC driver. Enter the class name and URL below.
Driver Class: oracle.lite.web.WTGJdbcDriver
New...
Edit...
Library: webtogo.jar
Browse...
Classpath: J\lib\webtogo.jar;C:\DELETE\JDeveloperStudio\BC4J\lib\olite40.jar
URL:
jdbc:oracle:webtogo
Help
< Back
Next >
Finish
Cancel

7. The Connection Wizard - Step 4 of 4: Click **Finish**. Do not test the Connection since you do not have any Client database to test the connection at this point.

18.3 Develop the ADF/BC4J Application

The following sections describe the steps to develop the ADF/BC4J application for Oracle Database Lite:

- Section 18.3.1, "Building the Data Model with ADF Business Components"
- Section 18.3.2, "Customize the Business Components Views"
- Section 18.3.3, "Creating a Master-Detail JavaServer Faces Page"
- Section 18.3.4, "Running the JSF Page"
- Section 18.3.5, "Configure the ADF/BC4J Application for the Oracle Database Lite Environment"

18.3.1 Building the Data Model with ADF Business Components

The data model provides data access and validation for an application. The data is validated by the model, regardless of the client implementation. This separates the validation and business rules from the user interface.

The following sections describe the steps to create an application in JDeveloper and create a Business Components model for your applications.

- Section 18.3.1.1, "Create a New Application and Projects"
- Section 18.3.1.2, "Create Business Components"

18.3.1.1 Create a New Application and Projects

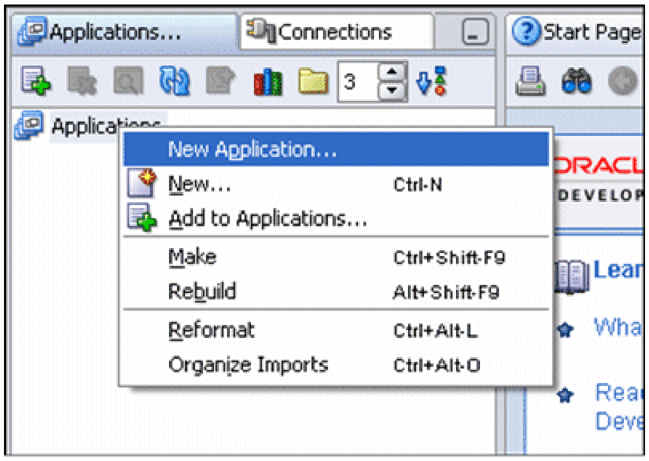
In JDeveloper, you work with projects contained in an application. The application is the highest point in the control structure.

A JDeveloper project is an organizational structure that logically groups related files. You can add multiple projects to your application to easily organize, access, modify, and reuse your source code. In the Applications Navigator, projects are displayed as the second level in the hierarchy, under the application.

Before you create any components, you must first create the application and a project. Perform the following steps:

1. Select the Applications tab to go back to the Applications Navigator.
2. Right-click the Applications node and select **New Application** from the context menu.

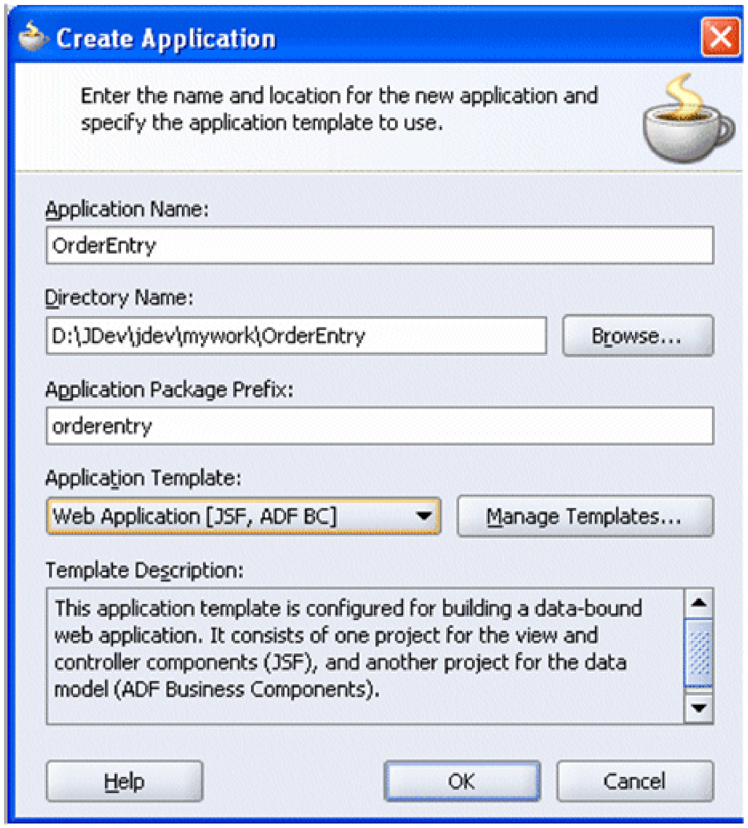
Figure 18–10 New Application



3. In the Create Application dialog box, enter the Application Name `OrderEntry`. Notice that the directory name changes automatically.

   Enter `orderentry` as the Application Package Prefix. For the Application Template, select the `Web Application [JSF, ADF BC]` value from the Application Template drop-down list.

Figure 18–11 Create Application



Enter the name and location for the new application and specify the application template to use.

Application Name: OrderEntry

Directory Name: D:\JDev\jdev\mywork\OrderEntry

Application Package Prefix: orderentry

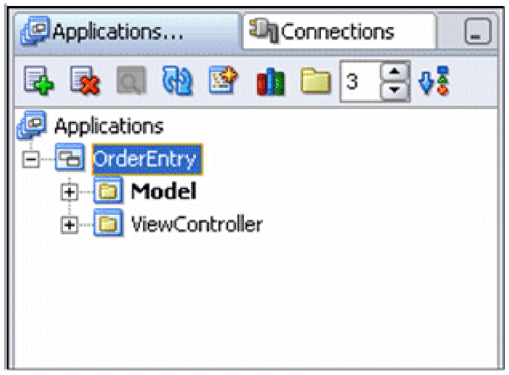
Application Template: Web Application [JSF, ADF BC]

Template Description: This application template is configured for building a data-bound web application. It consists of one project for the view and controller components (JSF), and another project for the data model (ADF Business Components).

   Click **OK**.

4. The Application should contain two projects: `Model` and `ViewController`.

Figure 18–12 OrderEntry in JDeveloper



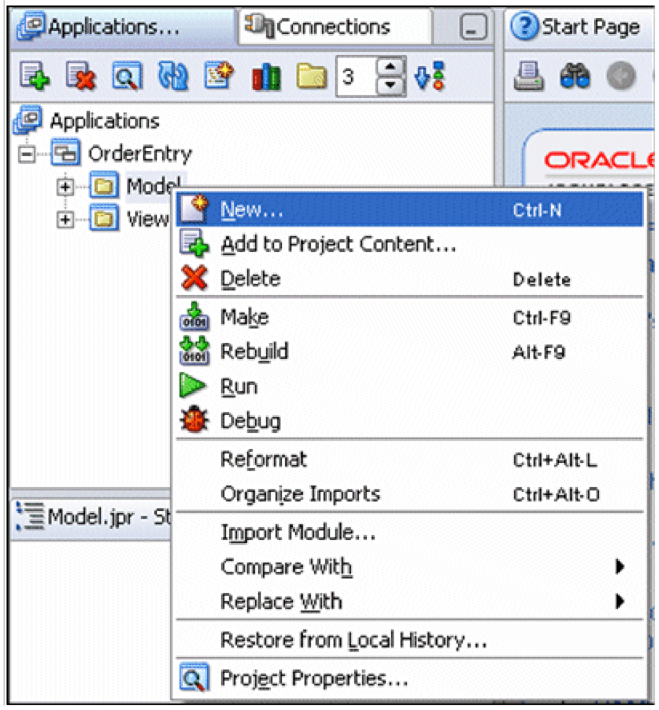
You now have an application and projects to contain and manage your application.

18.3.1.2 Create Business Components

In this section, you create ADF Business Components based on tables in the database. For this example, use the `adfconn` database connection, which you created earlier. You create these objects in the Model project.

1. In the Applications Navigator, right-click the Model project and select **New** from the context menu.

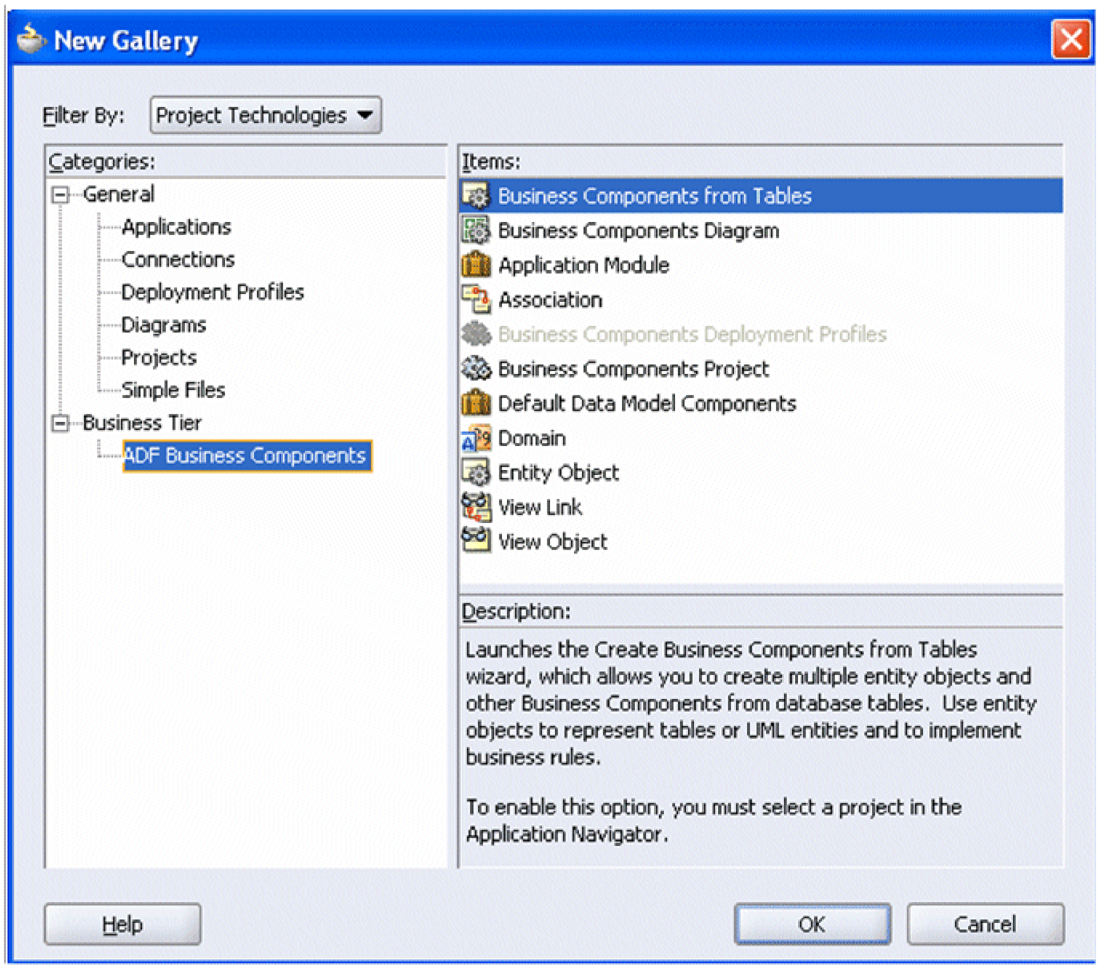
Figure 18–13 New Object in Model Project



2. In the New Gallery, expand Business Tier and select **ADF Business Components** in the Categories list.

   Select **Business Components from Tables** in the Items list.

Figure 18–14 Select ADF Business Components from Tables



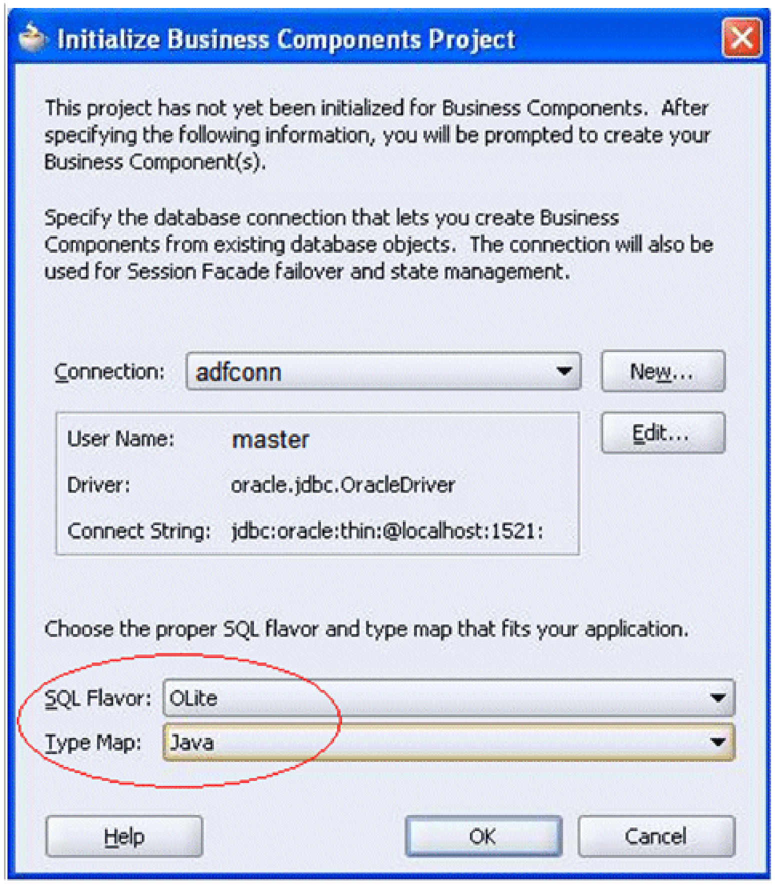
Launches the Create Business Components from Tables wizard, which allows you to create multiple entity objects and other Business Components from database tables. Use entity objects to represent tables or UML entities and to implement business rules.

To enable this option, you must select a project in the Application Navigator.

Click **OK**.

3. In the Business Components Project Initialization window, select the `adfconn` connection from the Connection list. Change SQL Flavor to OLite and Type Map to Java, and then click **OK**.

Figure 18–15 Initialize Business Components Project



Initialize Business Components Project

This project has not yet been initialized for Business Components. After specifying the following information, you will be prompted to create your Business Component(s).

Specify the database connection that lets you create Business Components from existing database objects. The connection will also be used for Session Facade failover and state management.

Connection: adfconn

User Name: master

Driver: oracle.jdbc.OracleDriver

Connect String: jdbc:oracle:thin:@localhost:1521:

Choose the proper SQL flavor and type map that fits your application.

SQL Flavor: OLite

Type Map: Java

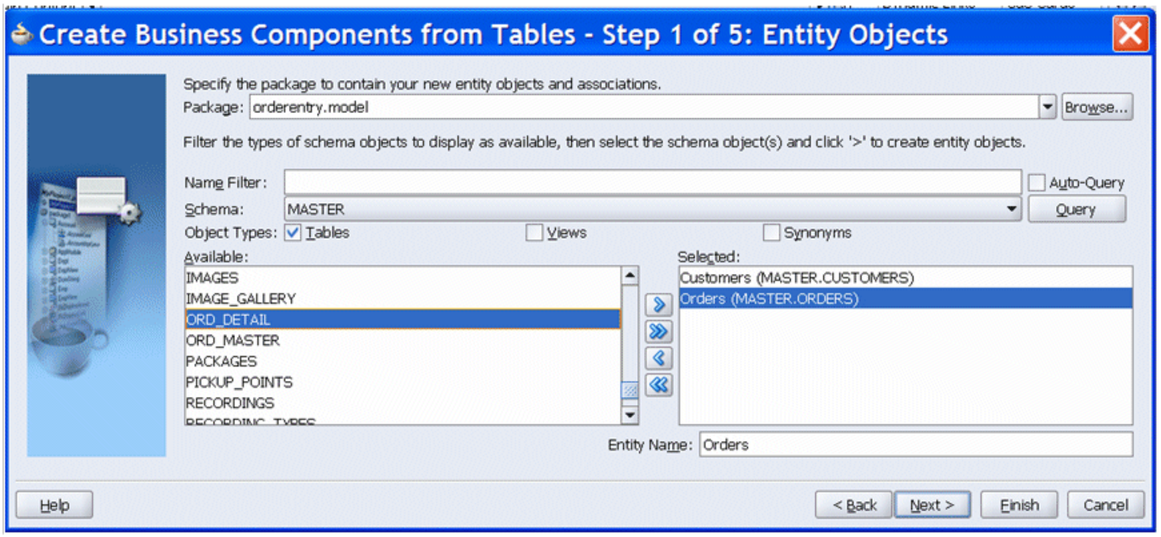
4. If the Welcome page of the Create Business Components wizard appears, click **Next**. If no package name is specified when creating the application, by default it takes the project name, which is `model`.

Note: An ADF Entity Object is a Java component that represents a row in an underlying database table as a domain business object in your J2EE application. It encapsulates the business rules for that domain object and automatically handles saving any change made by the user back to the database. If you are familiar with Oracle Forms, the entity object provides functionality similar to the Oracle Forms record manager, but with the ability to associate encapsulated business rules with each type of 'business record' structure.

---

5. Select the tables for the business component, as follows:
   a. Select **MASTER** from the Schema drop down list.
   b. Click **Query** to populate the list of available tables.
   c. Control-click to select both `CUSTOMERS` and `ORDERS` in the Available list.
   d. Click the right arrow to move both tables to the Selected list.

Figure 18–16 Select Tables for Business Components



Create Business Components from Tables - Step 1 of 5: Entity Objects

Specify the package to contain your new entity objects and associations.
Package: orderentry.model

Filter the types of schema objects to display as available, then select the schema object(s) and click '>' to create entity objects.

Name Filter:
Auto-Query
Schema: MASTER
Object Types: [✓ Tables] Views Synonyms

Available:
IMAGES
IMAGE_GALLERY
ORD_DETAIL
ORD_MASTER
PACKAGES
PICKUP_POINTS
RECORDINGS

Selected:
Customers (MASTER.CUSTOMERS)
Orders (MASTER.ORDERS)

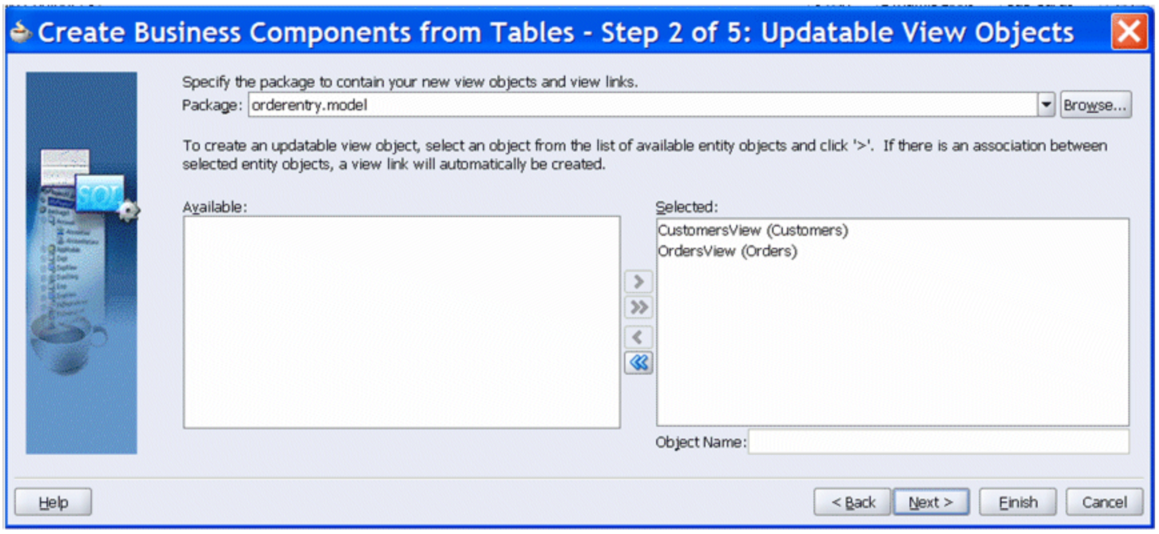
Entity Name: Orders

e. Click **Next** to continue.

6. On the Updatable View Objects page of the Create Business Components Wizard, select both Entity objects and click the right arrow button to move both tables to the Selected list.

Note: An ADF View Object is a Java component that represents a SQL query against one or more underlying tables. It allows you to project, join, filter, and sort business information in exactly the way the end-user needs to see it for the user interface you need to provide to your end users. When related to underlying ADF Entity Objects, the view object allows users to create, update, and remove rows with automatic enforcement of business rules. If you are familiar with Oracle Forms, the view object provides functionality similar to the Oracle Forms Data Block, but adds the flexibility to finely tune the SQL query and to automatically leverage centralized business rules encapsulated by the entity object.

Figure 18–17 Updatable View Objects



Create Business Components from Tables - Step 2 of 5: Updatable View Objects

Specify the package to contain your new view objects and view links.
Package: orderentry.model

To create an updatable view object, select an object from the list of available entity objects and click '>'. If there is an association between selected entity objects, a view link will automatically be created.

Available:

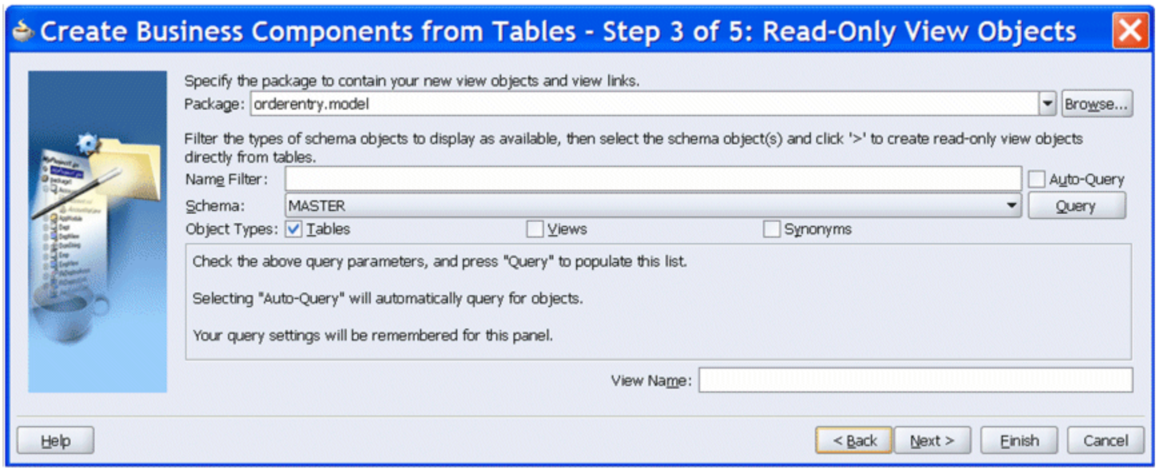
Selected:
CustomersView (Customers)
OrdersView (Orders)

Object Name:

Click **Next**.

7. Skip the Read-only View Objects page of the wizard by clicking **Next**. You will only be using view objects that can be updated.

Figure 18–18 Read-Only View Objects



Create Business Components from Tables - Step 3 of 5: Read-Only View Objects

Specify the package to contain your new view objects and view links.
Package: orderentry.model

Filter the types of schema objects to display as available, then select the schema object(s) and click '>' to create read-only view objects directly from tables.
Name Filter:
Auto-Query
Schema: MASTER
Object Types: [✓ Tables] Views Synonyms

Check the above query parameters, and press "Query" to populate this list.

Selecting "Auto-Query" will automatically query for objects.

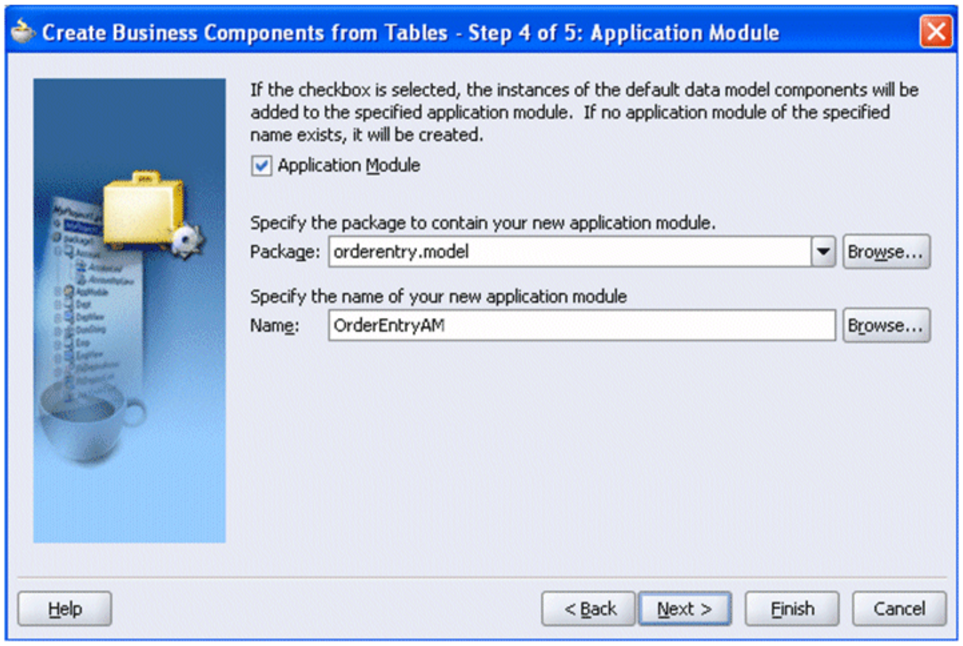
Your query settings will be remembered for this panel.

View Name:

8. On the Application Module page of the wizard, name the application module `OrderEntryAM`.

Note: An ADF Application Module is a Java component that represents a transactional data model of master/detail-related view object queries. It allows client interface technologies of any kind in a service-oriented architecture to easily manipulate the business information exposed by the view object instances contained in its data model. If you are familiar with Oracle Forms, the application module provides the functionality of a transactional data container similar to the Oracle Forms Form object, but is designed to allow any kind of user interface to work with the data in its view object 'data blocks'.

Figure 18–19 Application Module


Create Business Components from Tables - Step 4 of 5: Application Module

If the checkbox is selected, the instances of the default data model components will be added to the specified application module. If no application module of the specified name exists, it will be created.

☑ Application Module

Specify the package to contain your new application module.

Package: orderentry.model

Specify the name of your new application module

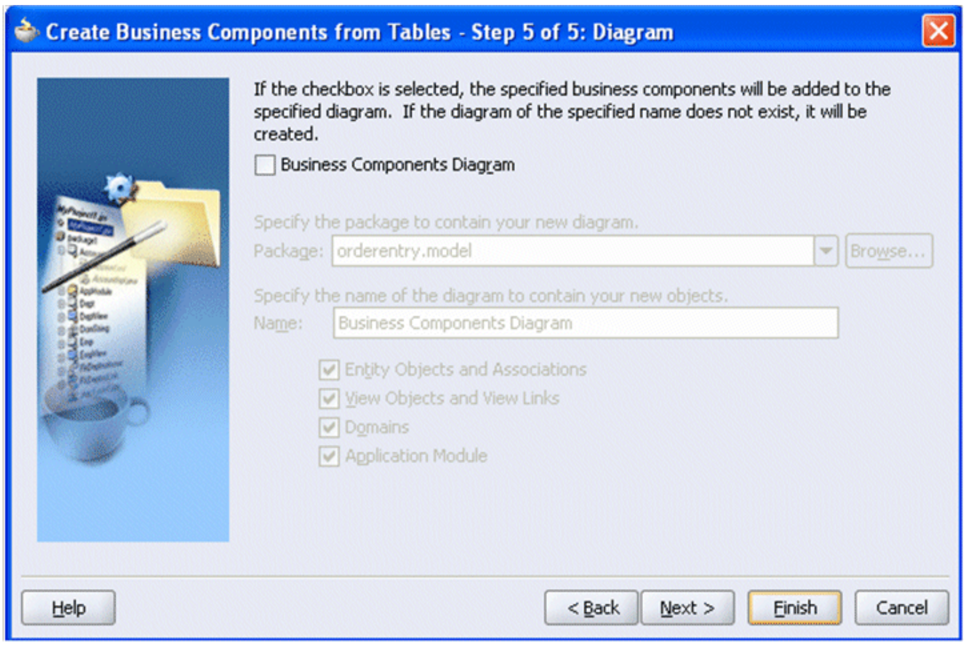
Name: OrderEntryAM

Click **Next**.

9. JDeveloper provides several different techniques for managing components. One is to use a diagram of the components and their relationships. In this step, JDeveloper provides such a diagram option.

   For this tutorial, you will not use this option. Click **Next** to continue.

Figure 18–20 Diagram


Create Business Components from Tables - Step 5 of 5: Diagram

If the checkbox is selected, the specified business components will be added to the specified diagram. If the diagram of the specified name does not exist, it will be created.

☐ Business Components Diagram

Specify the package to contain your new diagram.

Package: orderentry.model

Specify the name of the diagram to contain your new objects.

Name: Business Components Diagram

☑ Entity Objects and Associations

☑ View Objects and View Links

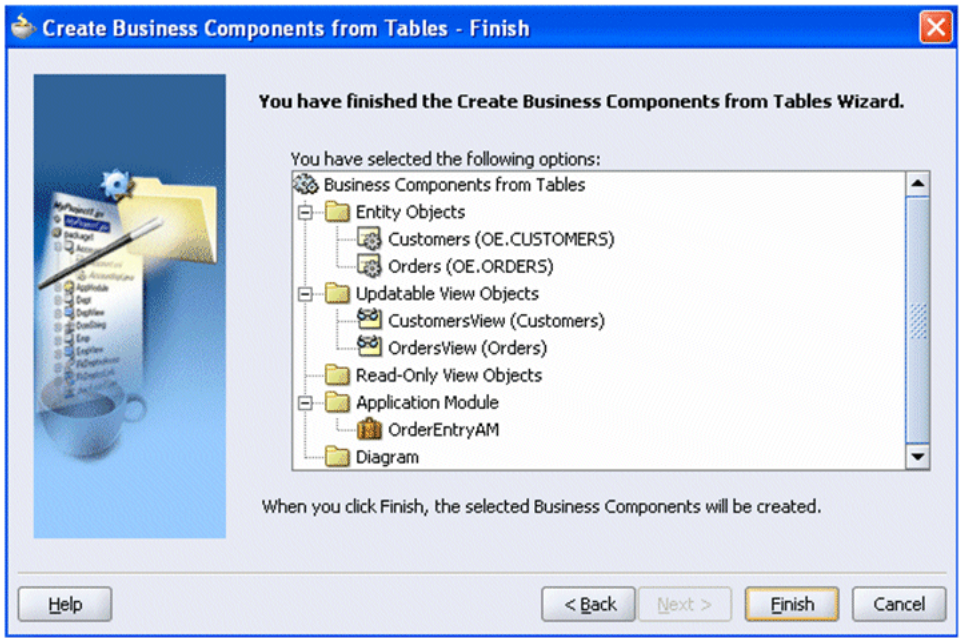
☑ Domains

☑ Application Module

10. The final page of the Business Components Wizard shows the objects and relationships that will be created.

Click **Finish** to complete the wizard actions.

Figure 18–21 Finish



You have finished the Create Business Components from Tables Wizard.

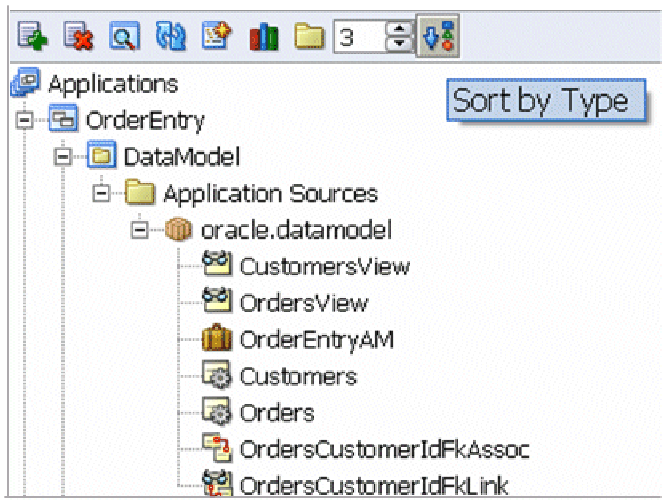
You have selected the following options:

- Business Components from Tables
  - Entity Objects
    - Customers (OE.CUSTOMERS)
    - Orders (OE.ORDERS)
  - Updatable View Objects
    - CustomersView (Customers)
    - OrdersView (Orders)
  - Read-Only View Objects
  - Application Module
    - OrderEntryAM
  - Diagram

When you click Finish, the selected Business Components will be created.

11. Using the far right button of the toolbar in the navigator pane, sort elements by type.

Figure 18–22 Sort by Type



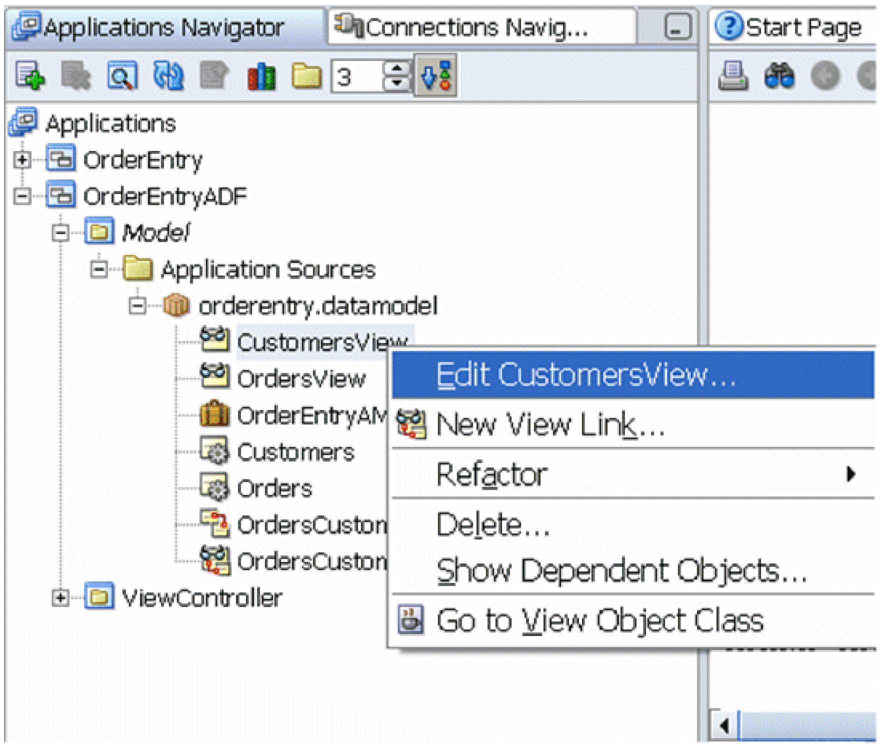
18.3.2 Customize the Business Components Views

In the previous sections, you created some default Business Components from two tables (`Customers` and `Orders`). The default view objects expose all of the columns from those tables. For your application, you want to expose only a few of those columns. ADF BC allows you to easily customize hose objects to fit your specific application needs.

In the following steps, you will add an Order By clause to the CustomersView to make sure the returned data is sorted by customer ID.

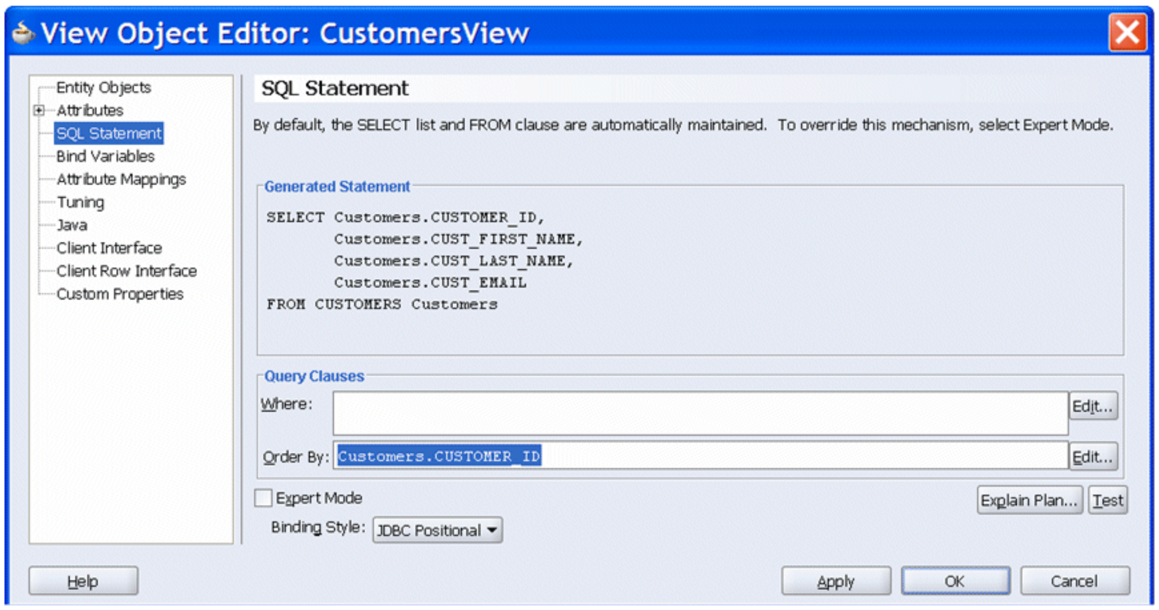
1. In the Applications Navigator, right-click the CustomersView node and select **Edit CustomersView** from the context menu.

Figure 18–23 Edit CustomersView



2. Select **SQL Statement** and add an `Order By` clause to the CustomersView to make sure the returned data is sorted by customer ID.

Figure 18–24 View Object Editor



By default, the SELECT list and FROM clause are automatically maintained. To override this mechanism, select Expert Mode.

Generated Statement

```
SELECT Customers.CUSTOMER_ID,
       Customers.CUST_FIRST_NAME,
       Customers.CUST_LAST_NAME,
       Customers.CUST_EMAIL
FROM CUSTOMERS Customers
```

Query Clauses

Where:

Order By: Customers.CUSTOMER_ID

Expert Mode

Binding Style: JDBC Positional

Click **OK** to apply the changes and exit the View Object Editor.

3. Click the **Save All** icon on the JDeveloper menu bar, or select **File > Save All** from the menu. You have now customized the Customers view to meet the specific needs of your application.

18.3.3 Creating a Master-Detail JavaServer Faces Page

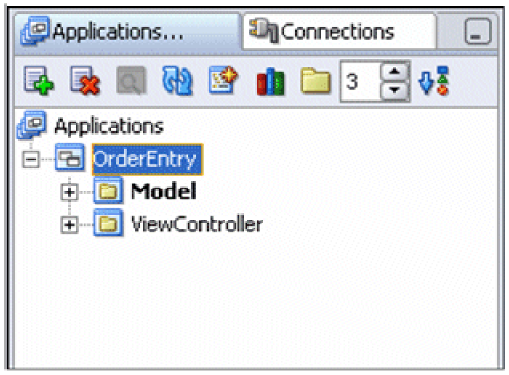
Conforming to the JSF standards, ADF Faces lets you concentrate on the application and layout rather than markup language and tags. Due to the integration of ADF Faces and ADF Business Components, you can easily change the default field labels for the user interface from within ADF Business Components.

In the next few steps, you create an ADF Faces application based on the ADF BC model that you just built. You also modify some of the ADF BC default settings to help enhance the default UI.

1. When you created the application, two projects were defined: `Model` and `ViewController`. The `Model` project contains the business components that serve as the data model for your application. The `ViewController` project will include the View portion of your application, which defines the user interface.

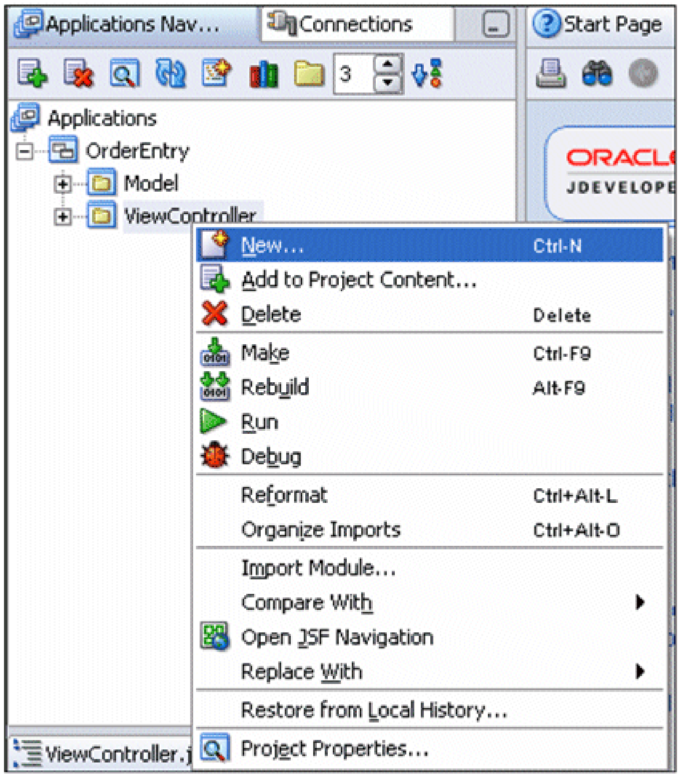
   Collapse the Model node so that the Applications Navigator appear as follows:

Figure 18–25 OrderEntry in JDeveloper



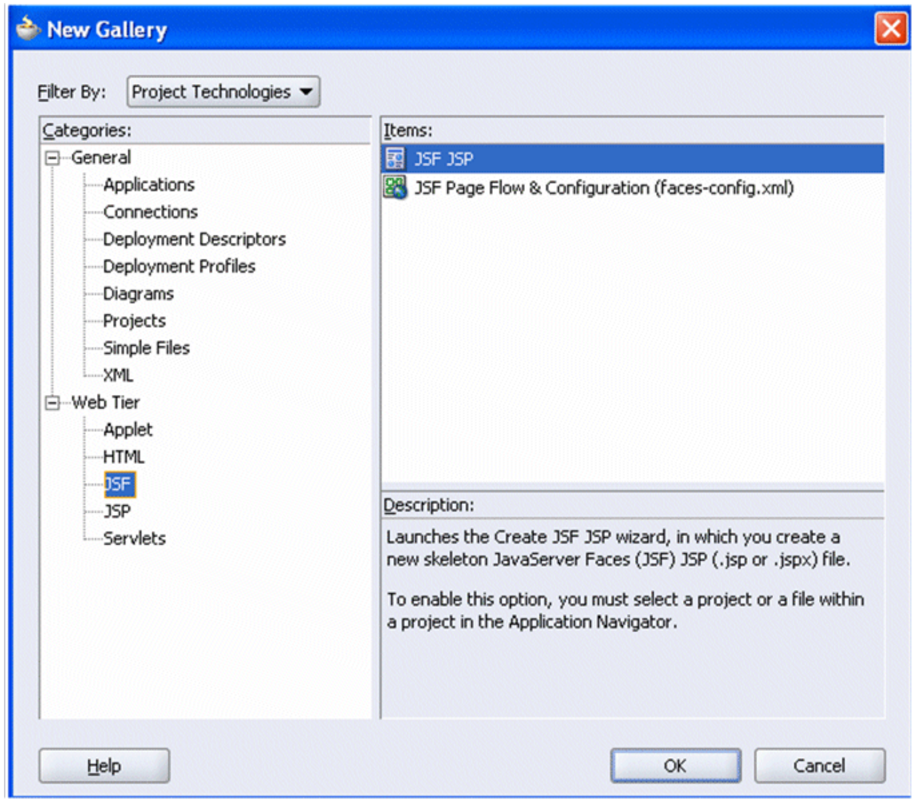
2. Create a new JSF by right-clicking ViewController in the Applications Navigator and selecting **New** from the context menu.

Figure 18–26 New Object Under ViewController



3. Select **JSF JSP** from the JSF Category.

Figure 18–27 New JSF JSP

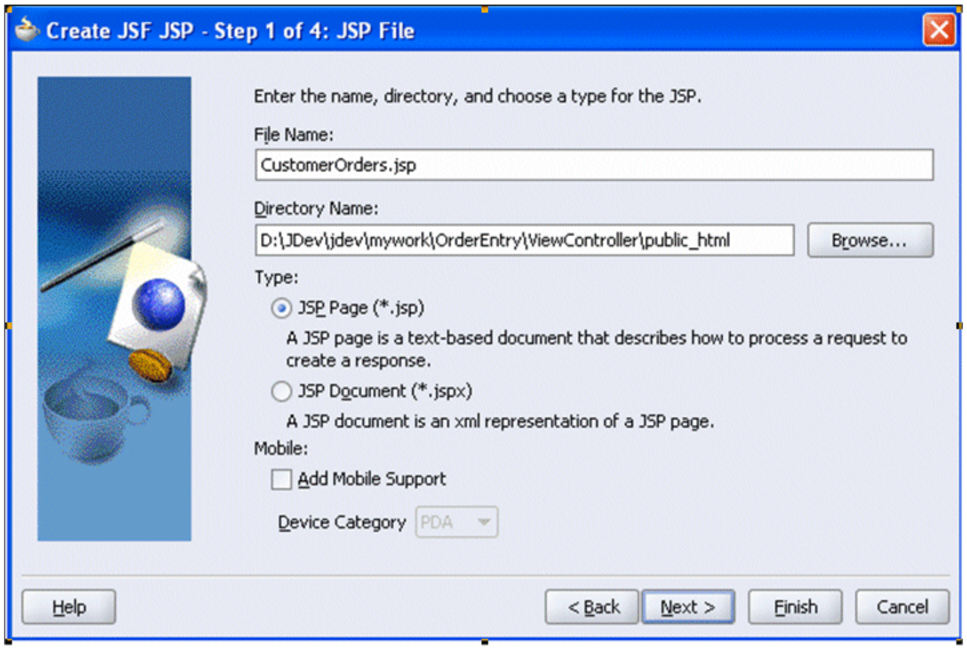


Launches the Create JSF JSP wizard, in which you create a new skeleton JavaServer Faces (JSF) JSP (.jsp or .jspx) file.

To enable this option, you must select a project or a file within a project in the Application Navigator.

4. Selecting a new JSF opens the Create JSF JSP Wizard. Perform the following for creating the `CustomerOrders.jsp`:
   a. Click **Next** to skip the Welcome page of the JSF JSP Wizard, if it appears.
   b. Name the new JSP `CustomerOrders.jsp`. Accept the other defaults and click **Next** to continue.

Figure 18–28 Step 1 of Creating JSP



Create JSF JSP - Step 1 of 4: JSP File

Enter the name, directory, and choose a type for the JSP.

File Name: CustomerOrders.jsp

Directory Name: D:\JDev\jdev\mywork\OrderEntry\ViewController\public_html

Type:

[✓ JSP Page (*.jsp)]

A JSP page is a text-based document that describes how to process a request to create a response.

JSP Document (*.jspx)

A JSP document is an xml representation of a JSP page.

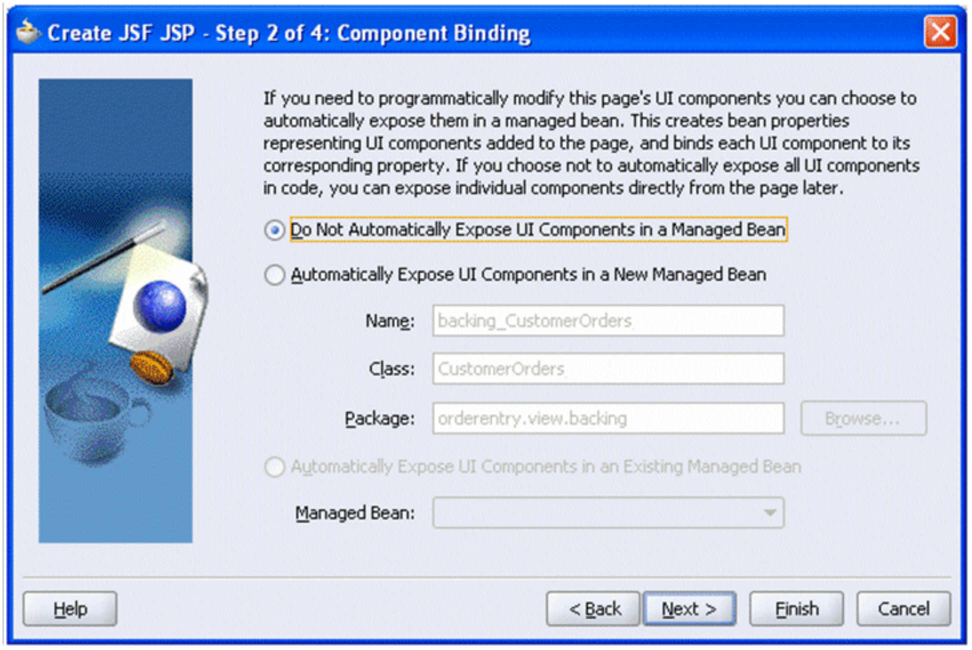
Mobile:

☐ Add Mobile Support

Device Category: PDA

c. On the next page, Component Binding, select the **Do Not Automatically Expose UI Components** option. Leave other default values and click **Next**.

Figure 18–29 Step 2 of Creating JSP



Create JSF JSP - Step 2 of 4: Component Binding

If you need to programmatically modify this page's UI components you can choose to automatically expose them in a managed bean. This creates bean properties representing UI components added to the page, and binds each UI component to its corresponding property. If you choose not to automatically expose all UI components in code, you can expose individual components directly from the page later.

[✓ Do Not Automatically Expose UI Components in a Managed Bean]

Automatically Expose UI Components in a New Managed Bean

Name: backing_CustomerOrders

Class: CustomerOrders

Package: orderentry.view.backing

Automatically Expose UI Components in an Existing Managed Bean

Managed Bean:

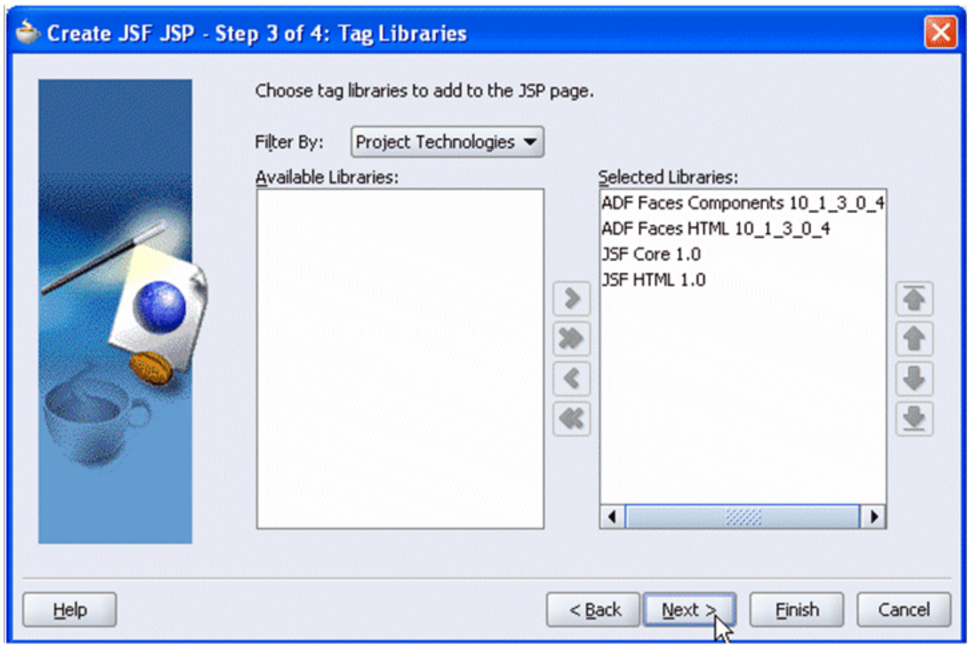
d. Select **libraries** in the Available Libraries window, and use the Add button to move them into the Selected Libraries section, as needed. Make sure the following libraries appear in Selected Libraries:

– JSF Core 1.0

– JSF HTML 1.0

- ADF Faces Components
- ADF Faces HTML

Figure 18–30 Step 3 of Creating JSP



Choose tag libraries to add to the JSP page.

Filter By: Project Technologies

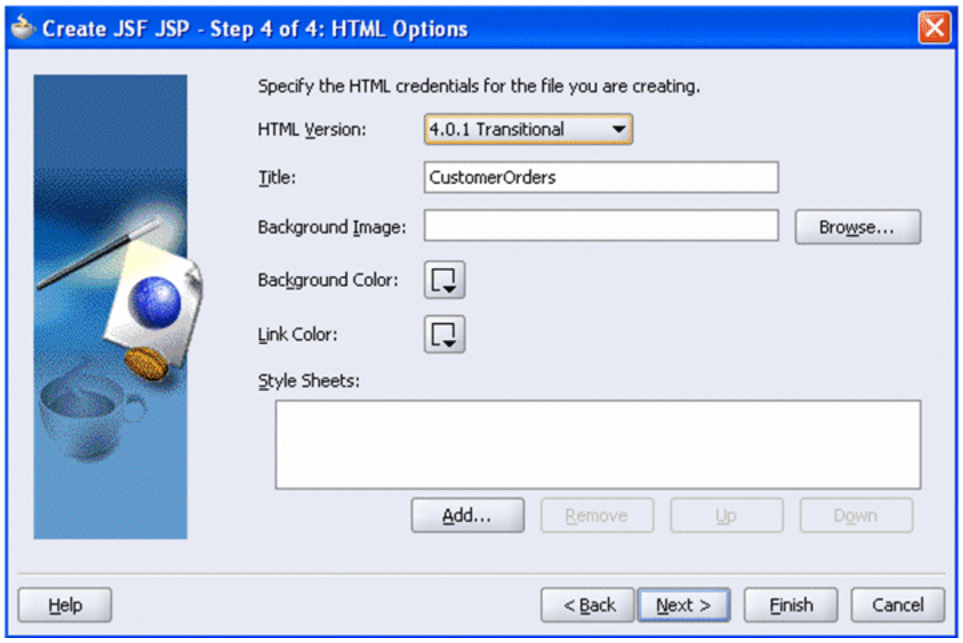
Available Libraries:

Selected Libraries:
ADF Faces Components 10_1_3_0_4
ADF Faces HTML 10_1_3_0_4
JSF Core 1.0
JSF HTML 1.0

Click **Next** to accept these libraries.

e. Click **Finish** to accept the default HTML options and create the new JSP.

Figure 18–31 Step 4 of Creating JSP



Specify the HTML credentials for the file you are creating.

HTML Version: 4.0.1 Transitional

Title: CustomerOrders

Background Image:

Background Color:

Link Color:

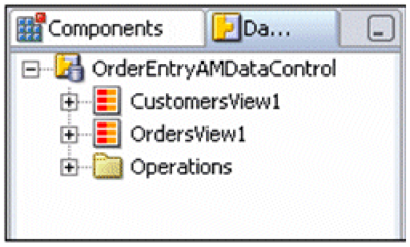
Style Sheets:

You now have an empty `CustomerOrders.jsp` page. In the next few steps, add a data-bound ADF Faces component to the page. This component displays a customer along with the orders that the customer has placed.

When you created the `CustomerOrders.jsp` page, JDeveloper opened it in a visual editor in the center of the JDeveloper IDE. You add the ADF Faces components by dragging them from either the Component Palette or the Data Control Palette to the visual editor. Here you will drop some databound components based on the view objects you created earlier using the Data Control Palette.

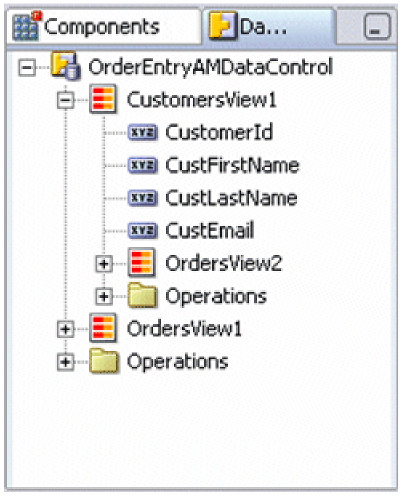
1. Expand OrderEntryDataAMControl in the Data Control palette.

Figure 18–32 Data Control Palette



2. Expand `CustomersView1`.

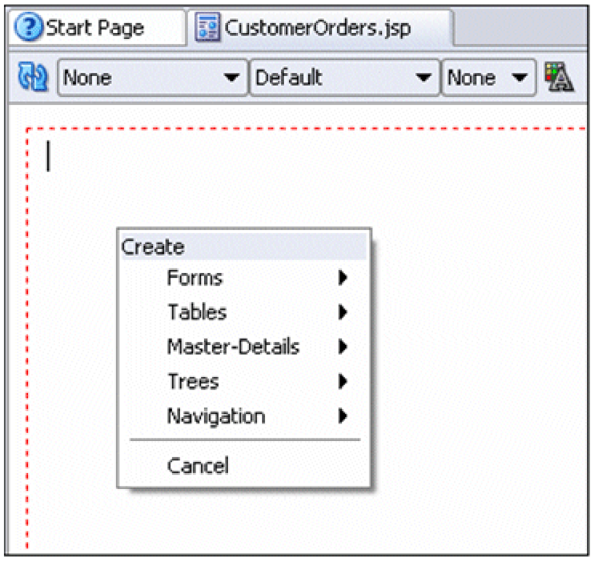
Figure 18–33 Expand CustomersView1



Note: By default, the Business Components from Tables wizard noticed the foreign key relationships between the `ORDERS` and `CUSTOMER` tables and created a default data model in the OrderEntryDataAM data model that features both an `OrdersView1`, allowing us to see all orders, as well as an `OrdersView2` that is linked with the `OrdersView1` showing all of the customers. In this scenario, we'll use the `CustomersView1` and the `OrdersView2` that displays customers and their set of orders.

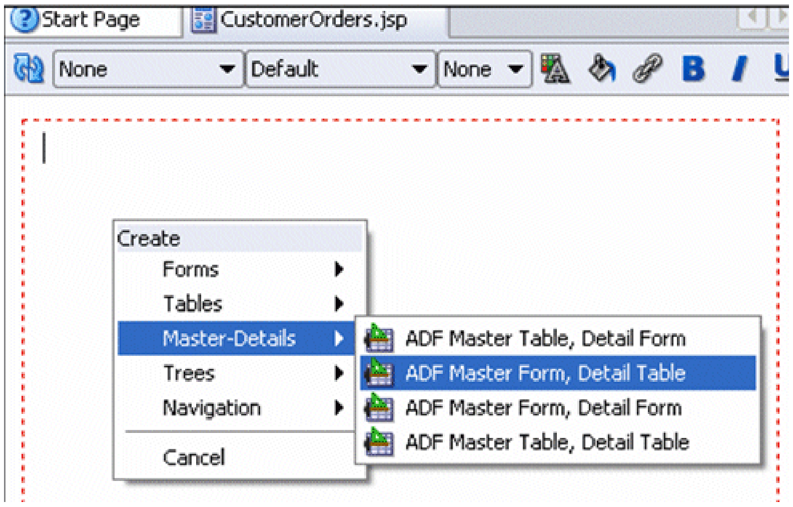
3. Drag `OrdersView2` to the visual editor. JDeveloper opens a context menu with the available options for that data control.

Figure 18–34 Visual Editor



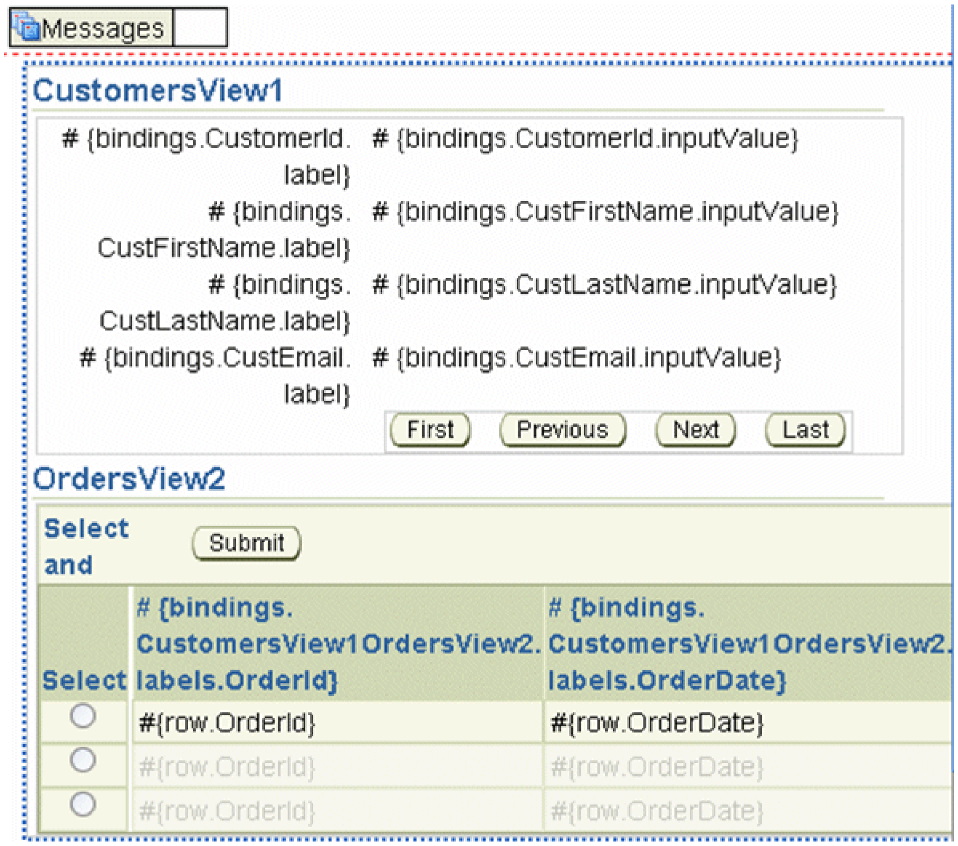
4. Place your cursor over the Master-Details option, and then select **ADF Master Form, Detail Table**.

Figure 18–35 Master-Details Selection


Start Page
CustomerOrders.jsp
None
Default
None
Create
Forms
Tables
Master-Details
Trees
Navigation
Cancel
ADF Master Table, Detail Form
ADF Master Form, Detail Table
ADF Master Form, Detail Form
ADF Master Table, Detail Table

5. JDeveloper adds the ADF Master Detail component to your JSF page.

Figure 18–36 Add ADF Master-Detail to JSP

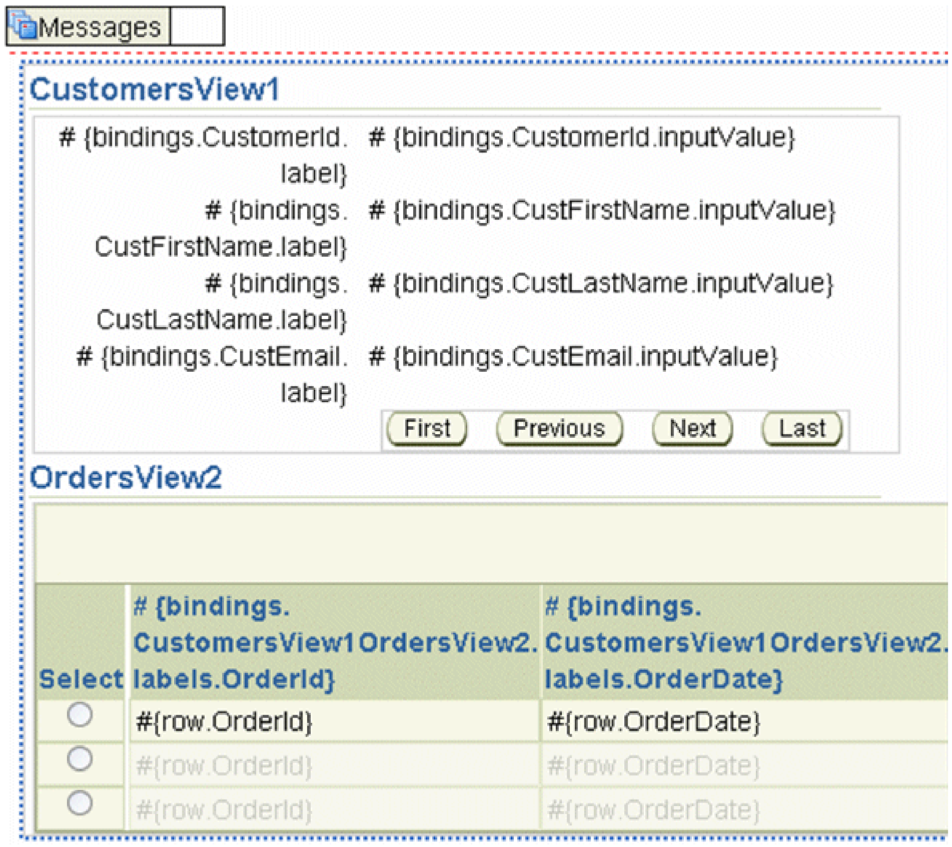


# {bindings.CustomerId.label}	# {bindings.CustomerId.inputValue}
# {bindings.CustFirstName.label}	# {bindings.CustFirstName.inputValue}
# {bindings.CustLastName.label}	# {bindings.CustLastName.inputValue}
# {bindings.CustEmail.label}	# {bindings.CustEmail.inputValue}

Select	# {bindings.CustomersView1OrdersView2.labels.OrderId}	# {bindings.CustomersView1OrdersView2.labels.OrderDate}
○	#{row.OrderId}	#{row.OrderDate}
○	#{row.OrderId}	#{row.OrderDate}
○	#{row.OrderId}	#{row.OrderDate}

6. In the JSF Page in the OrderView2, eliminate the Submit button by selecting the Component **Submit** and click the Delete key on your keyboard. The page should look as follows:

Figure 18–37 Submit Components


Messages

CustomersView1

# {bindings.CustomerId.label}	# {bindings.CustomerId.inputValue}
# {bindings.CustFirstName.label}	# {bindings.CustFirstName.inputValue}
# {bindings.CustLastName.label}	# {bindings.CustLastName.inputValue}
# {bindings.CustEmail.label}	# {bindings.CustEmail.inputValue}

OrdersView2

Select	# {bindings.CustomersView1OrdersView2.labels.OrderId}	# {bindings.CustomersView1OrdersView2.labels.OrderDate}
○	#{row.OrderId}	#{row.OrderDate}
○	#{row.OrderId}	#{row.OrderDate}
○	#{row.OrderId}	#{row.OrderDate}

You now have a complete JSF that is databound to your ADF BC business services.

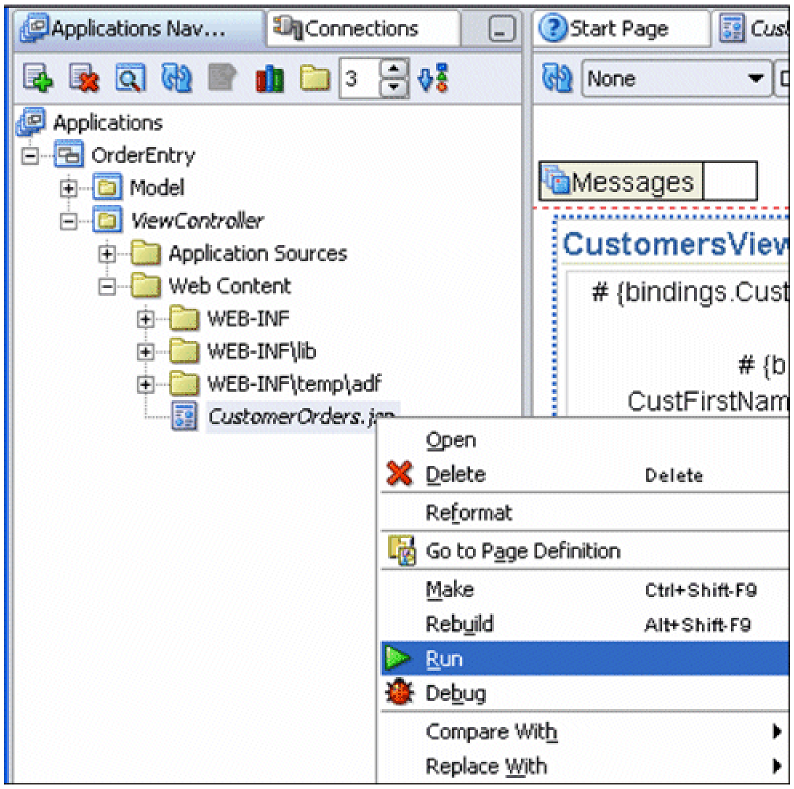
18.3.4 Running the JSF Page

Now that you have built your new ADF Faces application, you need to test it. JDeveloper makes it easy to test JSF through a built-in OC4J server. The server is automatically launched when you test a page from within JDeveloper.

The next few steps take you through the testing process.

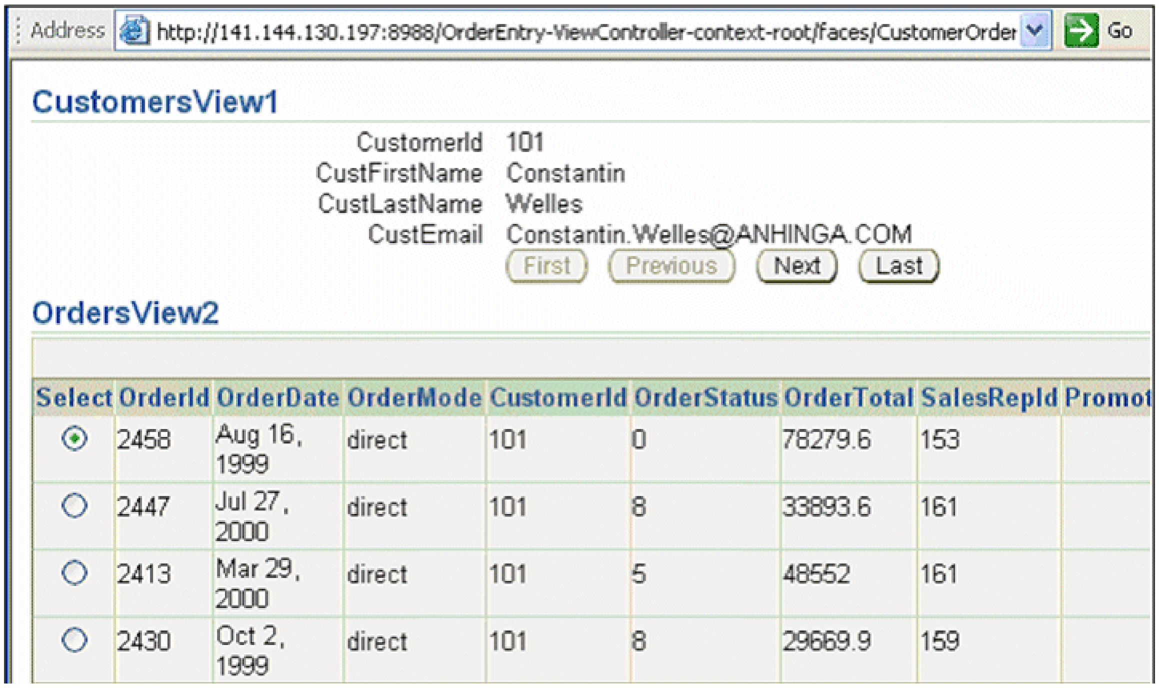
1. To test the page, right-click `CustomerOrders.jsp` in the Applications Navigator and select **Run** from the context menu. Alternatively, you can right-click inside the visual editor and select **Run** from that context menu.

Figure 18–38 Test the Page



2. After execution, the results page should be as follows:

Figure 18–39 Results Page



Address http://141.144.130.197:8988/OrderEntry-ViewController-context-root/faces/CustomerOrder

CustomersView1

CustomerId	101
CustFirstName	Constantin
CustLastName	Welles
CustEmail	Constantin.Welles@ANHINGA.COM

OrdersView2

Select	OrderId	OrderDate	OrderMode	CustomerId	OrderStatus	OrderTotal	SalesRepId	Promot
●	2458	Aug 16, 1999	direct	101	0	78279.6	153	
○	2447	Jul 27, 2000	direct	101	8	33893.6	161	
○	2413	Mar 29, 2000	direct	101	5	48552	161	
○	2430	Oct 2, 1999	direct	101	8	29669.9	159	

Note: JDeveloper will open your default web browser and display the page. If this doesn't happen, visit the Tools -> Preferences and select the Web Browser and Proxy category. Here you can enter the command line to your preferred browser. Then, try running the page again after setting this preference.

---

3. Navigate through the customer rows to see the differences in the orders that each customer has placed. Note that the first few customers in the list have multiple orders.

When you are finished, close the browser. Make sure you stop the JDeveloper OC4J Server before proceeding to next section. To stop the JDeveloper OC4J Server, select **Run -> Terminate -> Embedded OC4J Server**.

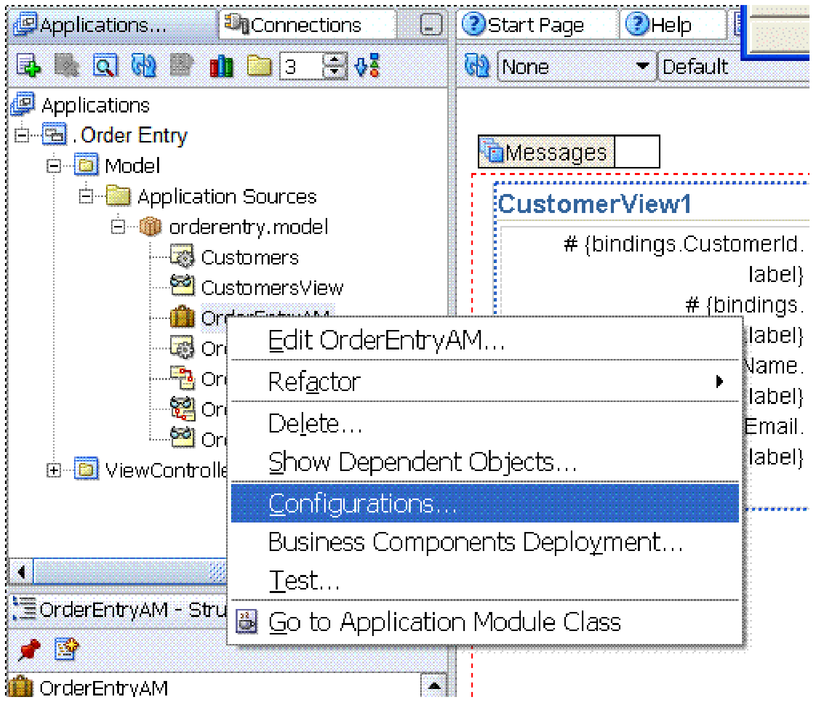
18.3.5 Configure the ADF/BC4J Application for the Oracle Database Lite Environment

Using JDeveloper, configure the application to use the Oracle Database Lite environment, as described in the following sections:

Change the application configuration to use the `WTGJdbcConnection`, as follows:

1. In the JDeveloper Applications Navigator, right-click the OrderEntryAM node and select **Configurations**.

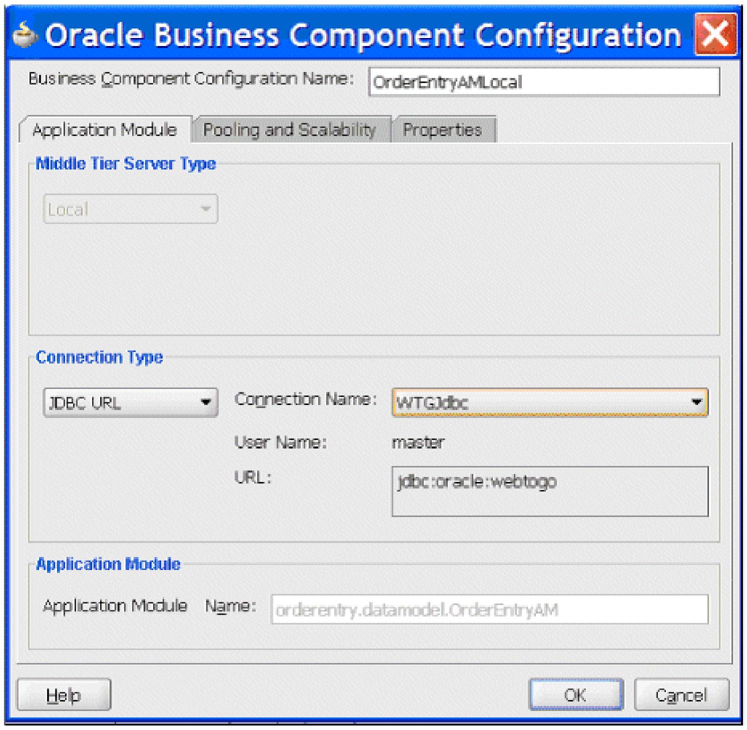
Figure 18–40 Selecting Configurations in JDeveloper



2. Select **JDBCName** and edit the connection. The Oracle Business Component Configuration window is displayed.

3. Select **WTGJdbc** for Connection Name and click **OK**.

Figure 18–41 Business Component Configuration Window



Business Component Configuration Name: OrderEntryAMLocal

Middle Tier Server Type: Local

Connection Type: JDBC URL

Connection Name: WTGJdbc

User Name: master

URL: jdbc:oracle:webtogo

Application Module Name: orderentry.datamodel.OrderEntryAM

4. Save your project.

The ADF/BC4J application has been configured to use Oracle Database Lite connection

18.3.6 Deploy the Application as WAR file

To deploy the application, create a deployment profile and then deploy the application as a WAR file, as follows:

1. In the Application Navigator, select the `CustomerOrders.jsp`.
2. Choose Run -> Deploy -> New Deployment Profile and create a new deployment profile.
3. Choose Run -> Deploy and then select the deployment profile created in step 2.

18.4 Package the ADF/BC4J Application

In order to package the ADF/BC4J application, you must perform the following:

- Section 18.4.1, "Include the ADF Runtime Libraries with the ADF/BC4J Application"
- Section 18.4.2, "Package the Application from the Packaging Wizard"

18.4.1 Include the ADF Runtime Libraries with the ADF/BC4J Application

In order for the application to execute correctly, the ADF runtime libraries must be included. Perform the following to include these libraries in your ADF/BC4J application:

1. Unarchive the WAR file with the ADF/BC4J application into a temporary directory, such as `adftutapp`. This explodes the `CustomerOrders.jsp` and the `WEB-INF` directory into the `adftutapp` directory.
2. Navigate to the `WEB-INF\lib` directory and copy all ADF/BC4J runtime libraries to this directory.

> **Note:** For information regarding list of libraries to be copied, refer to Chapter 22.12.3 "Installing the ADF Runtime Libraries Manually" in the Oracle Application Framework, Developer's Guide, 10*g* Release 3 (10.1.3).

18.4.2 Package the Application from the Packaging Wizard

To package the JSP application, perform the following steps.

1. Copy the `adftutapp` directory and its contents into the following location:

```
<Mobile_ServerHome>\Mobile\Sdk\wtgsdk\root
```

2. Using the Command Prompt window, run the Packaging Wizard and provide the screen inputs that are listed and described in Table 18–2.

Table 18–2 Packaging Wizard Input Details for BC4J Application

Screen	Input	Details
Platform	Oracle Lite Web OC4J; US	N/A
Application	Application Name	ADF BC4J Oracle Database Lite Tutorial Application
Application	Virtual Path	`/bc4jtutorial`
Application	Description	Oracle Lite Tutorial Application
Application	Application Classpath	no input
Application	Default Page	`faces/CustomerOrders.jsp`
Application	Local Application Directory	`<`*`ORACLE_HOME`*`>\Mobile\SDK \wtgsdk\root\adftutapp`
Files	The Packaging Wizard loads all files into the directory under the local application directory.	N/A

> **Note:** For all ADF-based applications, you need to also perform the following:
>
> - For Platform input, the user should select the correct language for the `Oracle Lite Web OC4J; <lang>`.
> - For the Default page Input for the application screen, always prepend the `faces/` directory before the default page name.
>
> If you miss these steps, your application will not perform correctly.

3. Retain the default values for Files, Servlet, Database and Roles screens.

4. On the Snapshots screen, click **Import**. You can now connect to the Oracle Database by providing the values shown in Table 18–3 in the "Connect to Database" dialog:

Table 18–3 Connect to Database Dialog

Field	Description
Username	`master`
Password	`master`
Database URL	`jdbc:oracle:thin:@<database_ hostname>:<port>:<SID>`

5. After specifying the Database Connection values, select **`CUSTOMERS`** from the list of tables and click **Add**.
6. Select `ORDERS` from the list of tables. Click **Add** and then click **Close**.
7. From the snapshot panel, select **Customers** and change the weight from 0 to 1.
8. From the snapshot panel, select **ORDERS** and click **Edit**. Change the weight from 0 to 2.
9. Retain the default values for Sequences and DDLs.
10. Package the ADF/BC4J application into a JAR file.

18.5 Publish and Configure the ADF/BC4J Application from the Mobile Manager

To publish and configure the JSP application from the Mobile Manager, perform the following steps:

1. Using the Command Prompt window, enter `runmobileserver` to start the Mobile Server.
2. Using the following URL, browse the local host.

```
http://<localhost>:<portnumber>/webtogo
```

Note: If the above port number is other than 80, specify the appropriate port number.

3. Login into the Mobile Server using the administrator username and password.
4. Click **Mobile Manager**. Select the Mobile Server tab and then click **Host name**.
5. Click **Applications** and publish the JAR file that you just created.

18.6 Test the ADF/BC4J Application

Perform the following to test your ADF/BC4J application:

1. Log on to the Mobile Server with the administrator username and password.
2. Select **Mobile Manager**.
3. Click on the Mobile Server tab.
4. Select the host.

5. Click **Users**.
6. Create a new user called `tutorial` and grant permission to this user for the "ADF/BC4J Oracle Database Lite Tutorial Application."
7. Test the application by executing the ADF/BC4J application on the Mobile Client for Oracle Lite WEB OC4J, as described in Section 18.7, "Run the ADF/BC4J Application on the Mobile Client for Oracle Lite WEB OC4J".

18.7 Run the ADF/BC4J Application on the Mobile Client for Oracle Lite WEB OC4J

Before you execute, you must have JRE 1.5.x or higher installed for the OC4J client.

To execute the ADF/BC4J application on the Mobile Client for Oracle Lite WEB OC4J, perform the following steps:

1. Point the client machine browser to the following URL:

   ```
   http://<Server_IP_Address>/setup
   ```

   where `Server_IP_Address` is your server machine IP address.
2. Download and install the Mobile Client for Oracle Lite WEB OC4J
3. Point the client machine browser to the following URL: `http://<localhostname>`, where `localhostname` is the client machine host name.
4. Log in to the client machine-using `tutorial` as both the username and password.
5. After the client machine synchronizes the application and data from the server, click the "ADF/BC4J Oracle Database Lite Tutorial" Application link to test the application on the client machine.

19

Tutorial for Building Mobile Applications for Win32

To demonstrate the steps involved in building Mobile applications for the Win32 platform, this tutorial presents a simplified Mobile field service example. The following sections guide you through the Mobile application development process for the Win32 platform. When developing, you can use Visual Studio 2003.

19.1 Plan the Mobile Application Demo for Win32

Let us assume that you have a `TASK` table on the server that contains information about tasks that must be accomplished by your Mobile field service technicians for a day. Listed below is the `TASK` table structure. Each row in the `TASK` table describes work to be done at a customer site.

- `TASK(ID number(4) primary key`
- `Description varchar(40) not null`
- `CustName varchar(30) not null`
- `CustPhone varchar(12)`
- `CustStAddr varchar(40) not null`
- `CustCity varchar(40) not null`
- `Notes varchar(100)`

Let us also assume that you have three service technicians, Tom, Dick, and Harry. You want to assign all the tasks in the City of Cupertino to Tom, those in the City of Mountain View to Dick, and those in the City of Palo Alto to Harry. You envision your application to work as follows:

Each service technician has a laptop that he uses to obtain his task list in the morning. He will perform the task during the day and will update the Notes column of a task with information about its status or what he has done. At the end of his work day, the service technician uploads his changes to the server.

We will assume the following environment for your application.

- The Mobile Server is installed on the machine called `mserver`.

- The test Oracle database that is used to store the application data and the Mobile Server Repository is installed on the machine `oradbserver` with the listener on port 1521. The Oracle database instance name is `orcl`. We will assume that you can log in to the database with the user name `master` and password `master`. You can substitute any user for `master` so long as the user has the right privileges.
- You have already installed the Mobile Development Kit on your development machine.

Our implementation plan is as follows. The exact sequence of commands for each step is given later.

1. Create the `TASK` table in the `oradbserver` and insert some rows into it. This step is not needed if you already have a database that contains a table similar to `TASK`.
2. Use MDW to create a publication that contains a single publication item based on the `TASK` table.
3. Use the Packaging Wizard to define and publish the Mobile Field Service application to the Mobile Server.
4. Use the Mobile Manager to create users Tom, Dick, and Harry on the Mobile Server. Grant all users the privilege to execute the Mobile Field Service application and create a subscription for each of them.
5. Install the Oracle Database Lite 10*g* client on your development machine in a separate directory (emulating a technician's machine). Run the Mobile Sync application to download the Mobile Field Service application (which is currently empty) and data.
6. On your development machine, use mSQL to look at the rows in the `TASK` snapshot and update the rows by entering notes in the Notes column.
7. Synchronize the changes you made in the snapshot with the server database by running the Mobile Sync application again.
8. Connect to the server database and check that your changes are there. Modify the Description of one of the rows for the customer in Cupertino.
9. Run the Mobile Sync application again. You will see the changes that you made on the server are in the snapshot in the client database.
10. Develop a C or C++ program against Oracle Database Lite to:
    - show the tasks to the technician, and
    - let the technician choose a task and enter notes for it
11. Use the Packaging Wizard to update the application to include the above program.

The Mobile Server is now ready for real life testing.

19.2 Description of Tasks for Win32 Demo

The following sections describe the command sequence for successfully creating the Win32 demo:

1. Section 19.2.1, "Create TASK Table on the Server Database"
2. Section 19.2.2, "Create Publication for Application"
3. Section 19.2.3, "Package the Application Using the Packaging Wizard"

19.2.1 Create TASK Table on the Server Database

We will use the Oracle 10*g* thin JDBC driver to connect to the Oracle database running in the `oradbserver` machine. Ensure that the thin JDBC driver (`<`*`ORACLE_HOME`*`>\jdbc\lib\ojdbc14.jar`) file is included in your `CLASSPATH` environment variable. Connect as `master` with password `master`.

```
D:>msql master/master@jdbc:oracle:thin:@oradbserver:1521:orcl
```

Now create the `TASK` table in this database. The SQL script to create and populate the server database is provided in the following directory.

`<`*`ORACLE_HOME`*`>\mobile\sdk\samples\odbc\win32\MFS`

```
SQL>create table TASK(

1> ID number(4) primary key,
2> Description varchar(40) not null,
3> CustName varchar(30) not null,
4> CustPhone varchar(12),
5> CustStAddr varchar(40) not null,
6> CustCity varchar(40) not null,
7> Notes varchar(100));
```

We will now insert four rows into this table.

```
SQL> insert into task values(1,'Refrigerator not
working','Able','408-999-9999','123 Main St.','Cupertino',null);
SQL> insert into task values(2,'Garbage Disposal
broken','Baker','408-888-8888','234 Central Ave','Cupertino',null);
SQL> insert into task values(3,'Refrigerator makes
noise','Choplin','650-777-7777','1 North St.','Mountain View',null);
SQL> insert into task values(4,'Faucet leaks','Dean','650-666-6666','10 University
St.','Palo Alto','Beware of dogs');
SQL> commit;
SQL> exit
```

19.2.2 Create Publication for Application

As described fully in Chapter 5, "Using Mobile Database Workbench to Create Publications", you can use MDW to create your publication. Launch MDW by executing `oramdw` from `<`*`ORACLE_HOME`*`>/Mobile/Sdk/bin`. The following sections detail how to use MDW to create a publication for the application in this tutorial.

Note: While creating this publication, use Chapter 5, "Using Mobile Database Workbench to Create Publications" for a deeper understanding of how to use MDW and the type of information that you must provide.

- Section 19.2.2.1, "Create a Project"
- Section 19.2.2.2, "Create Publication Item"
- Section 19.2.2.3, "Create Publication"

19.2.2.1 Create a Project

Create a new project for this application by selecting **File->New->Project**. This brings up a wizard where you enter the following information:

Note: For more information, see Section 5.2, "Create a Project".

1. Define a name and location for the project.
2. Enter the username, password, JDBC driver type, database host, database port and database SID for the Mobile repository. Username and password are `master/master`, and the database URL is `jdbc:oracle:thin:@oradbserver:1521:orcl`.

   Provide the Mobile Repository access information. Because you are interacting with the repository to create and manipulate synchronization objects, including the SQL scripts for the publication items, you need access to the Mobile Repository.
3. Specify schema username and password. Enter the user and password of the schema owner for the schema that you are using for the Mobile application. The Mobile application schema contains all database tables, views, synonyms used to build the snapshots for the application.
4. Verify the information that you entered and click **Finish**.

19.2.2.2 Create Publication Item

For this project, you need to create the `taskpi` publication item.

Note: For more information, see Section 5.4, "Create a Publication Item".

Perform the following to create the publication item:

1. Start the new publication item wizard by selecting **File->New->Publication Item**.
2. Enter the name as `taskpi` and the type as `Fast`. If you want this publication item to use automatic synchronization, make sure that the "Enable Automatic Synchronization" checkbox is checked. Uncheck to use manual synchronization. Click **Next**.
3. Select the schema name as `MASTER`, the object type as `Table`, and leave the object filter blank. Click **Search**. When the search ends, select the `task` table from the object list. Click **Next**.
4. Click **>>** to select all of the columns in the `task` table. Click **Next**.
5. In the Query tab, select **Edit** to edit the query, as follows:

   ```
   select * from master.task where CustCity = :city
   ```

   Click **Next**.
6. If you checked the 'Enable Automatic Synchronization' checkbox, then an additional screen comes up. This screen enables you to specify users included in the compose. By default, all users are included. Leave checkbox unchecked and click **Next**.
7. The Summary page displays. Click **Finish**.

19.2.2.3 Create Publication

When you have completed the creation of the publication items, create the publication within the project by selecting **File->New->Publication**.

1. In the General tab, enter the name as `task`, which becomes part of the DSN for the client-side database.
2. In the Publication Item tab, click **Add** to add the publication item that you just created with the following configuration:

   ```
   Name: task
   Updatability: Updatable
   Conflict Resolution: Server Wins
   DML Callback: BLANK
   Grouping Function: BLANK
   Priority Condition: BLANK
   My Compose: BLANK
   Weight: 1
   Description: Blank
   ```

3. In the Events tab, set the thresholds for Automatic Synchronization rules, as follows:
   - Sync if number of modified records in database exceeds threshold value
   - Sync if number of modified records in out queue exceeds threshold value
4. Save the publication by selecting **File->Save**.

19.2.3 Package the Application Using the Packaging Wizard

Using the Packaging Wizard, you can package and publish the Task application into the Mobile Server.

Note: For full details on how to use the Packaging Wizard, see Chapter 7, "Using the Packaging Wizard".

You can select and describe the Task application by launching the Packaging Wizard, as follows:

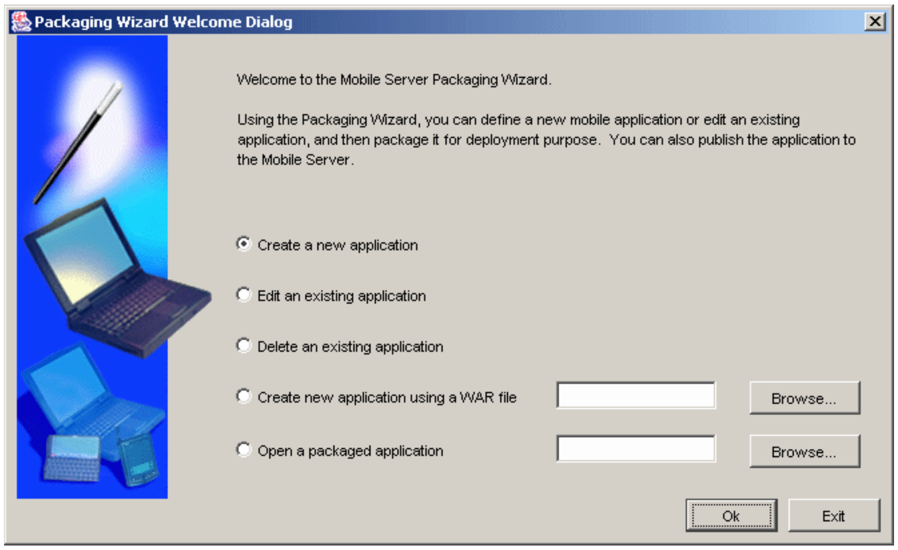
1. Start the Packaging Wizard, as follows:

   ```
   cd <ORACLE_HOME>\Mobile\Sdk\bin
   runwtgpack
   ```

   The Packaging Wizard appears and provides you with the option to create a new application, edit an existing application, delete an existing application, or open a packaged application, as displayed in Figure 19–1.

   Note: Deleting an existing application merely deletes the application from the XML file and does not remove the application from the Mobile Server.

Figure 19–1 Make a Selection Dialog



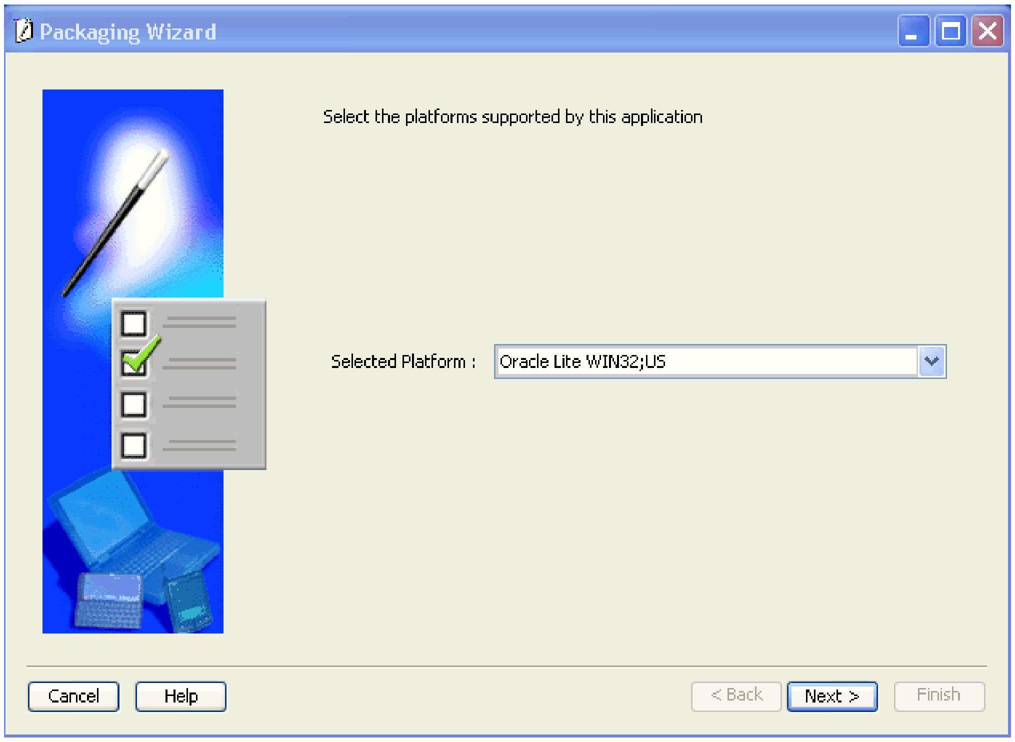
Welcome to the Mobile Server Packaging Wizard.

Using the Packaging Wizard, you can define a new mobile application or edit an existing application, and then package it for deployment purpose. You can also publish the application to the Mobile Server.

[✓ Create a new application]
Edit an existing application
Delete an existing application
Create new application using a WAR file
Open a packaged application

2. Select the **Create a new application** option and click **OK**.
3. The Select a Platform panel appears. As Figure 19–2 displays, this panel enables you to specify the platform for your application. Select **Oracle Lite WIN32;US** from the **Available Platform** list. Click **Next**.

Figure 19–2 Selecting a Platform

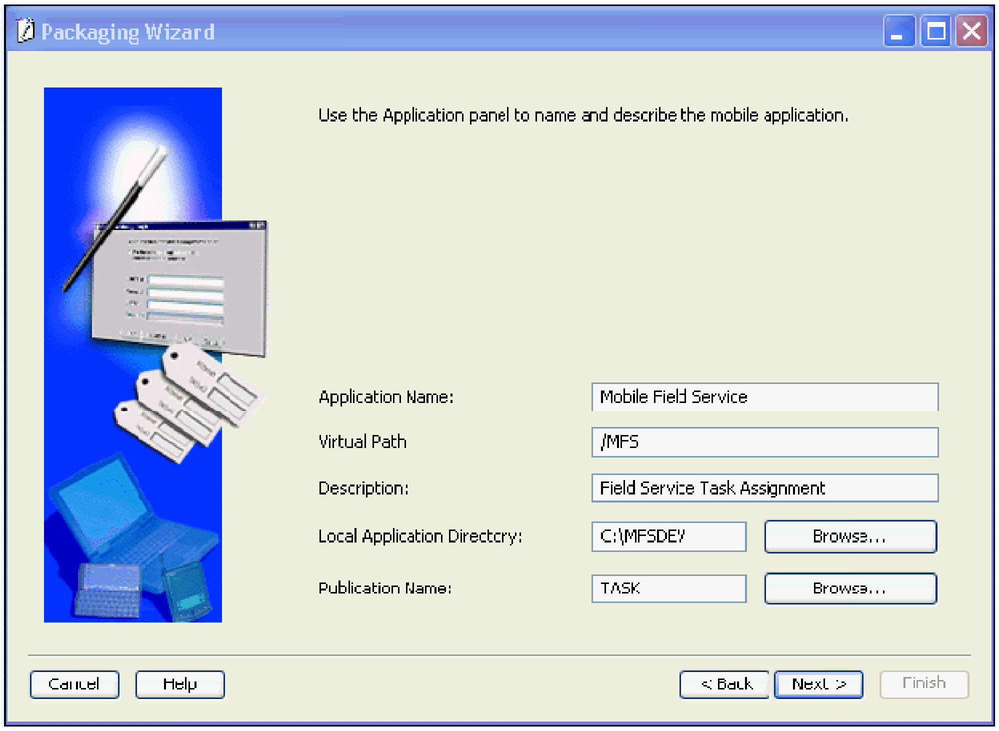


Select the platforms supported by this application

Selected Platform : Oracle Lite WIN32;US

4. As Figure 19–3 displays, the Application panel appears. Use the Application panel to modify "Mobile Field Service" application settings. As Table 19–1 describes, enter the specified values in the corresponding fields.

Figure 19–3 Application Panel



Use the Application panel to name and describe the mobile application.

Application Name: Mobile Field Service

Virtual Path: /MFS

Description: Field Service Task Assignment

Local Application Directory: C:\MFSDEV

Publication Name: TASK

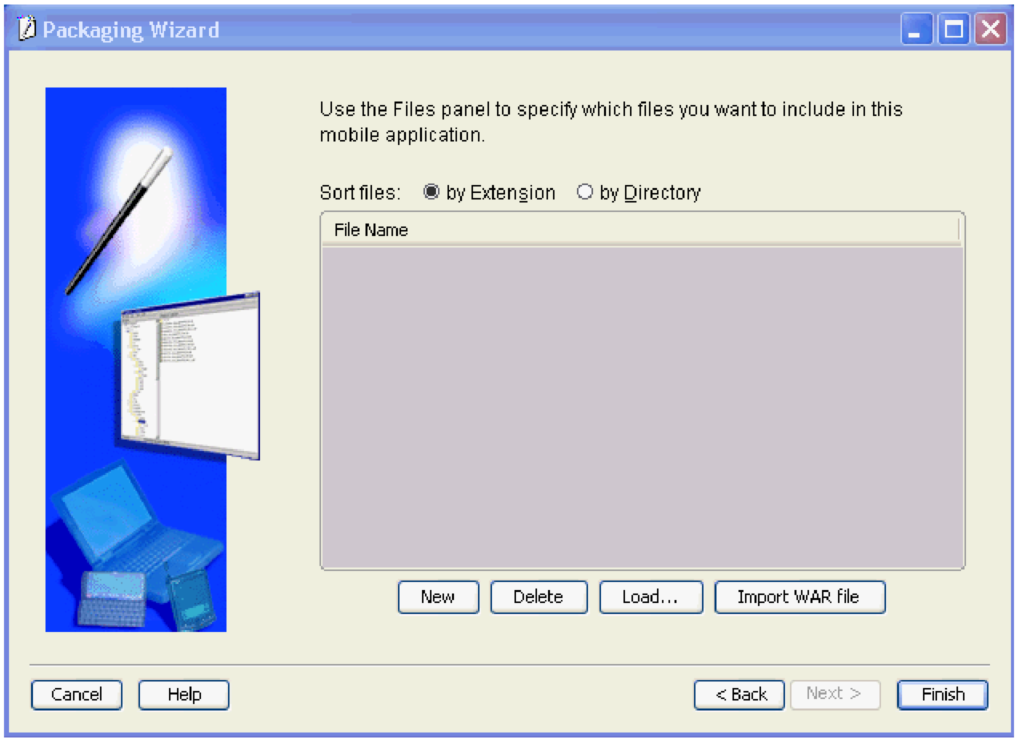
Table 19–1 The Task Application Values

Field	**Value**
Application Name	Mobile Field Service
Virtual Path	/MFS
Description	Field Service Task Assignment
Local Application Directory	`D:/MFSDEV`
Publication Name	Click on Browse. The 'Connect to database' window appears. Enter the following: ■ username: `mobiladmin` ■ password: `welcome123` ■ database URL: `jdbc:oracle:thin:@<hostname>:<port>:<SID>` The next window shows the available publications. Select `task`.

5. Click **Next**. As Figure 19–4 displays, the Files panel appears. Using the Files panel, you can select files that are part of the application. The Packaging Wizard uploads the selected files from the local application directory to the application repository on the Mobile Server.

The Files panel identifies files that the Packaging Wizard uploads from the local application directory to the application repository on the Mobile Server.

Figure 19–4 Uploading Application Files

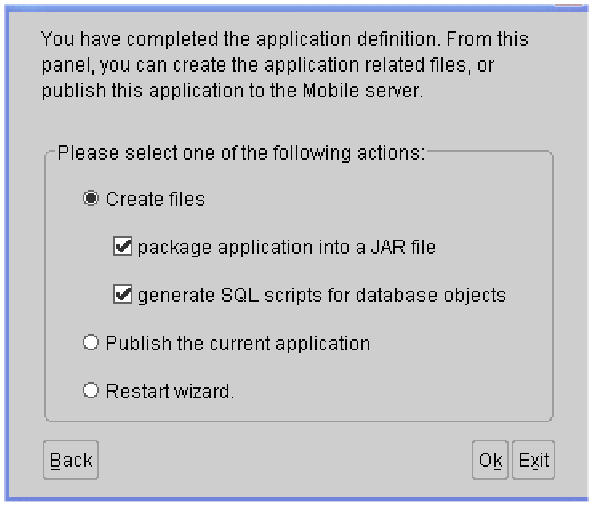


Use the Files panel to specify which files you want to include in this mobile application.

Sort files: [✓ by Extension] by Directory

6. Click **Next** till you arrive at the Application Definition Completed Dialog as shown in Figure 19–5.

Figure 19–5 Application Definition Completed Dialog



You have completed the application definition. From this panel, you can create the application related files, or publish this application to the Mobile server.

Please select one of the following actions:

[✓ Create files]

[✓ package application into a JAR file]

[✓ generate SQL scripts for database objects]

Publish the current application

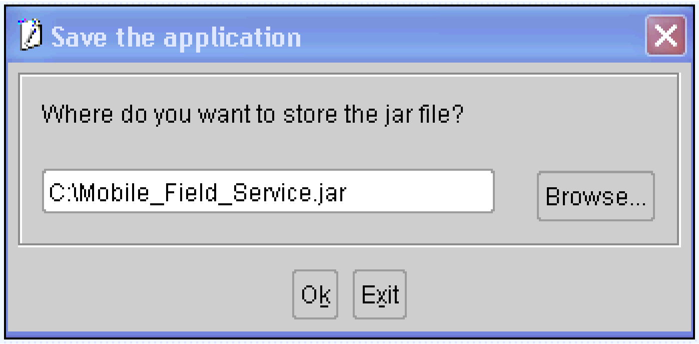
Restart wizard.

Using the Application Definition Completed panel, you can package the "Task" application into a JAR file. The Application Definition Completed Dialog remains open for you to initiate application packaging.

a. Select the **Create files** option and select both the **Package Application into a JAR file** and **Generate SQL scripts for database objects** boxes.

   At this stage, the Save the Application dialog prompts you for the name of the JAR file, which is `Mobile_Field_Service.jar`.

Figure 19–6 Save the Application Dialog



Where do you want to store the jar file?

C:\Mobile_Field_Service.jar

After choosing the JAR file, click **OK**. The JAR file is created and contains the application files and definition.

b. Back in the Application Definition Completed dialog, select the **Publish the Current Application** option and click **OK**.

The Publish the Application dialog appears. As Table 19–2 describes, enter the specified values.

Note: The Mobile Server must be up for successful publishing.

Table 19–2 Publish the Application Dialog Description

Field	Description	Value
Mobile Server URL	URL or IP Address of the machine where the Mobile Server is running.	`<Mobile Server>/webtogo`
Mobile Server User Name	User name of the Mobile Server user with administrative privileges.	Administrator
Mobile Server Password	Password of the Mobile Server user with administrative privileges.	admin
Repository Directory	Directory name where all files for this application will be stored inside the Mobile Server Repository.	`/tutorial`
Public Application	Do not select this check box unless you want to make this application available to all users.	Clear

c. To publish the application in the Mobile Server Repository, click **OK**. A dialog displays the application publishing status. You must wait until the application is published.

d. To confirm that the application is published successfully, click **OK**.

e. To exit the Packaging Wizard, click **Exit**.

You have now completed packaging and publishing your application.

19.3 Administer the Application

This section describes how to administer the application that you created and deployed through the following tasks.

- Section 19.3.1, "Start the Mobile Server and the Mobile Manager"

- Section 19.3.2, "Using the Mobile Manager to Create New Users for the Task Application"
- Section 19.3.3, "Setting Application Properties"
- Section 19.3.4, "Granting User Access to the Application"
- Section 19.3.5, "Defining Snapshot Template Values for the User"

For more information about Mobile Manager tasks described in this tutorial, see the *Oracle Database Lite Administration and Deployment Guide*.

19.3.1 Start the Mobile Server and the Mobile Manager

The Mobile Manager is a Web-based application that enables you to administer Mobile Server applications. To start the Mobile Manager, perform the following steps:

1. Using the command prompt, go to the following directory.

   ```
   <ORACLE_HOME>\Mobile\Server\bin
   ```

2. To start the Mobile Server for the first time and subsequent occasions, execute the `runmobileserver` command.
3. Start your Web browser and connect to the Mobile Server by enter the following URL:

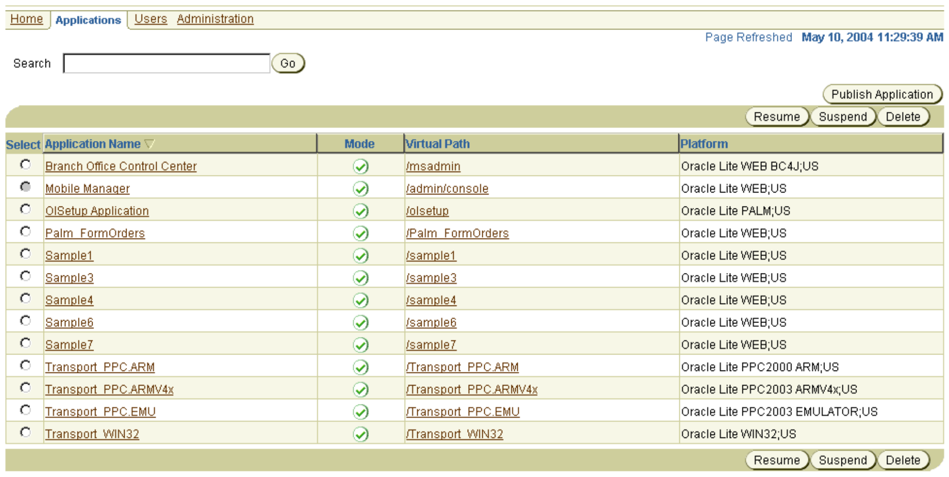
   ```
   http://<mobile_server>/webtogo
   ```

   Note: Replace `<mobile_server>` with the host name of your Mobile Server.

   ---

4. Log on as the Mobile Server Administrator using `administrator` as the User Name and `admin` as the Password.
5. To launch the Mobile Manager, click the **Mobile Manager** link in the workspace.
6. Click the Mobile Server link.
7. Click the **Applications** link. As Figure 19–7 displays, the Applications page appears. Locate the Task application, which shows all applications that are published.

Figure 19–7 Applications Page



Page Refreshed May 10, 2004 11:29:39 AM

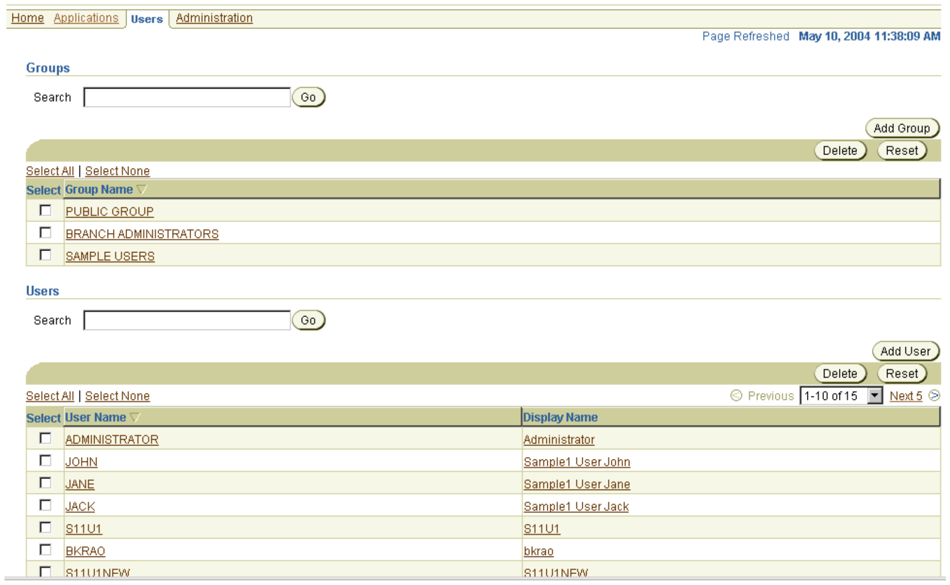
Select	Application Name	Mode	Virtual Path	Platform
○	Branch Office Control Center	✓	/msadmin	Oracle Lite WEB BC4J;US
●	Mobile Manager	✓	/admin/console	Oracle Lite WEB;US
○	OlSetup Application	✓	/olsetup	Oracle Lite PALM;US
○	Palm_FormOrders	✓	/Palm_FormOrders	Oracle Lite WEB;US
○	Sample1	✓	/sample1	Oracle Lite WEB;US
○	Sample3	✓	/sample3	Oracle Lite WEB;US
○	Sample4	✓	/sample4	Oracle Lite WEB;US
○	Sample6	✓	/sample6	Oracle Lite WEB;US
○	Sample7	✓	/sample7	Oracle Lite WEB;US
○	Transport_PPC.ARM	✓	/Transport_PPC.ARM	Oracle Lite PPC2000 ARM;US
○	Transport_PPC.ARMV4x	✓	/Transport_PPC.ARMV4x	Oracle Lite PPC2003 ARMV4x;US
○	Transport_PPC.EMU	✓	/Transport_PPC.EMU	Oracle Lite PPC2003 EMULATOR;US
○	Transport_WIN32	✓	/Transport_WIN32	Oracle Lite WIN32;US

19.3.2 Using the Mobile Manager to Create New Users for the Task Application

For the Task application, create users Tom, Dick and Harry. We only show how to create the user Tom in the following steps:

1. On the Mobile Manager home page, click the **Users** link. As Figure 19–8 displays, the Users page appears.

Figure 19–8 Users Page



Home Applications **Users** Administration

Page Refreshed **May 10, 2004 11:38:09 AM**

Groups

Search [ ] Go

Add Group

Delete Reset

Select All | Select None

Select	Group Name
☐	PUBLIC GROUP
☐	BRANCH ADMINISTRATORS
☐	SAMPLE USERS

Users

Search [ ] Go

Add User

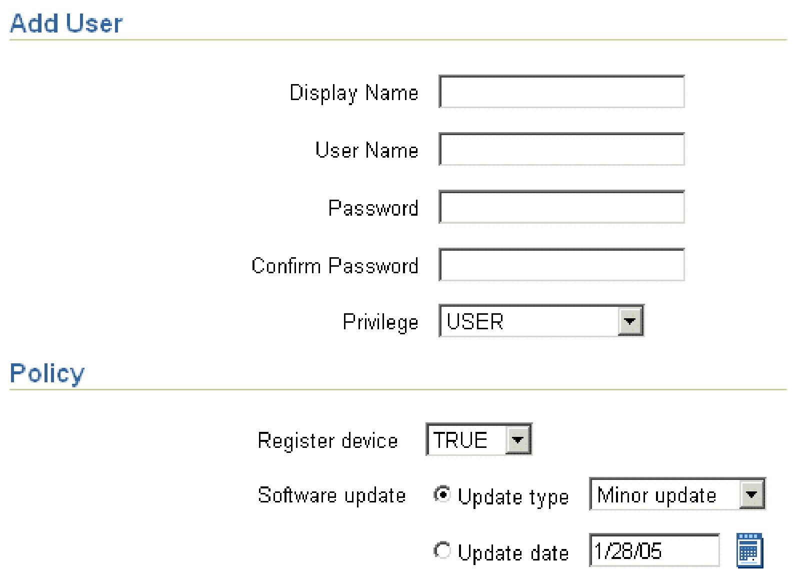
Delete Reset

Select All | Select None — Previous 1-10 of 15 Next 5

Select	User Name	Display Name
☐	ADMINISTRATOR	Administrator
☐	JOHN	Sample1 User John
☐	JANE	Sample1 User Jane
☐	JACK	Sample1 User Jack
☐	S11U1	S11U1
☐	BKRAO	bkrao
☐	S11U1NEW	S11U1NEW

2. Click **Add User**. As Figure 19–9 displays, the Add User page appears.

Figure 19–9 Add User Page



Add User

Display Name	
User Name	
Password	
Confirm Password	
Privilege	USER

Policy

Register device	TRUE	
Software update	● Update type	Minor update
	○ Update date	1/28/05

3. As described in Table 19–3, enter the following information in the Add User page and click **Save**.

Table 19–3 Add User Page Description

Field	Value
Display Name	`Tom Jones`
User Name	`Tom`
Password	`tomjones`
Password Confirm	`tomjones`
Privilege	User
Register Device	True
Software Update	Select all updates

Repeat these steps to create the Dick and Harry users.

19.3.3 Setting Application Properties

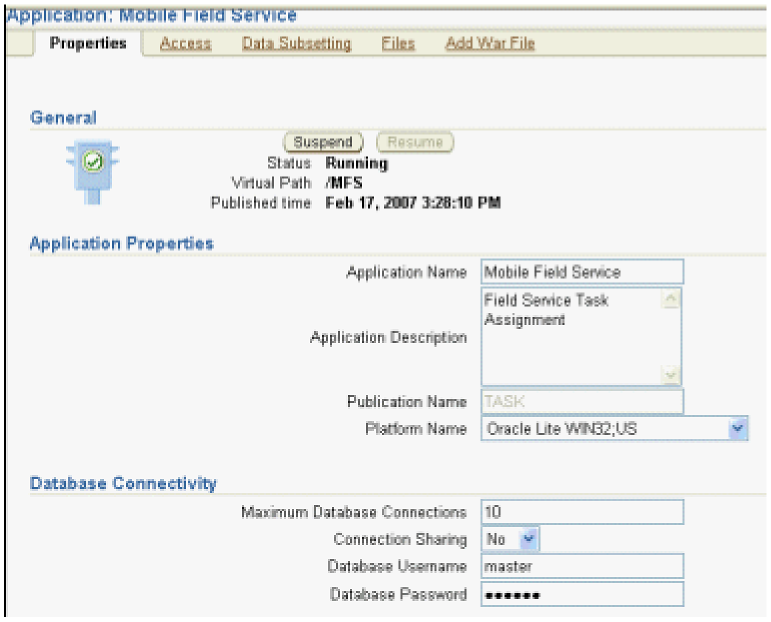
To set the Task application properties, perform the following steps:

1. On Mobile Manager home page, click the **Applications** link. The Applications page appears.
2. To search for the application that you just published, enter Task in the **Application Name** field and click **Search**. The Task application appears in the workspace.

> **Note:** To display all the available applications, leave the search field blank and click **Search**. This action generates a list of all the available Mobile Server applications in the workspace.

3. Click the **Task** application link. As Figure 19–10 displays, the Application Properties page lists application properties and database connectivity details.

Figure 19–10 Application Properties Page



Status	Running
Virtual Path	/MFS
Published time	Feb 17, 2007 3:28:10 PM
Application Name	Mobile Field Service
Application Description	Field Service Task Assignment
Publication Name	TASK
Platform Name	Oracle Lite WIN32;US
Maximum Database Connections	10
Connection Sharing	No
Database Username	master
Database Password	••••••

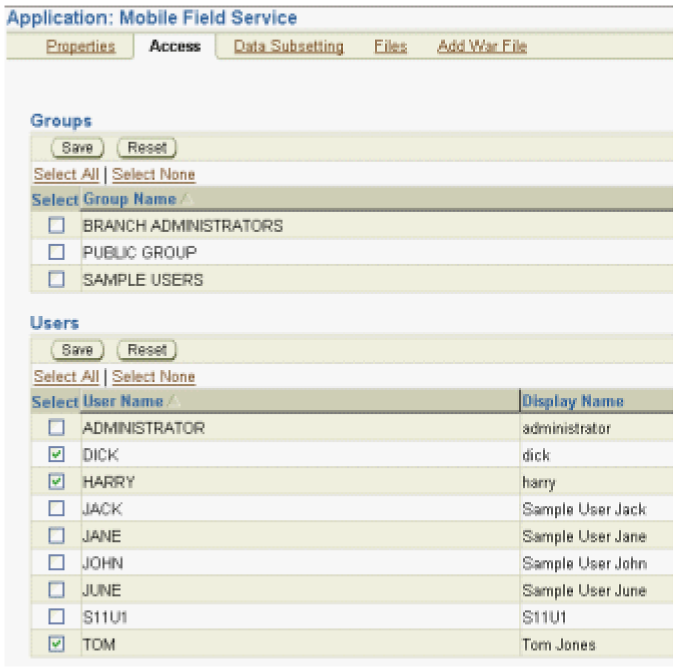
4. In the **Database Password** field, type the application demo schema password. In the past, this password was `master`. Click **Apply**. The Mobile Manager displays a confirmation message.

19.3.4 Granting User Access to the Application

To grant the Tom, Dick and Harry users access to the Task application, perform the following steps:

1. Navigate to the Application Properties page and click the **Access** link. As Figure 19–11 displays, the Access page lists groups and users that are associated with the application. The check boxes on this page indicate whether or not the user or group has access to the application.

Figure 19–11 Access Page



Select	Group Name
☐	BRANCH ADMINISTRATORS
☐	PUBLIC GROUP
☐	SAMPLE USERS

Select	User Name	Display Name
☐	ADMINISTRATOR	administrator
✓	DICK	dick
✓	HARRY	harry
☐	JACK	Sample User Jack
☐	JANE	Sample User Jane
☐	JOHN	Sample User John
☐	JUNE	Sample User June
☐	S11U1	S11U1
✓	TOM	Tom Jones

2. Under the Users table, locate the Tom, Dick and Harry users and select the check boxes for these users.

3. Click **Save**. The Mobile Manager displays a confirmation message. The users have now been granted access to the Task application.

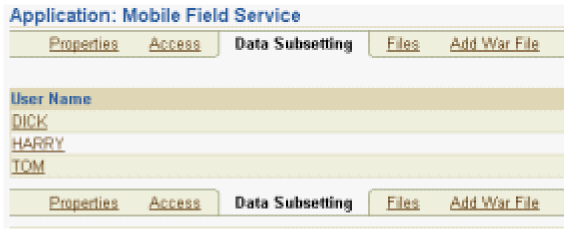
19.3.5 Defining Snapshot Template Values for the User

Define the snapshot template variables for the users, Tom, Dick and Harry. For the Mobile Field Service application, we have only one publication item and it has only one subscription parameter called `city`.

To modify a user's Data Subsetting parameters, perform the following steps:

1. Navigate to the Applications page and click the **Task** application link. The Application Properties page appears. Click the **Data Subsetting** link. As Figure 19–12 displays, the Data Subsetting page appears.

Figure 19–12 Data Subsetting Page



Application: Mobile Field Service

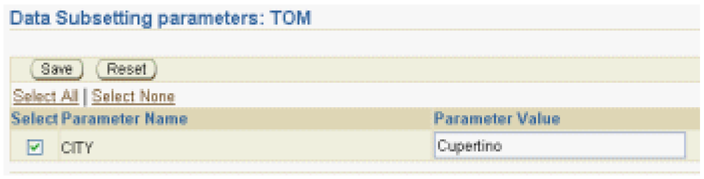
Properties | Access | Data Subsetting | Files | Add War File

User Name
DICK
HARRY
TOM

Properties | Access | Data Subsetting | Files | Add War File

2. Under the User Name column, click the user name link `Tom`. As Figure 19–13 displays, the Data Subsetting Parameters page appears.

Figure 19–13 Data Subsetting Parameters Page



Data Subsetting parameters: TOM

Select	Parameter Name	Parameter Value
✓	CITY	Cupertino

3. Select the `city` parameter and enter the value `Cupertino`. Click **Save**. The Mobile Manager displays a confirmation message. Click **OK**.

Repeat these steps for Dick and Harry. For more information about snapshots, refer to the *Oracle Database Lite Administration and Deployment Guide*.

19.4 Execute the Application on the Mobile Client for Web-to-Go

This section describes how to set up a Mobile client to use the application that you created and tested in the Development section, deployed in the Deployment section, and then administered in the Administration section.

In this section, you will perform the following tasks:

- Section 19.4.1, "Install the Mobile Client on the Win32 Device"

> **Note:** You must install the application and test it on a separate machine from the Mobile Server.

- Section 19.4.2, "Browse the TASK Snapshot and Update a Row"
- Section 19.4.3, "Develop your Mobile Field Service Application Using Oracle Database Lite"
- Section 19.4.4, "Republish the Application with the Application Program"

19.4.1 Install the Mobile Client on the Win32 Device

You must install the Mobile client before you can use the application that you created and deployed.

> **Note:** You must install the Mobile Client on a machine which does not host the Mobile Server installation.

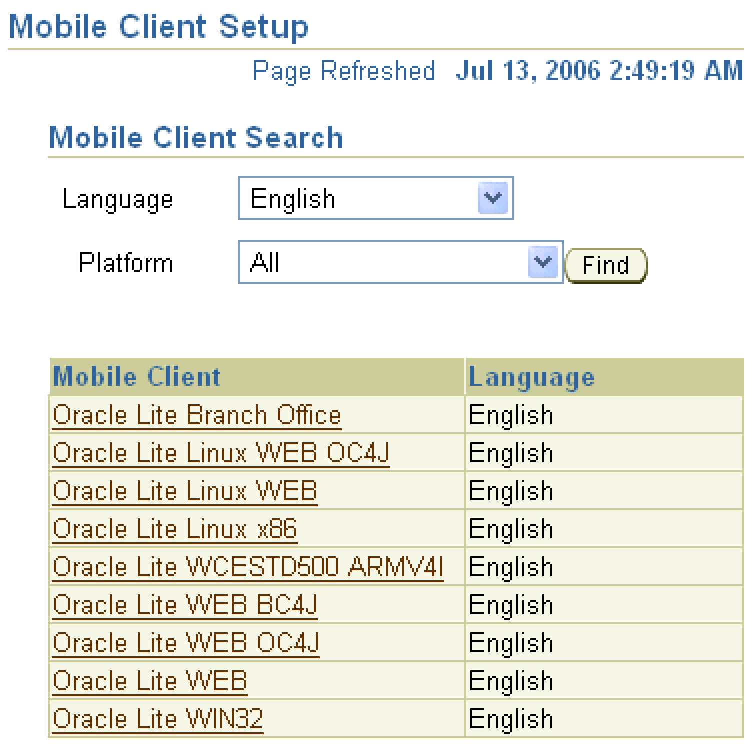
To install the Mobile Client for Win32, perform the following actions:

1. Start your Web browser and connect to the Mobile Server by entering the following URL.

   ```
   http://<mobile_server>/webtogo/setup
   ```

2. As Figure 19–14 displays, the Mobile Client Setup page lists a set of Mobile clients by platform. To download the Mobile Client for Win32 setup program, click the Oracle Lite WIN32 link.

Figure 19–14 Mobile Client Setup Page



Mobile Client Setup

Page Refreshed Jul 13, 2006 2:49:19 AM

Mobile Client Search

Language: English

Platform: All

Mobile Client	Language
Oracle Lite Branch Office	English
Oracle Lite Linux WEB OC4J	English
Oracle Lite Linux WEB	English
Oracle Lite Linux x86	English
Oracle Lite WCESTD500 ARMV4I	English
Oracle Lite WEB BC4J	English
Oracle Lite WEB OC4J	English
Oracle Lite WEB	English
Oracle Lite WIN32	English

Note: While installing the Mobile Client, you will be prompted for the User name and Password. Enter `Tom` as the user name and `tomjones` as the password.

---

3. If you are using Netscape, choose a location to save the setup program and click **OK**. In Windows Explorer, double-click `setup.exe` to run the setup program.

   If you are using Internet Explorer, run the setup program from your browser window.

4. While installing the Mobile Client, you will be prompted for the user name and password. Enter `Tom` as the user name and `tomjones` as the password.

5. The setup program prompts you to choose an installation directory for the Mobile client, such as `D:\MFS`, and downloads all the required components and starts the Mobile client on your machine. Browse the directory and familiarize yourself with its structure.

6. Perform the initial synchronization to bring down the first snapshot and create the Oracle Lite database. Start the Command Prompt and enter the following:

   ```
   D:\MFS\Mobile\bin>msync
   ```

   This executes the Mobile Sync application, downloaded as part of the application installation. You can also execute the Mobile Sync application located in the `\sdk\bin` directory.). When the dialog appears, enter the following information:

   User Name: `Tom`

   Password: `tomjones`

   Server: `mserver`

   Click the **Sync** button. A message box appears showing the progress of synchronization. When the synchronization process is complete, click the **Cancel** button on the Mobile Sync application dialog.

You now have an Oracle Database Lite database on your development machine. It contains a snapshot called `TASK` which has two rows in it; both rows have `Cupertino` for the `CustCity` column. These are the service requests by customers in Cupertino and Tom has been assigned these tasks.

The initial synchronization process also created an ODBC data source name (DSN) called `tom_mfs` (the user name followed by the underscore character followed by the database name).

19.4.2 Browse the TASK Snapshot and Update a Row

You can update a row in the Task snapshot, as follows:

```
D:>MFS\Mobile\bin>msql system/manager@jdbc:polite:tom_mfs
SQL> select * from task;
```

The following two rows are displayed.

```
SQL> update task set Notes ='Replaced the motor:$65' where ID = 1;
1 row(s) updated
SQL> commit;
commit complete
SQL> exit
```

You have successfully updated a row of the `TASK` snapshot. Perform another synchronization to upload the changes to the server.

19.4.3 Develop your Mobile Field Service Application Using Oracle Database Lite

An example ODBC program called `MFS.exe` is provided with the Mobile Development Kit in the following directory:

<*ORACLE_HOME*>`\Mobile\Sdk\samples\odbc\win32\mfs\`

The`\src` directory contains the source and the makefile for it.

This example displays the task list and prompts the user to enter the Task ID for the chosen task, before entering notes. When the user enters the Task ID value as -1, the program terminates. For any valid Task ID, the MFS application prompts the user to enter notes. Enter notes without using quotes. You can try to improve the example as required.

To republish this program to the Mobile Server, copy the `mfs.exe` file into the directory `D:\MFSDEV\Win32`.

19.4.4 Republish the Application with the Application Program

Use the Packaging Wizard to republish the application, as follows:

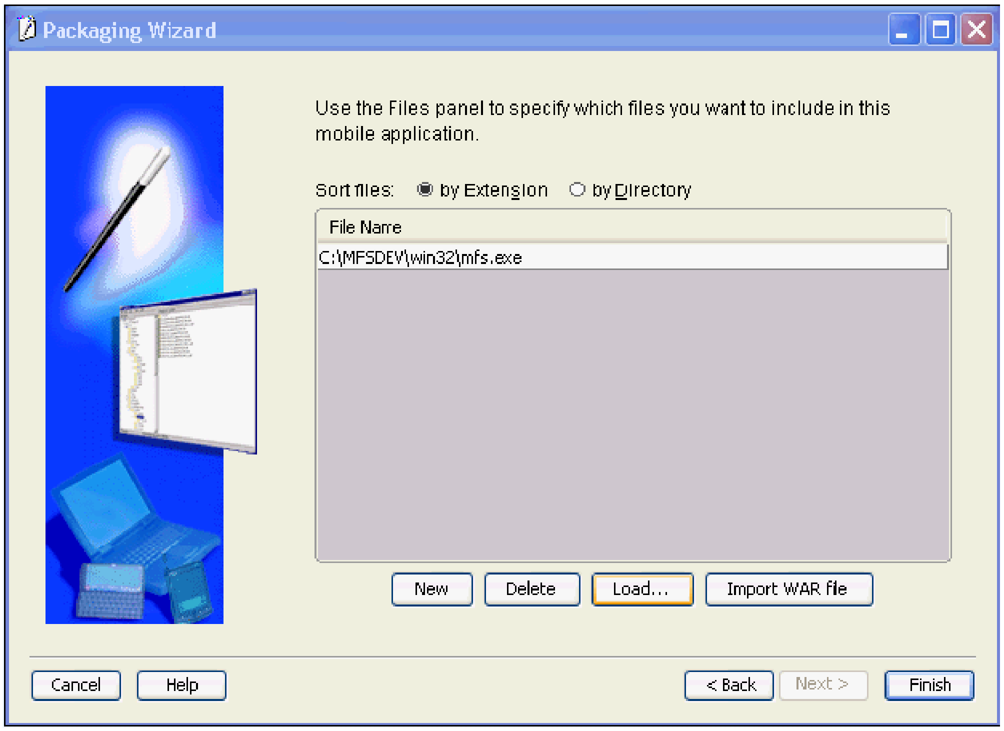
1. From the Command Line, enter the following:

   ```
   D:>runwtgpack
   ```

2. Select the "Edit an existing application" option. From the drop down list, select "Mobile Field Service" and click the **OK** button.

3. Click the **Files** tab. As shown in Figure 19–15, verify that the `mfs.exe` file is listed in the "File Name" window and click **Finish**.

Figure 19–15 Load the MFS.EXE File.



Packaging Wizard

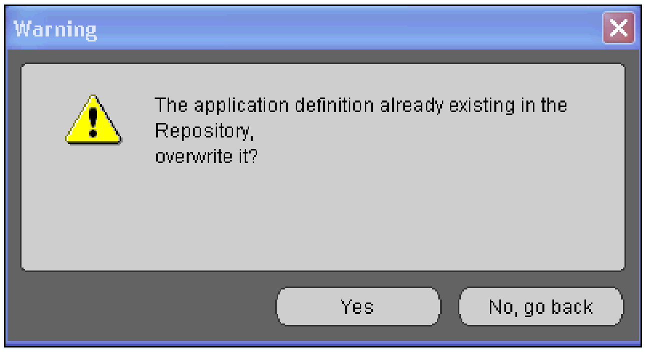
Use the Files panel to specify which files you want to include in this mobile application.

Sort files: ● by Extension ○ by Directory

File Name
C:\MFSDEV\win32\mfs.exe

4. Select the "Publish the current application" option and click **OK**. You will be prompted to enter the login information for the Mobile Server. Click **OK** after entering the information. A message box warns you that the application already exists on the Mobile Server and asks whether you want to overwrite it. Click **YES**.

Figure 19–16 Republish the Application



Warning

The application definition already existing in the Repository, overwrite it?

5. If you get the message "Application Published Successfully", click **OK** and then click **EXIT**. You have successfully republished an application that has a file called `mfs.exe` and one publication item.
6. Test your application by using a fresh Windows 32 machine. Follow Step 4 to install the Oracle Database Lite 10*g* client and the Mobile Field Service application on the machine. Then execute the Mobile Field Service application by executing the `D:\MFS\Mobile\oldb40\TOM\mfs.exe` program, as follows:

   ```
   D:\MFS\Mobile\oldb40\TOM\mfs.exe TOM_MFS system manager
   ```

7. When `TOM` is the user. Enter notes for one of the tasks. Then execute `D:\MFS\Mobile\bin\msync.exe` to synchronize your changes with the server.

20 Tutorial for Building Mobile Applications for Windows CE

You can implement Mobile applications with Oracle Database Lite for WinCE. Oracle Database Lite supports various application models for the Windows Mobile/Pocket PC device, such as ODBC, JDBC, and ADO.NET. When developing your own WinCE application, you can use Visual Studio 2005.

This chapter uses a tutorial to demonstrate how to create, deploy, administer, and use a Windows CE application. The tutorial shows a Visual Basic.NET (Visual Studio.NET) application that uses the Oracle Database Lite ADO.NET interface for Windows Mobile.

The following sections detail the development process:

- Section 20.1, "Overview of the WinCE Sample Application"
- Section 20.2, "Develop the Application"
- Section 20.3, "Create Publication for Application"
- Section 20.4, "Package and Publish the Application"
- Section 20.5, "Administer the Application"
- Section 20.6, "Run the Application on the Windows Mobile/Pocket PC Device"

20.1 Overview of the WinCE Sample Application

The sample WinCE application details typical activities of delivery personnel in the Transportation and Logistics industry, which includes package pick-up and delivery.

1. Before he leaves the dispatch center, the delivery person collects the complete delivery package list and the package delivery destination information for the day on his device.
2. As he delivers and picks-up packages, the delivery person updates the package pick-up and delivery status on his client device.
3. When he returns to the dispatch center, he synchronizes his updated information with the central server running in the dispatch center over any wireless network.

20.1.1 Before You Start

Before starting the Mobile application development process, you must ensure that the development computer and the client device meet the requirements specified below.

- Section 20.1.1.1, "Application Development Computer Requirements"

- Section 20.1.1.2, "Client Device Requirements"

20.1.1.1 Application Development Computer Requirements

Table 20–1 lists the configuration and installation requirements for the Mobile application development computer.

Table 20–1 Application Development Computer Requirements

Requirement	Description
Windows NT/2000/XP User Login	The login user on the Windows NT/2000 development computer must have "Administrator" privileges.
Installed Java Components	Java Development Kit 1.4.2 or higher.
Installed Oracle Database Lite 10*g* Components	Oracle Database 9.2 or higher. The Mobile Server (Oracle Database Lite CD-ROM). The Mobile Development Kit (Oracle Database Lite CD-ROM).
Installed Windows Mobile/Pocket PC Components	Microsoft Active Sync 3.8 or higher.

20.1.1.2 Client Device Requirements

You must connect the client device to the desktop and install the Oracle Database Lite client for Pocket PC on the device. For more information on how to install the Mobile Client on the device, see Section 20.6.1, "Install the Oracle Database Lite Mobile client for Pocket PC".

20.2 Develop the Application

This section explains how to develop and test the WinCE Transport application using the Mobile Development Kit. The WinCE Transport application is written in Visual Basic.NET (Visual Studio.NET).

To develop and test the WinCE Transport application, perform the following tasks.

1. Section 20.2.1, "Create Database Objects in the Oracle Server"
2. Section 20.2.2, "Write the Application Code"
3. Section 20.2.3, "Compile the Application"

20.2.1 Create Database Objects in the Oracle Server

During deployment, the Mobile Server automatically creates the Oracle Database Lite database in the client device along with the requisite tables and data. To publish the application, users must create the database objects used by the application in the back-end Oracle database.

20.2.1.1 The WinCE Transport Application Database Objects

The WinCE Transport application uses the following database objects:

- Packages Table
- Routes Table
- Trucks Table

Table 20–2 lists the columns for the Packages table for storing information about the package.

Table 20–2 Packages Table

Column	Description
`DID`	Package ID
`DDSC`	Package Description
`DWT`	Package Weight
`DSTR`	Destination Street
`DCTY`	Destination City
`DST`	Destination State
`DRTNR`	Route Number
`DRTNM`	Route Name
`DESN`	Signature
`DSTS`	Package Status
`TID`	Truck Number
`PRTY`	Priority
`PTNO`	Point Number
`TIND`	Delivery 'D', or Pick-up 'P'

Table 20–3 lists the columns for the Routes table for storing information about a route.

Table 20–3 Routes Table

Column	Description
`ROUTE_NO`	Route Number (Primary Key)
`ROUTE_NM`	Route Name
`EST_TIME`	Estimated Time

Table 20–4 lists columns for the Trucks table for storing information about the availability status and destination information for a truck.

Table 20–4 Trucks Table

Column	Description
`TRUCK_NO`	Truck Number (Primary Key)
`TRUCK_STATUS`	Status of the Truck
`ALERT_ADDRESS`	Mobile or Pager address to send alert to (Portal User Interface)
`DRIVER_ID`	ID of the Truck Driver

To Create Database Objects

In order to execute the Transport demo, you must set up the schema and the database objects. We have provided a SQL script that will create the `master` schema and the database objects in the back-end. However, if the `master` schema is already created, then remove the statements that create this schema from the `create.sql` script.

Note: Ensure that the `CLASSPATH` includes `ojdbc14.jar`.

Execute the `create.sql` script, as follows:

```
> cd ORACLE_HOME\Mobile\Sdk\samples\ado.net\wince\Transport\sql

> msql system/<sys_pwd>@jdbc:oracle:thin:@<host>:<port>:
      <oracle_sid> @create.sql
```

Note: While entering the above command to create database objects, you must include a mandatory space between "`<oracle_sid>`" and "`@create.sql`".

Where:

- `<sys_pwd>` is the system password. This is required if you are creating the `master` schema. However, if you have eliminated the statements that create the schema, you can use `master/master` for username/password.
- `<host>:<port>` refers to the name and listening port of the machine where the back-end Oracle database is installed.

20.2.2 Write the Application Code

The WinCE Transport application, located in cd *`ORACLE_HOME`*`\Mobile\Sdk\samples\ado.net\wince\Transport`, uses Visual Basic.NET (Visual Studio.NET), which is available with the sample application. The following sections describe the Transport application code:

- Section 20.2.2.1, "Transport Module (Transport.vb)"
- Section 20.2.2.2, "Main Form (frmMain.vb)"
- Section 20.2.2.3, "View Packages (frmView.vb)"
- Section 20.2.2.4, "Create Package (frmNew.vb)"

20.2.2.1 Transport Module (Transport.vb)

To open a database connection, you must declare a connection object,. which—in this tutorial—is called `conn`. The scope of the connection object is project level. The `Connect` sub-routine in the `transport.vb` module establishes a connection to the local Oracle Lite database with the DSN `transport`; the `Disconnect` sub-routine releases the connection.

Within the `Connect` sub-routine, the DSN is initialized as follows:

```
Dim dsn As String = "dsn=transport;uid=system;pwd=" & pwd
conn = New Oracle.DataAccess.Lite.OracleConnection(dsn)
conn.Open()
```

The DSN username and password are `system` and the user password; thus, only the user can connect since the user password is used.

20.2.2.2 Main Form (frmMain.vb)

The `frmMain.vb` file implements the main form of the Transport Tutorial application. This form connects to Oracle Database Lite on `Load` time and invokes the `Create Package` and `View Packages` forms, using the appropriate command buttons.

If the synchronization button is pushed, notice that the following is executed:

```
Disconnect()
OracleEngine.Synchronize(True)
Connect(UserName, Password)
```

In order to retrieve information from the database, the connection was established at the start of the application. Since you can only have a single connection to the back-end database—and the `OracleEngine.Synchronization` method creates a connection to the database as part of its functionality—the original connection is disconnected before the synchronization is invoked. Once synchronization is complete, the original connection is re-established. See Section 13.1.6.2, "Using the OracleEngine to Synchronize" for more information on this class.

20.2.2.3 View Packages (frmView.vb)

This form displays existing packages from the database. It also allows the user to modify and save existing packages. This form demonstrates the usage of the `OracleDataAdapter` and `DataSet` classes.

Note: The `OracleDataAdapter` is the same as the Microsoft ADO.Net `DataAdapter` class. For more information on `DataAdapter` and `DataSet` classes, see the Microsoft ADO.Net documentation.

When this form is loaded, it creates an instance of the `OracleDataAdapter` object and sets the appropriate `OracleCommand` objects namely, `Select`, `Update`, and `Delete`. These `OracleCommand` objects are created by the `transport.vb` module during the main form loading process. Once an `OracleAdapter` object has been created successfully, this form creates a `Dataset` object and populates it with data from Oracle Database Lite, using the `OracleDataAdapter` object that was created.

Note: For more information on the `OracleCommand` class, see Section 13.1.3, "Create Commands With the OracleCommand Class".

```
dba = New OracleDataAdapter
dba.SelectCommand = cmdSel
dba.DeleteCommand = cmdDel
dba.UpdateCommand = cmdUpd

' Fill dataset
'
dset = New DataSet
dba.Fill(dset)
```

Once the `Dataset` is filled with Oracle Database Lite data, this form populates the UI controls using data from the `DataSet` object.

```
Dim table As DataTable = dset.Tables(0)
Dim rows As DataRowCollection = table.Rows
Dim row As DataRow = rows.Item(index)
```

```
Me.packDesc.Text = row.Item(1).ToString()
Me.packWeight.Text = row.Item(2).ToString()
Me.packStreet.Text = row.Item(3).ToString()
Me.packCity.Text = row.Item(4).ToString()
Me.packState.Text = row.Item(5).ToString()
Me.packRoute.Text = row.Item(7).ToString()
```

When users make changes to the package data, this form uses the `OracleAdapter Update` method to save the changes to Oracle Database Lite.

```
Dim row As DataRow = table.Rows(index)
row.BeginEdit()
row(6) = Me.packPriority.SelectedItem.ToString()
row(8) = Me.packStatus.SelectedItem.ToString()
row.EndEdit()
dba.Update(table)
```

20.2.2.4 Create Package (frmNew.vb)

This form allows users to create a new package entry in Oracle Database Lite. During the `Load` duration, this form creates a unique Package ID and populates the drop down list controls with truck numbers and route names.

When the user saves this form, it uses the `OracleCommand` and `OracleParameter` classes to save user changes in Oracle Database Lite.

Note: For more information on the `OracleCommand` class, see Section 13.1.3, "Create Commands With the OracleCommand Class".

```
cmd = GetConnection().CreateCommand()
rts = Me.packRoute.SelectedItem.ToString()

' Obtain route number
'
cmd.CommandText = "SELECT ROUTE_NO FROM ROUTES where ROUTE_NM='" & rts & "'"
res = cmd.ExecuteReader()
 While res.Read() = True
 rtn = res.GetString(0)
 End While
res.Close()

cmd.CommandText = "INSERT INTO PACKAGES (
 (DID,DDSC,DWT,DSTR,DCTY,DST,DRTNR,DRTNM,DSTS,TID,PRTY,PTNO,TIND) values
 (?,?,?,?,?,?,?,?,'NEW',?,?,'1','P')"

' Set DID
'
par = cmd.CreateParameter()
par.DbType = DbType.String
par.Direction = ParameterDirection.Input
par.Value = id
cmd.Parameters.Add(par)

 ' Set DDSC
 '
par = cmd.CreateParameter()
par.DbType = DbType.String
par.Direction = ParameterDirection.Input
par.Value = Me.packDesc.Text
cmd.Parameters.Add(par)
...
cmd.ExecuteNonQuery()
cmd.Dispose()
```

20.2.3 Compile the Application

To install the application on the device, you must create a CAB file. The CAB file is uploaded into the Mobile Server Repository during the application's publish phase. You can create a CAB file using the Visual Basic.NET (Visual Studio.NET).

20.2.3.1 Create CAB Files

To create the CAB file for this demo, perform the following:

1. Start the Visual Studio.NET and click on **File->Select Open**
2. Browse for the `Transport.sln` file, which is located in the *`ORACLE_HOME`*`\Mobile\SDK\samples\ado.net\wince\Transport` directory. Ignore the warning message, "`The .NET assembly 'Oracle.DataAccess.Lite.dll' could not be found.`"
3. Right click on **References**.
4. Select **Add Reference**.
5. Click **Browse** and choose `Oracle.DataAccess.Lite.dll` from the *`ORACLE_HOME`*`\Mobile\SDK\ado.net\wince\v1.x` or `v2.x` directory.
6. In the 'Solution Configuration' list box, select **Release** instead of **Debug**.
7. Click **Build->Build CAB File**, which will build the CAB file for you.

20.2.3.2 Install the Application from the CAB File

You can download and install the application on the device after packaging and publishing the application. See Section 20.4, "Package and Publish the Application" for directions on how to package and publish the application.

20.3 Create Publication for Application

As described fully in Chapter 5, "Using Mobile Database Workbench to Create Publications", you can use MDW to create your publication. Launch MDW by executing `oramdw` from *`$ORACLE_HOME`*`/Mobile/Sdk/bin`. The following sections detail how to use MDW to create a publication for the application in this tutorial.

Note: While creating this publication, use Chapter 5, "Using Mobile Database Workbench to Create Publications" heavily for a deeper understanding of how to use MDW and the type of information that you must enter.

- Section 20.3.1, "Create a Project"
- Section 20.3.2, "Create Publication Items"
- Section 20.3.3, "Create Publication"

20.3.1 Create a Project

Create a new project for this application by selecting **File->New->Project**. This brings up a wizard where you enter the following information:

Note: For more information, see Section 5.2, "Create a Project".

---

1. Define a name and location for the project.
2. Enter the username, password, JDBC driver type, database host, database port and database SID for the Mobile repository.

   Provide the Mobile Repository access information. Because you are interacting with the repository to create and manipulate synchronization objects, including the SQL scripts for the publication items, you need access to the Mobile Repository.
3. Specify schema username and password. Enter the user and password of the schema owner for the schema that you are using for the Mobile application. The Mobile application schema contains all database tables, views, synonyms used to build the snapshots for the application.
4. Verify the information that you entered and click **Finish**.

20.3.2 Create Publication Items

For this project, you need to create three publication items for packages, routes, and trucks. Start the new publication item wizard by selecting **File->New->Publication Item**.

Note: For more information, see Section 5.4, "Create a Publication Item".

20.3.2.1 Create Packages Publication Item

1. Enter the name as `packages` and the type as `Fast`.
2. Select the schema name as `MASTER`, the object type as `Table`, and leave the object filter blank. Click **Search**. When the search ends, select `Packages` from the object list.
3. Click '>>' to select all of the columns in the `Packages` table.
4. In the Query tab, select **Edit** if you want to edit the query.
5. Click **Run** to test.
6. Verify and click **Finish**.

20.3.2.2 Create Routes Publication Item

1. Enter the name as `routes` and the type as `Fast`.
2. Select the schema name as `MASTER`, the object type as `Table`, and leave the object filter blank. Click **Search**. When the search ends, select `Routes` from the object list.
3. Click '>>' to select all of the columns in the `Routes` table.
4. In the Query tab, select **Edit** if you want to edit the query.
5. Click **Run** to test.

6. Verify and click **Finish**.

20.3.2.3 Create Trucks Publication Item

1. Enter the name as `trucks` and the type as `Fast`.
2. Select the schema name as `MASTER`, the object type as `Table`, and leave the object filter blank. Click **Search**. When the search ends, select `Trucks` from the object list.
3. Click '>>' to select all of the columns in the `Trucks` table.
4. In the Query tab, select **Edit** if you want to edit the query.
5. Click **Run** to test.
6. Verify and click **Finish**.

20.3.3 Create Publication

When you have completed the creation of the publication items, create the publication within the project by selecting **File->New->Publication**.

1. In the General tab, enter the name as `transport`, which is the DSN for the client-side database.
2. In the Publication Item tab, add the three publication items that you just created with the following configuration:

```
Name: PACKAGES
Updatability: Updatable
Conflict Resolution: Server Wins
DML Callback: BLANK
Grouping Function: BLANK
Priority Condition: BLANK
My Compose: BLANK
Weight: 1
Description: Blank

Name: ROUTES
Updatability: Read Only
Conflict Resolution: Custom
DML Callback: BLANK
Grouping Function: BLANK
Priority Condition: BLANK
My Compose: BLANK
Weight: 2
Description: Blank

Name: TRUCKS
Updatability: Read Only
Conflict Resolution: Custom
DML Callback: BLANK
Grouping Function: BLANK
Priority Condition: BLANK
My Compose: BLANK
Weight: 3
Description: Blank
```

3. Save the publication by selecting **File->Save**.

20.4 Package and Publish the Application

The following sections describe how to package the application and prepare it for publishing into the Mobile Server:

1. Section 20.4.1, "Define the Application Using the Packaging Wizard"
2. Section 20.4.2, "Publish the Application"

20.4.1 Define the Application Using the Packaging Wizard

Using the Packaging Wizard, you can select and describe the Transport application.

20.4.1.1 Create a New Application

Using the Mobile Server Packaging Wizard, you can publish the WinCE application into the Mobile Server. For more information on how to use the Packaging Wizard, see Chapter 7, "Using the Packaging Wizard".

You can select and describe the WinCE Transport application by launching the Packaging Wizard in regular mode.

To launch the Packaging Wizard in regular mode, perform the following steps.

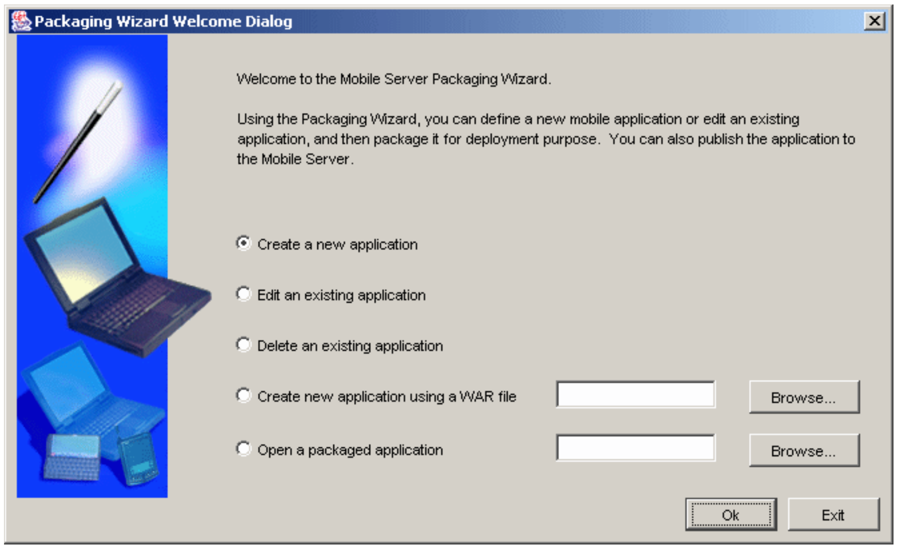
1. Using the Command Prompt, enter the following.

   ```
   cd ORACLE_HOME\Mobile\SDK\bin
   ```

   ```
   wtgpack
   ```

   As Figure 20–1 displays, the Packaging Wizard displays the Welcome panel. Select the **Create a new application** option and click **OK**.

Figure 20–1 Welcome Dialog



Packaging Wizard Welcome Dialog

Welcome to the Mobile Server Packaging Wizard.

Using the Packaging Wizard, you can define a new mobile application or edit an existing application, and then package it for deployment purpose. You can also publish the application to the Mobile Server.

[✓ Create a new application]
Edit an existing application
Delete an existing application
Create new application using a WAR file
Open a packaged application

2. The Select Platforms panel appears. Choose '**Oracle Lite PPC50 ARMV4I;US**' from the list displayed and click **Next**.
3. The Application panel appears. As Table 20–5 describes, enter the WinCE Transport application settings. Figure 20–2 displays the Applications panel.

Figure 20–2 Applications Panel

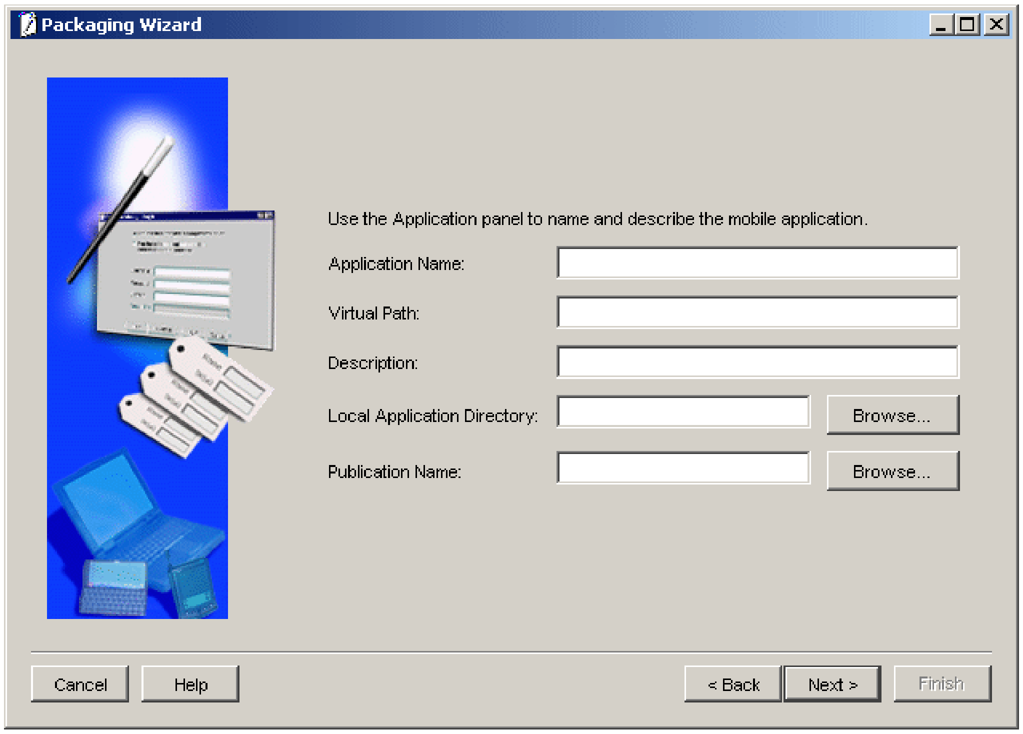
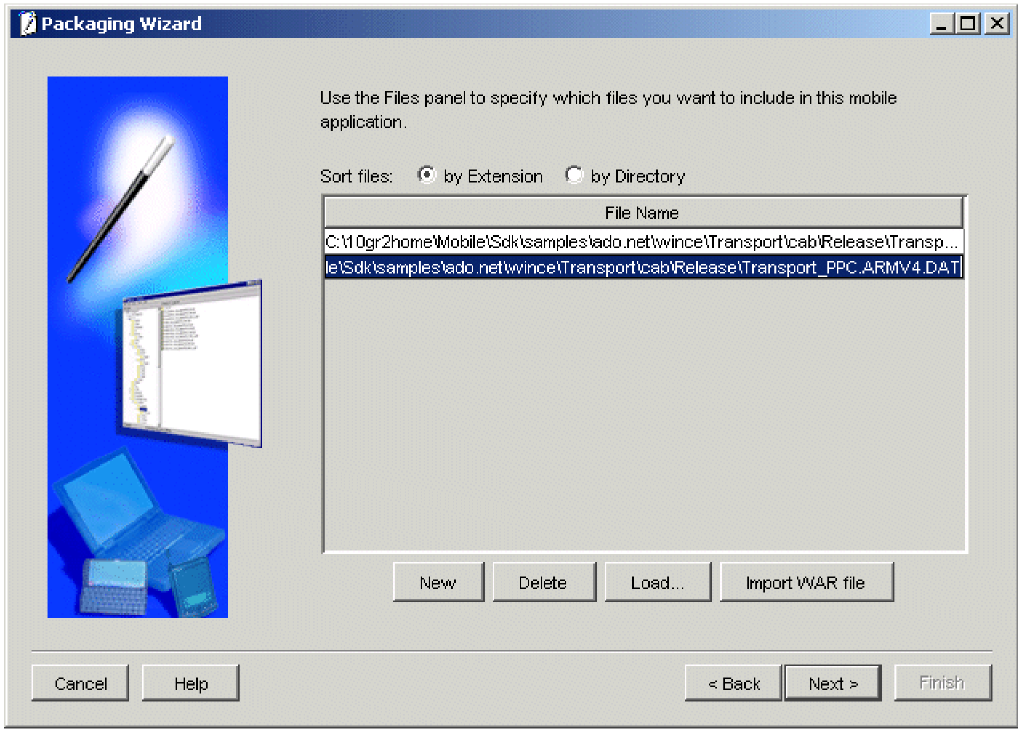


Table 20–5 The WinCE Transport Application Settings

Field	**Value**
Application Name	Transport
Virtual Path	`/Transport`
Description	Transport and Logistics Management
Local Application Directory	`<`*`ORACLE_HOME`*`>\Mobile\Sdk\samples\ado.net\wince\Transport\cab\Release`
Publication Name	Select **Browse** to locate the publication that was created by MDW, named `transport`. This pops up a "Publication Name" screen where you can select the publication and click **Add**.

4. Click **Next**. As Figure 20–3 displays, the Files panel appears.

Figure 20–3 Files Panel



Use the Files panel to specify which files you want to include in this mobile application.

C:\10gr2home\Mobile\Sdk\samples\ado.net\wince\Transport\cab\Release\Transp...
le\Sdk\samples\ado.net\wince\Transport\cab\Release\Transport_PPC.ARMV4.DAT

The Files panel automatically lists all files that reside in the directory, based on the 'Local Application Directory' specified in the previous Application panel. Ensure that you select the correct CAB file.

For example, in this tutorial, you must select the `Transport_PPC.ARMV4.CAB` and `Transport_PPC.ARMV4.DAT`, because your target device is Pocket PC with the ARM chipset. If other `.CAB` and `.DAT` files are in this listing, then use the Delete button in the Files panel to delete these files from the list.

After selecting the appropriate CAB file, you must define the application connection details to the Oracle Lite database.

On the Files panel, click **Next**.

20.4.2 Publish the Application

Using the Application Definition Completed dialog, you can package and publish the WinCE Transport application.

To publish the Transport application, perform the following steps.

1. In the Application Definition Completed dialog, select the **Publish the Current Application** option and click **OK**.
2. The Publish the Application dialog appears. As Table 20–6 describes, enter the specified values.

Table 20–6 Publish the Application Dialog Description

Field	Description	Value
Mobile Server URL	URL or IP Address of the machine where the Mobile Server is running.	`<Mobile Server>/webtogo`

Table 20–6 (Cont.) Publish the Application Dialog Description

Field	Description	Value
Mobile Server User Name	User name of the Mobile Server user with administrative privileges.	Administrator
Mobile Server Password	Password of the Mobile Server user with administrative privileges.	admin
Repository Directory	Directory name where all files for this application will be stored inside the Mobile Server Repository.	`/transport`
Public Application	Do not select this check box unless you want to make this application available to all users.	Clear

3. To publish the application in the Mobile Server Repository, click **OK**. A dialog displays the application's publishing status. You must wait until the application is published.
4. To confirm that the application is published successfully, click **OK**.
5. To exit the Packaging Wizard, click **Exit**.

At this stage, you have completed all the development tasks required for packaging or publishing the application.

20.5 Administer the Application

This section describes how to administer the Mobile application published by you into the Mobile Server. To administer the application, perform the following tasks.

1. Section 20.5.1, "Start the Mobile Server"
2. Section 20.5.2, "Launch the Mobile Manager"
3. Section 20.5.3, "Create a New User"
4. Section 20.5.4, "Set the Application Properties"
5. Section 20.5.5, "Grant User Access to the Application"

For more information on the Mobile Manager see the *Oracle Database Lite Administration and Deployment Guide*.

20.5.1 Start the Mobile Server

To start the Mobile Server in standalone mode, enter the following command using the Command Prompt.

```
> runmobileserver
```

20.5.2 Launch the Mobile Manager

Using the login user name and password, you can log in to the Mobile Server and launch the Mobile Manager.

To start the Mobile Manager, perform the following steps.

1. Open your Web browser and connect to the Mobile Server by entering the following URL.

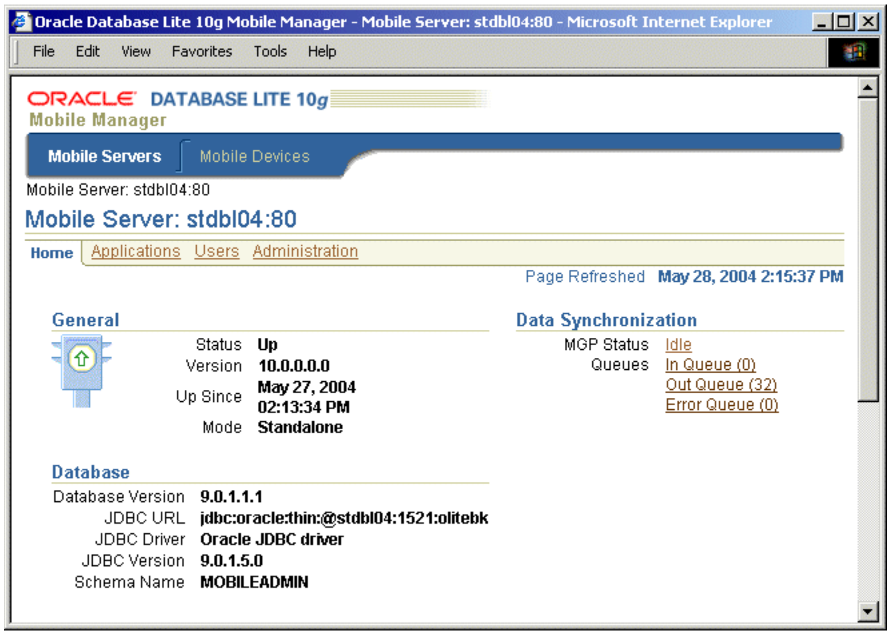
   ```
   http://<mobile_server>/webtogo
   ```

Note: You must replace the `<mobile_server>` variable with your Mobile Server's host name.

2. Log in as the Mobile Server administrator using `administrator` as the User Name and `admin` as the Password.
3. To launch the Mobile Manager, click the Mobile Manager link in the workspace. The Mobile Server farms page appears. To display your Mobile Server's home page, click your Mobile Server link.

   Figure 20–4 displays the Mobile Server home page.

Figure 20–4 Mobile Server Home Page



Mobile Server: stdbl04:80

Page Refreshed May 28, 2004 2:15:37 PM

Status	Up
Version	10.0.0.0.0
Up Since	May 27, 2004 02:13:34 PM
Mode	Standalone

MGP Status	Idle
Queues	In Queue (0)
	Out Queue (32)
	Error Queue (0)

Database Version	9.0.1.1.1
JDBC URL	jdbc:oracle:thin:@stdbl04:1521:olitebk
JDBC Driver	Oracle JDBC driver
JDBC Version	9.0.1.5.0
Schema Name	MOBILEADMIN

20.5.3 Create a New User

To create a new Mobile Server user, perform the following steps.

1. In the Mobile Manager, click the **Users** tab.
2. Click **Add User**.
3. Enter data as described in Table 20–7.
4. Click **Save**. The Mobile Manager displays a confirmation message.
5. Click **OK**.

Table 20–7 lists the values that you must enter in the **Add User** page.

Table 20–7 The Add User Page Description

Field	Value
Display Name	bob
User Name	bob

Table 20–7 (Cont.) The Add User Page Description

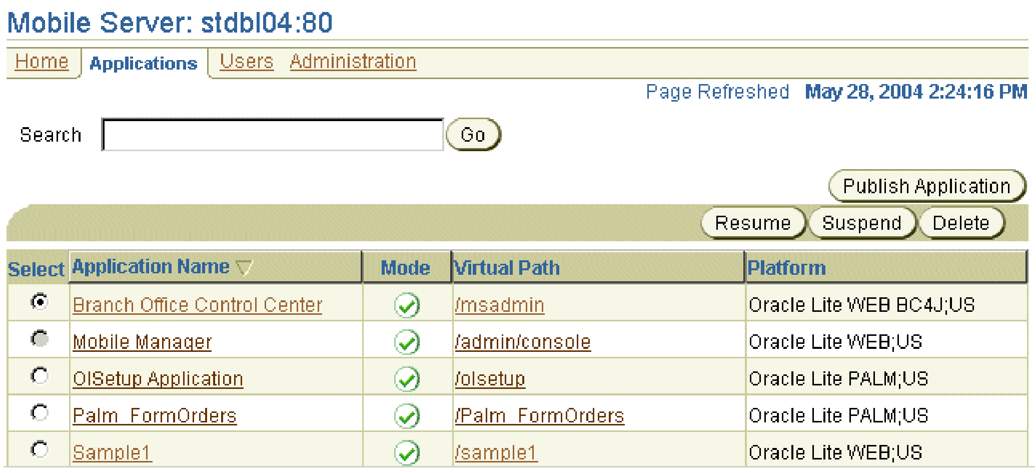
Field	Value
Password	bobhope
Password Confirm	Re-enter the password for confirmation
System Privilege	Select the "User" option

20.5.4 Set the Application Properties

To set the WinCE Transport Application properties, perform the following steps.

1. In the Mobile Manager, click the **Applications** tab. As Figure 20–5 displays, The **Applications** page appears. You can search the list of available applications by application name.

Figure 20–5 Applications Page



Select	Application Name	Mode	Virtual Path	Platform
●	Branch Office Control Center	✓	/msadmin	Oracle Lite WEB BC4J;US
○	Mobile Manager	✓	/admin/console	Oracle Lite WEB;US
○	OlSetup Application	✓	/olsetup	Oracle Lite PALM;US
○	Palm_FormOrders	✓	/Palm_FormOrders	Oracle Lite PALM;US
○	Sample1	✓	/sample1	Oracle Lite WEB;US

2. Click **Transport**. The Transport application page appears. It displays an application's properties and database connectivity details.

3. In the **Platform Name**, select **Oracle Lite PPC50 ARMV4I; US**. In the **Database User** field, enter `master` for the `master` schema. In the **Database Password** field, enter `master`. This is the default password for the `master` user schema of the Oracle Server Database.

4. Click **Apply**.

20.5.5 Grant User Access to the Application

To grant user access to the Transport application, perform the following steps.

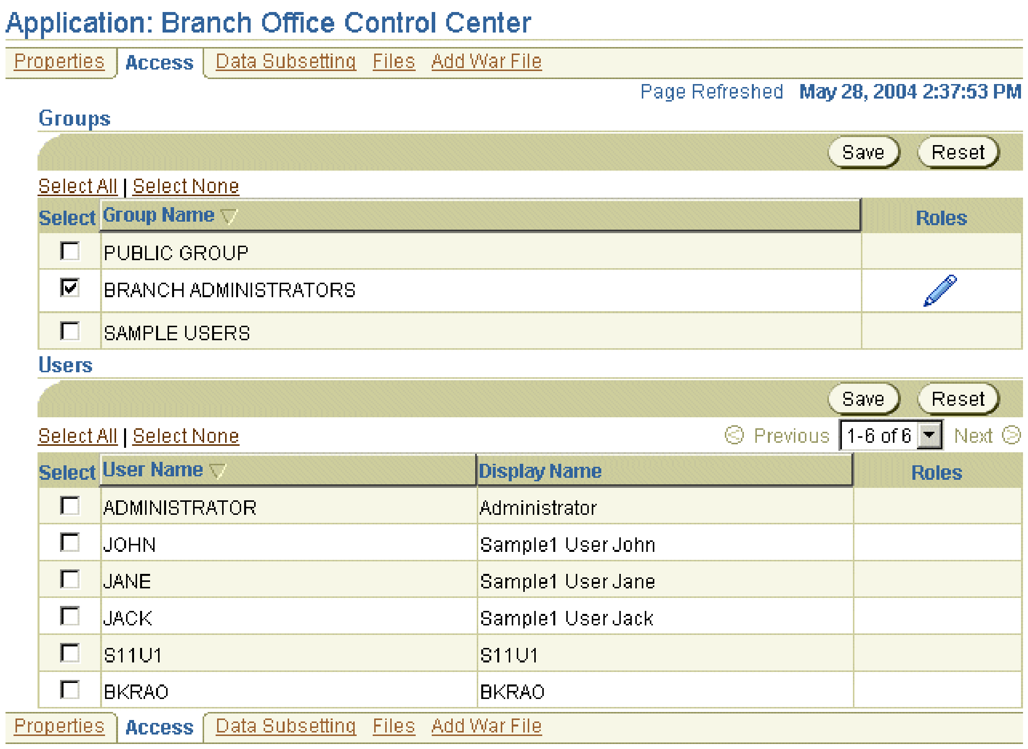
1. In the Transport application page, click the **Access** link. As Figure 20–6 displays, the Access page lists application users and application groups. To grant access to a user or a group of users to the Transport application, select the corresponding boxes.

   For example, to provide access to a user named BOB, locate the user name "BOB" in the **Users** list and select the corresponding box.

2. Click **Save**. The user "BOB" is granted access to the Transport application.

   Figure 20–6 displays the Access page of the Transport application.

Figure 20–6 Access Page



Page Refreshed May 28, 2004 2:37:53 PM

Groups

Select	Group Name	Roles
☐	PUBLIC GROUP	
☑	BRANCH ADMINISTRATORS	✎
☐	SAMPLE USERS	

Users

Select	User Name	Display Name	Roles
☐	ADMINISTRATOR	Administrator	
☐	JOHN	Sample1 User John	
☐	JANE	Sample1 User Jane	
☐	JACK	Sample1 User Jack	
☐	S11U1	S11U1	
☐	BKRAO	BKRAO	

20.6 Run the Application on the Windows Mobile/Pocket PC Device

The following sections describe how to run the application after creating, testing, deploying, and administering the application:

1. Section 20.6.1, "Install the Oracle Database Lite Mobile client for Pocket PC"
2. Section 20.6.2, "Install and Synchronize the Transport Application and Data"

20.6.1 Install the Oracle Database Lite Mobile client for Pocket PC

To install the Oracle Database Lite Mobile client for Pocket PC, perform the following actions.

1. Open your desktop browser and enter the following URL to connect to the Mobile Server.

   ```
   http://<mobile_server>/webtogo/setup
   ```

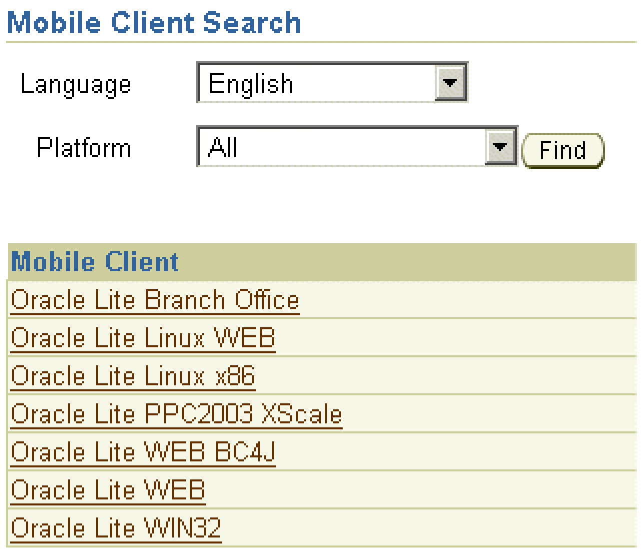
   Note: You must replace the `<mobile_server>` variable with the host name or IP address of your Mobile Server.

   A Web page appears displaying links to various Oracle Database Lite Mobile clients with different platforms. You can filter the selection by Language and Platform.

2. Click the hyperlink **Oracle Lite PPC50 ARMV4I;US** to access the setup program for the Pocket PC device with the ARM chipset.

   Figure 20–7 displays the Mobile Client Setup page.

Figure 20–7 Mobile Client Setup Page



Mobile Client
Oracle Lite Branch Office
Oracle Lite Linux WEB
Oracle Lite Linux x86
Oracle Lite PPC2003 XScale
Oracle Lite WEB BC4J
Oracle Lite WEB
Oracle Lite WIN32

3. If you are using Netscape as your browser, choose a location on your desktop to save the setup program and click **OK**. Open the Windows Explorer program and locate the "`setup.exe`". To run the setup program, double-click "`setup.exe`".

   If you are using Internet Explorer, run the "`setup`" program from your browser window. Once started, the setup program asks you to provide the user name and password to log on to the Mobile Server. Enter **BOB** as the User Name and **bobhope** for the Password. Click **OK**.

4. The setup program asks you to provide an install directory. Enter the directory where you want to install the client, such as `C:\mobileclient\`. Click **OK**. To confirm your install directory, click **Yes**.

5. The setup program automatically downloads all the required components to the specified destination on your desktop computer.

6. Assume that you have a Pocket PC device attached to your desktop computer and are connected with Microsoft ActiveSync. The installation for your Pocket PC device starts automatically.

7. Click **Yes** to confirm installing **Oracle Lite PPC ARM; US** to the default application directory. The application installation starts on the device. Once completed, the **Mobile Client for Pocket PC** is installed on your device under the `\ORACE` directory.

20.6.2 Install and Synchronize the Transport Application and Data

To install the Transport application and data, perform the following steps.

1. On the device, locate and tap the `msync` application icon in the programs group.

2. The msync dialog appears. To download the Transport application and snapshots for user BOB, enter data as described in Table 20–8.

Table 20–8 Values You Must Enter in the msync Dialog

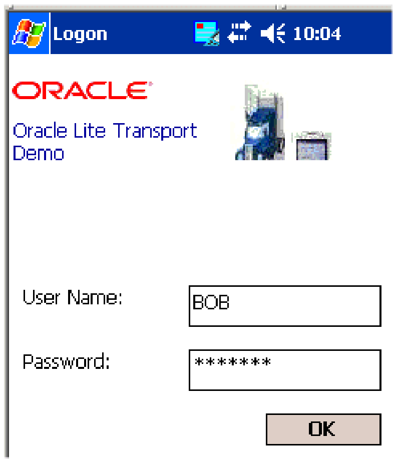
Name	Value
UserName	bob

Table 20–8 (Cont.) Values You Must Enter in the msync Dialog

Name	Value
Password	bobhope (all lowercase)
Save password box	Select
Server	Machine name or IP address

Figure 20–8 displays the **msync** dialog on the Pocket PC.

Figure 20–8 Running msync on Pocket PC



User Name: BOB

Password: *******

3. To save these values, click **Apply**.
4. To synchronize your application and data to the device, click **Sync**. If you receive an error message for invalid username/password, re-enter the clear text password in the login window.

> **Note:** Ensure that the device is connected to the desktop or the network and that the Mobile Server is running.

5. Once the synchronization is complete, click **Exit**. The Update window appears.

> **Note:** After the synchronization process is complete, a `transport.odb` file is created under the `\OraCE` directory.

6. Click **Install** to install the application. Click **Exit**.
7. Using the Start menu on the device, locate the Transport application in the Programs menu.
8. To run the Transport application, click the **Transport** icon.

A
Oracle Lite Database Utilities

This appendix describes how to use the following Oracle Lite database utilities for the Windows 32 and Windows CE platforms. Table A–1 lists all of the utility names:.

Table A–1 Database Tools and Utilities

Utility	Description
Section A.1, "The mSQL Tool"	Allows users to execute SQL statements against the Oracle Lite database.
Section A.2, "CREATEDB"	Use this to create your Oracle Lite database.
Section A.3, "REMOVEDB"	Use this to remove your Oracle Lite database.
Section A.4, "ENCRYPDB"	Use this to encrypt your Oracle Lite database.
Section A.5, "DECRYPDB"	Use this to decrypt your Oracle Lite database.
Section A.6, "BACKUPDB"	Use this to backup your Oracle Lite database.
Section A.7, "ODBC Administrator and the Oracle Database Lite ODBC Driver"	Use this to manage ODBC connections by creating data source names (DSNs) that associate the Oracle Database Lite ODBC Driver with the Oracle Database Lite that you want to access through the driver.
Section A.8, "ODBINFO"	Use this utility to find out the version number and volume ID of an Oracle Database Lite database.
Section A.9, "VALIDATEDB"	Use this to validate the structure of an Oracle Lite database and find any corruption of the database.
Section A.10, "Transferring Data Between a Database and an External File"	Use either the command-line tool or programmatic APIs to load data from an external file into a table in Oracle Database Lite, or to unload (dump) data from a table in Oracle Database Lite to an external file.
Support for Linguistic Sort	Allows databases to be created with linguistic sort capability enabled. See Section 2.10, "Support for Linguistic Sort" for more information.
dropjava	This is a command-line utility you can use to remove Java classes from Oracle Database Lite. For more information, see Section 11.3.1.4.1, "Using dropjava".
loadjava	This is a command-line utility you can use to load a Java class into Oracle Database Lite. For more information, see Section 11.3.1.1.1, "loadjava".

A.1 The mSQL Tool

Mobile SQL (mSQL) is a GUI-based application that runs on the client device (laptop and Windows CE). It allows the user to execute SQL statements against the local database. It is a development tool that enables users to execute SQL statements to the Oracle Lite database.

Using mSQL you can accomplish the following:

- Create databases
- View tables
- Execute SQL statements

Note: UTF8 SQL Scripts are not supported in mSQL.

The following sections describe how to use the mSQL tool on two platforms:

- Section A.1.1, "The mSQL Tool for Windows 32"
- Section A.1.2, "The mSQL Tool for Windows CE"

A.1.1 The mSQL Tool for Windows 32

On Windows 32 platform, the mSQL tool accesses the database through JDBC. The following sections describe how to use the mSQL command-line to access the database for the Windows 32 platform:

- Section A.1.1.1, "Starting mSQL"
- Section A.1.1.2, "Populating your Database Using mSQL"
- Section A.1.1.3, "SET TERM {ON | OFF}"
- Section A.1.1.4, "SET TIMING {ON | OFF}"
- Section A.1.1.5, "SET VERIFY {ON | OFF}"

A.1.1.1 Starting mSQL

Start mSQL by opening the `ORACLE_HOME\Mobile\SDK\Bin` directory and double-click the `msql.exe` file. This starts the command-line interface that accepts standard SQL commands. For more information, see the *Oracle Database Lite SQL Reference*.

A.1.1.2 Populating your Database Using mSQL

You can use SQL scripts to create tables and schema, and to insert data into tables. A SQL script is a text file, generally with a `.sql` extension, that contains SQL commands. You can run a SQL script from the mSQL prompt, as follows:

```
msql> @<ORACLE_HOME>\DBS\Poldemo.sql
```

You can also type:

```
msql> START <filename>
```

Note: You do not need to include the `.sql` file extension when running the script.

A.1.1.3 SET TERM {ON|OFF}

Controls the display of output generated by commands executed from a script. OFF suppresses the display so that you can spool output from a script without seeing the output on the screen. ON displays the output. TERM OFF does not affect output from commands you enter interactively.

A.1.1.4 SET TIMING {ON|OFF}

Controls the display of timing statistics. ON displays timing statistics on each SQL command. OFF suppresses timing of each command.

A.1.1.5 SET VERIFY {ON|OFF}

Controls whether to list the text of a SQL statement or PL/SQL command before and after replacing substitution variables with values. ON lists the text; OFF suppresses the listing.

A.1.2 The mSQL Tool for Windows CE

The mSQL tool allows the user to execute SQL statements against the local database. You can use either the mSQL tool as a command-line or GUI tool.

The following sections describe the GUI or the command-line tool:

- Section A.1.2.1, "The mSQL GUI Tool"
- Section A.1.2.2, "The Command-Line Version of the mSQL Tool for Windows CE"

A.1.2.1 The mSQL GUI Tool

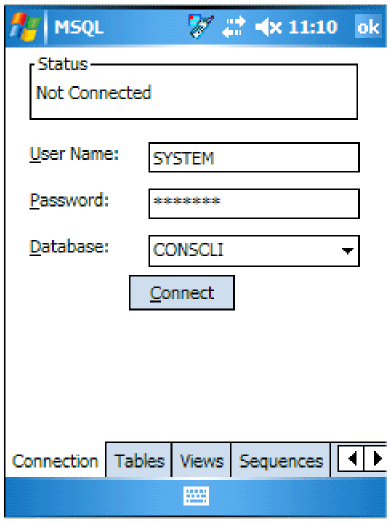
Start the mSQL GUI tool by double-clicking on `msql.exe`. The mSQL GUI tool provides you the ability to perform the following tasks:

- Section A.1.2.1.1, "Connect to the Oracle Lite Database"
- Section A.1.2.1.2, "Execute SQL Statement Against Oracle Lite Database"
- Section A.1.2.1.3, "Create or Encrypt the Oracle Lite Database"
- Section A.1.2.1.4, "Table Contents of the Oracle Lite Database"
- Section A.1.2.1.5, "Views of the Oracle Lite Database"
- Section A.1.2.1.6, "Sequences of the Oracle Lite Database"

A.1.2.1.1 Connect to the Oracle Lite Database Select the Connect tab to connect to the Oracle Lite database on the Mobile device.

1. If you have more than one Oracle Lite database on the device, select the appropriate database from the pull-down.
2. Provide the username and password for this database. If this database was created with a synchronization, the username is `SYSTEM` and the password is the same as the Mobile user password.
3. Click **Connect**.

Figure A–1 Connect to the Oracle Lite Database



Status
Not Connected

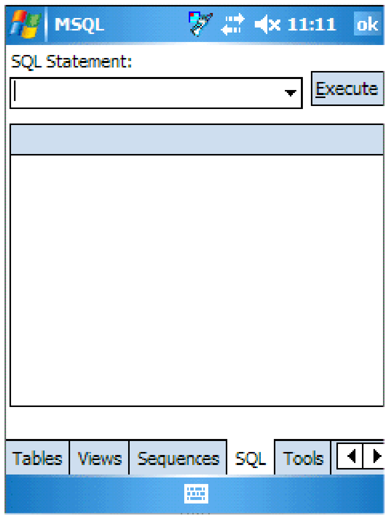
User Name: SYSTEM

Password: ********

Database: CONSCLI

A.1.2.1.2 Execute SQL Statement Against Oracle Lite Database Select the SQL tab to execute a SQL statement against the Oracle Lite database. After connecting, enter your SQL statement and click **Execute**. The statement and any results are displayed in the bottom window.

Figure A–2 Execute a SQL Statement



A.1.2.1.3 Create or Encrypt the Oracle Lite Database The Tools tab enables you to create an Oracle Lite database or to encrypt or validate and existing database.

- Create Database: You can create an Oracle Lite database on the Mobile device. This database can only be used as a standalone database. This database cannot use the synchronization feature with a back-end Oracle database. Use this feature to create a database to use for a standalone application.
- Encrypt/Decrypt/Validate: Select the Oracle Lite database that you want to execute the EncrypDB, DecrypDB or ValidateDB commands against.

  If you are using this database as an embedded database and not for synchronization, then provide the password for the encryption. However, if you are using this database within the Mobile client option for synchronization, do not provide a password, as modifying this password will create an issue for synchronization.

Note: If you are using the Mobile client option and want to encrypt your Oracle Lite database, use the `ENCRYPT_DB` parameter, which is described in Appendix G, "`POLITE.INI` Parameters" in the *Oracle Database Lite Administration and Deployment Guide*.

Click on the appropriate button for each of these functions.

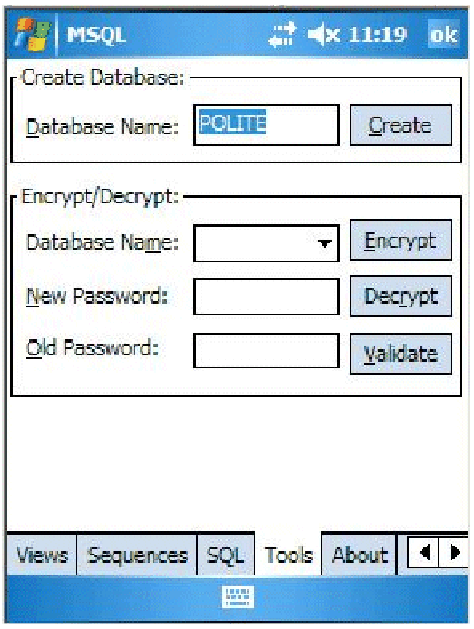
See the following sections for details on EncryptDB, DecryptDB or ValidateDB:

– Appendix A.6, "BACKUPDB"

– Appendix A.5, "DECRYPDB"

– Appendix A.9, "VALIDATEDB"

Figure A–3 Create, Encrypt, Decrypt, or Validate the Oracle Lite Database



MSQL 11:19 ok

Create Database:

Database Name: POLITE [Create]

Encrypt/Decrypt:

Database Name: [Encrypt]

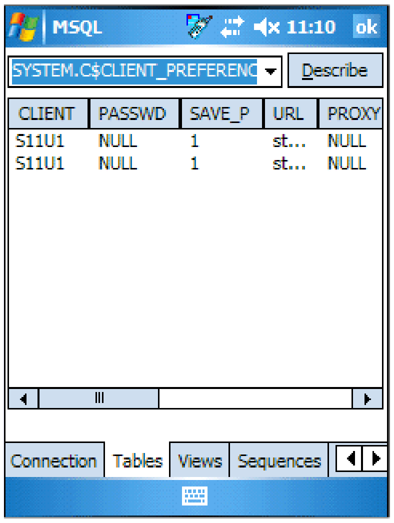
New Password: [Decrypt]

Old Password: [Validate]

Views | Sequences | SQL | Tools | About

A.1.2.1.4 Table Contents of the Oracle Lite Database When you select the Tables tab, you can select any of the tables in the Oracle Lite database and click **Describe**. The structure and contents of this table is displayed.

Figure A–4 Table Description for Oracle Lite Database

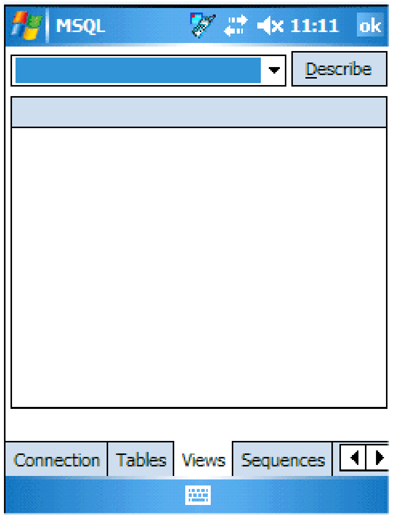


SYSTEM.C$CLIENT_PREFERENC

CLIENT	PASSWD	SAVE_P	URL	PROXY
S11U1	NULL	1	st...	NULL
S11U1	NULL	1	st...	NULL

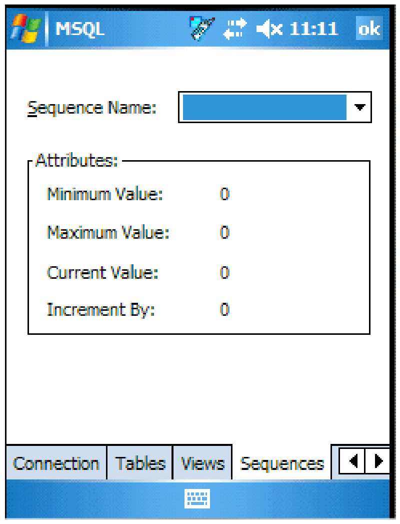
A.1.2.1.5 Views of the Oracle Lite Database When you select the Views tab, you can select any of the views in the Oracle Lite database and click **Describe**. The view definition is displayed.

Figure A–5 Show Views of Oracle Lite Database



A.1.2.1.6 Sequences of the Oracle Lite Database When you select the Sequences tab, you can select any of the sequences in the Oracle Lite database and click **Describe**. The sequence definition is displayed.

Figure A–6 Display Sequence Definition of Oracle Database Lite



A.1.2.2 The Command-Line Version of the mSQL Tool for Windows CE

The following sections describe the command-line version of mSQL and how to access the database for the Windows CE platform:

- Section A.1.2.2.1, "Starting mSQL"
- Section A.1.2.2.2, "Manage Snapshots Using mSQL"

A.1.2.2.1 Starting mSQL Start mSQL by opening the <*ORACLE_HOME*>`\Mobile\Sdk\WinCE`, selecting the folder representing the version Windows CE, and selecting the processor on your device. Double-click on the `msql.exe` file. This starts the GUI which accepts standard SQL commands. For more information, see the *Oracle Database Lite SQL Reference*.

A.1.2.2.2 Manage Snapshots Using mSQL The Oracle Lite database format is the same on Windows 32 and Windows CE platforms. Manage your snapshots, as follows:

1. Create and test your snapshots on Windows 32 using the Windows 32 mSQL command-line utility.
2. Copy the database to the Windows CE platform.
3. Use the Windows CE mSQL tool to manipulate the database that is on your device.

The mSQL tool enables the user to execute SQL statements against the local database and access functionality provided by the interfaces of the underlying Oracle Lite database engine.

A.2 CREATEDB

Description

Utility for creating a database.

Syntax

```
CREATEDB DataSourceName DatabaseName Database_SysUser_Password [[[VolID] DATABASE_
SIZE] EXTENT_SIZE] [collation sequence]
```

Keywords and Parameters

`DataSourceName`

Data source name, used to look up the `ODBC.INI` file for the default database directory.

Note: If you specify an invalid DSN, Oracle Database Lite ignores the DSN and creates the database in the current directory. To access this database through ODBC, you must create a DSN for the database that points to the directory in which the database resides. For instructions on adding a DSN, see Section A.7.1, "Adding a DSN Using the ODBC Administrator".

`DatabaseName`

Name of the database to be created. It can be a full path name or just the database name. If only the database name is given, the database is created under the Data Directory for the data source name specified in the `ODBC.INI` file. The extension for the database name must always be `.ODB`. If a name without the `.ODB` is given, the `.ODB` is appended.

`DATABASE_SysUser_Password`

The database system user password.

`VolID`

When specified, the `VolID` is used as the database ID, instead of the database ID from the `POLITE.INI` file. The ID must be unique for each database. If you specify a volumn id, then you also specify the database and extent sizes. Thus, the createdb executable knows that the volume id, database size and extent size are being specified when three numbers are provided in a row.

Note: For the volume id, database size, and extent size, specify only the number; do not specify name=value. See the examples for more information.

`DATABASE_SIZE`

The database size in bytes. If you want to specify the database size, then you also must specify the volume id and extent size.

`EXTENT_SIZE`

An incremental amount of pages in a database file. When a database runs out of pages in the current file, it extends the file by this number of pages. If you want to specify the extent size, then you also must specify the volume id and database size.

`COLLATION_SEQUENCE`

This parameter is a string constant which creates the database as enabled for linguistic sorting when a value other than the default is used. A collation sequence specified here overrides a collation sequence set using the `NLS_SORT [collation_sequence]` parameter in the `POLITE.INI` file. The string can also be one of the options listed in Table A–2:

Table A–2 Collation Sequence Values

Collation Sequence	Description
`BINARY`	Default. Two strings are compared character by character and the characters are compared using their binary code value. You cannot perform a linguistic sort with an Oracle Lite database that has a binary collation sequence.
`FRENCH`	Two strings are compared according to the collation sequence of French. Supported by ISO 8859-1 or IBM-1252.
`GERMAN`	Two strings are compared according to the collation sequence of German. Supported by ISO 8859-1 or IBM-1252.
`CZECH`	Two strings are compared according to the collation sequence of Czech. Supported by ISO 8859-2 or IBM-1250.
`XCZECH`	Two strings are compared according to the collation sequence of Xczech. Supported by ISO 8859-2 or IBM-1250.

Note: There is no way to alter a collation sequence after the database is created.

Examples

Create the `db1` database with DSN of `polite` and password `manager`: `createdb polite db1 manager`

Create the `db2.odb` database with DSN `polite` and password `manager300`: `createdb polite c:\testdir\db2.odb manager300`

Create `polite` database with DSN polite, password of `manager`, and a collation sequence of `french`: `createdb polite polite manager french`

Create polite database with DSN `polite`, password `manager`, volume id of 199, database size of 1000, and extent size of 1:
`createdb polite polite manager 199 1000 1`

A.3 REMOVEDB

Description

Utility for deleting a database.

Syntax

REMOVEDB DataSourceName Database Name

Keywords and Parameters

DataSourceName

Data source name of the database you want to remove. The DSN can be a dummy argument such as none, in which case the database name must be a fully qualified filename.

DatabaseName

The name of the database to delete. It can be a full path name or just the database name. If only the database name is given, the database is deleted from the Data Directory for the data source name specified in the `ODBC.INI` file.

Examples

```
removedb polite db1
```

```
removedb none c:\testdir\db2.odb
```

A.4 ENCRYPDB

Description

Enables you to encrypt Oracle Database Lite with a password, which prevents unauthorized access to the database and encrypts the database, so that the data stored in the database files cannot be interpreted. To decrypt the database, see Section A.5, "DECRYPDB".

This tool is used by embedded applications to encrypt the database used by the application. If you are using this database as an embedded database and not for synchronization, then provide the Mobile user password for the encryption. However, if you are using this database within the Mobile client option for synchronization, do not provide a password, as modifying this password will create an issue for synchronization.

This is more difficult on a handheld as it is sometimes difficult for users to find the RUN option in order to execute the command with arguments.

Note: If you are using the Mobile client option and want to encrypt your Oracle Lite database, use the `ENCRYPT_DB` parameter, which is described in Appendix G, "`POLITE.INI` Parameters" in the *Oracle Database Lite Administration and Deployment Guide*.

`ENCRYPDB` uses AES-128 encryption.

Syntax

`ENCRYPDB` *DSN* | `NONE` *DBName* [*New_Password* [*Old_Password*]]

Keywords and Parameters

- `DSN`—Data Source Name of Oracle Database Lite that you want to encrypt. If you specify `NONE`, `DBName` must be a fully qualified database name with the full path name (without the `.ODB` extension). If the `DSN` is a value other than `NONE`, then the name must appear as a data source name in the `ODBC.INI` file.
- `DBName`—Name of the database to be encrypted. If DSN was specified as `NONE`, `DBName` must be entered with the full path name.
- `New_Password and Old_Password`—Optional, the password (or previously used password) for encrypting the database. This password can be 128 characters in length. If you do not enter a password, `ENCRYPDB` prompts you to enter one. Since both passwords are optional in the command line to invoke the utility, the command line could have three different forms:

- No password given: If the database is already encrypted, then ENCRYPDB assumes that the user is trying to change the password of the database. It prompts the user for the old password once and new password twice, and encrypts the database using the new password. If the database is not already encrypted, ENCRYPDB prompts for the new password twice and encrypts the database using this new password.
- One password given: This password is assumed to be the new password. If the database is already encrypted, ENCRYPDB prompts for the old password and encrypts the database using the new password.
- Both passwords given: ENCRYPDB assumes that the first password is the new password and the second is the old password.

To run from the command line, you must pass in the DSN name and the database name. The following encrypts the employee database with DSN of Employee with the test password:

```
Encrypdb Employee employee test test
```

Comments

If you call this utility from another program, the possible values returned are listed in Table A–3:

Table A–3 ENCRYPDB Return Codes

Return Code	Description
EXIT_SUCCESS	Success
EXIT_USAGE	Command line arguments are not properly used or are in error
EXIT_PATH_TOO_LONG	Path is too long
EXIT_SYSCALL	I/O error while making new encrypted copy on disk
EXIT_BAD_PASSWD	Incorrect password supplied

The default Oracle Database Lite (POLITE.ODB) is not encrypted. After encrypting an Oracle Database Lite, every user that attempts to establish a connection to the encrypted Oracle Database Lite must provide the valid password. If the password is not provided, Oracle Database Lite returns an error. An Oracle Database Lite database cannot be encrypted if there are any open connections to the database.

You should consider the following when encrypting and decrypting Oracle Database Lite:

- You cannot decrypt an encrypted database without the password. Make sure you back up your database in a secure place before you encrypt it. Another user of the same database can create a copy with a new user name for a user who loses their password, otherwise, there is no method to recover a database where the passwords are lost.
- A password encrypts the entire database. It is not a user-specific password.
- Database encryption does not prevent a third party from removing an Oracle Lite Database. That is, removedb and rmdb remove a database without checking the password. Use tools that protect unauthorized users from manipulating your file system.
- ODBC applications that connect to an encrypted Oracle Database Lite database need to specify a valid password. It is customary to prompt for the password at

runtime rather than to code it in the application. Most ODBC applications can use the `SQLDriverConnect` function with the `DRIVER=` option, rather than the `SQLConnect` function, if the applications require the Oracle Database Lite ODBC driver to prompt for the password at runtime.

- All sample applications provided with this release of Oracle Database Lite are designed to run against a database that is not encrypted.
- You can use `DECRYPDB` and `ENCRYPDB` (in this order) to change the password of a database. However, `DECRYPDB` creates an Oracle Database Lite database in plain text before `ENCRYPDB` encrypts it. This results in a database in plain text form, for a short period of time, and is not recommended.
- For encrypted databases, all user names and passwords are written to a file named `DSN.OPW`. Each user can then use the password as a "key" to unlock the `.OPW` file before the `.ODB` file is accessed. When you copy or back up the database, you should include the `.OPW` file.

A.5 DECRYPDB

Description

This tool allows you to decrypt an encrypted Oracle Lite database used with an embedded application. For more information, see Section A.6, "BACKUPDB".

This tool is used by embedded applications to decrypt the database used by the application. To encrypt an Oracle Lite database used by a Mobile client, see the `ENCRYPT_DB` parameter in the `POLITE.INI` Appendix in Appendix G, "`POLITE.INI` Parameters" in the *Oracle Database Lite Administration and Deployment Guide*.

SYNTAX

DECRYPDB *DSN* | NONE *DBName* [*Password*]

Keywords and Parameters

`DSN`

Data Source Name of Oracle Database Lite that you want to decrypt. If you specify `NONE`, you must the enter the `DBName` with the full path name (without the `.ODB` extension).

`DBName`

Name of the database to be decrypted. If DSN was specified as `NONE`, the `DBName` must be entered with the full path name.

`Password`

Optional. The password used previously to encrypt Oracle Database Lite. If you do not enter the password, `DECRYPDB` prompts you to enter it.

Comments

An Oracle Database Lite database cannot be decrypted if there is any open connection to the database.

If you call this utility from another program, the possible values returned are listed in Table A–4:

Table A–4 DECRYPDB Return Codes

Return Code	Description
`EXIT_SUCCESS`	Success
`EXIT_USAGE`	Command line arguments are not properly used or are in error
`EXIT_PATH_TOO_LONG`	Path is too long
`EXIT_SYSCALL`	I/O error while making new decrypted copy on disk
`EXIT_BAD_PASSWD`	Incorrect password supplied

For more information, see the comments in Section A.6, "BACKUPDB".

A.6 BACKUPDB

Description

For either the Mobile client or embedded solutions, you can back up the Oracle Lite database either by using the `backupdb` utility or by copying the files to another location.

Oracle Database Lite uses the ODB and OBS files with dependent log files that can be backed up by copying to another location. Before any files can be copied, disconnect all applications that access the database and shut down the multi-user service, if running. Once that has been accomplished, execute the backupdb utility, which copies the `*.odb`, `*.obs, *.opw`, and `*.plg` files to the filename of your choice to make a backup of the database.

```
BACKUPDB DSN|NONE DBName backup_filename
```

If you want to restore the backup, then execute the `backupdb` executable, with `NONE` and reversing the `filename` and the `dbname`, as follows:

```
BACKUPDB NONE backup_filename DBName
```

This is more difficult on a handheld as it is sometimes difficult for users to find the RUN option in order to execute the command with arguments.

Syntax

```
BACKUPDB DSN| NONE DBName <backup_filename> ]
```

Keywords and Parameters

- `DSN`—Data Source Name of Oracle Database Lite that you want to backup. If you specify `NONE`, `DBName` must be a fully qualified database name with the full path name (without the `.ODB` extension). If the `DSN` is a value other than `NONE`, then the name must appear as a data source name in the `ODBC.INI` file.
- `DBName`—Name of the database to be encrypted. If DSN was specified as `NONE`, `DBName` must be entered with the full path name.
- `backup_filename`—File where you want the backup to be stored. This can include an absolute or relative path. If no path is included, then the file is stored in the directory where the command is executed.

To run from the command line, you must pass in the DSN name and the database name. The following backs up the `employee` database with DSN of `Employee` into the `backupemployee` file:

```
Backupdb Employee employee backupemployee
```

A.7 ODBC Administrator and the Oracle Database Lite ODBC Driver

A Data Source Name (DSN) associates the Oracle Database Lite ODBC Driver with the Oracle Database Lite database that you want to access through the driver. The Oracle Database Lite installation process creates a default `DSN`, `POLITE`, for the Oracle Database Lite database. You can also create additional DSNs for the additional Oracle Database Lite databases that you create.

Microsoft provides the ODBC Administrator, a tool for managing the `ODBC.INI` file and associated registry entries in `Windows 2000/XP`. The `ODBC.INI` file and the Windows registry store the DSN entries captured through the ODBC Administrator. Using the ODBC Administrator, you can relate a DSN to the Oracle Database Lite ODBC Driver.

Note: This document does not provide instructions on using the ODBC Administrator. See the ODBC Administrator tool online help for this information.

In the ODBC Administrator, in addition to the DSN, you must specify the parameters listed in Table A–5:

Table A–5 ODBC Administrator DSN Parameters

DSN Parameter	Description
`Data Description`	An optional description for the data source.
`Database Directory`	The path to the data directory where the database resides. This is an existing path.
`Database`	Oracle Database Lite database name to be created. Do not include the `.ODB` extension.
`Default Isolation Level`	Determines the degree to which operations in different transactions are visible to each other. For more information on the supported isolation levels, refer the *Oracle Database Lite Developer's Guide*. The default level is "`Read Committed`".

Table A–5 (Cont.) ODBC Administrator DSN Parameters

DSN Parameter	Description
`Autocommit`	Commits every database update operation in a transaction when that operation is performed. Autocommit values are Off and On. The default value is Off. **Note**: In the Microsoft ODBC SDK, the ODBC driver defaults to auto-commit mode. However, the default for Oracle Database Lite is manual-commit mode. In this environment, if you execute SQLEndTrans / SQLTransact call with SQL_COMMIT option using the ODBC driver, you receive a SQL_SUCCESS, because ODBC believes that auto-commit is on. However, no commit actually occurs, because ODBC transfers the transaction to Oracle Database Lite, whose default is manual-commit. You must configure the Microsoft ODBC Driver Manager to transfer control of the SQLEndTrans / SQLTransact API call to Oracle Database Lite by explicitly setting autocommit to OFF in ODBC. When you do this, ODBC does not try to autocommit, but gives control of the transaction to Oracle Database Lite. To set auto-commit to off, execute either the SQLSetConnectAtrr or SQLSetConnectOption method with SQL_AUTOCOMMIT_OFF as the value of the SQL_AUTOCOMMIT option. Then, the SQLEndTrans / SQLTransact calls will commit as defaulted within Oracle Database Lite. Thus, if you want auto-commit on, turn it on only within Oracle Database Lite.
`Default Cursor Type`	■ *Forward Only*: Default. A non-scrollable cursor which only moves forward but not backward through the result set. As a result, the cursor cannot go back to previously fetched rows. ■ *Dynamic*: Capable of detecting changes to the membership, order, or values of a result set after the cursor is opened. If a dynamic cursor fetches rows that are subsequently deleted or updated by another application, it detects those changes when it fetches those rows again. ■ *Keyset Driven*: Does not detect change to the membership or order of a result set, but detects changes to the values of rows in the result set. ■ *Static*: Does not detect changes to the membership, order or values of a result set after the cursor is opened. If a static cursor fetches a row that is subsequently updated by another application, it does not detect the changes even if it fetches the row again.

For example, the DSN entry for `POLITE` in the `ODBC.INI` file may contain:

```
[POLITE]
Description=Oracle Lite Data Source
Data_Directory=C:\ORANT\OLDB40
Database=POLITE
IsolationLevel=Repeatable Read
CursorType=Dynamic
```

Note: The `ODBC.INI` file is available in Windows under `%WINDIR%` and in Linux under `$OLITE_HOME/bin`. For the Linux platform, you must have write permissions on the directory where this is located to be able to modify them.

A.7.1 Adding a DSN Using the ODBC Administrator

To add a DSN using the ODBC Administrator:

1. Start the ODBC Administrator, either by selecting its icon in the Oracle Database Lite program group, or by typing the following at a DOS prompt:

   ```
   C:\>ODBCAD32
   ```

2. Click **Add**.
3. Double-click the **Oracle Database Lite *nn* ODBC Driver**, where *nn* is the release number, from the list of Installed ODBC Drivers.
4. Next, add the DSN name and define the parameters in the ODBC driver setup dialog. Refer the preceding table for help in defining the parameters.

A.7.2 Adding a DSN which points to Read-Only Media (CD-ROM)

1. Create the DSN as explained in Section A.7.1, "Adding a DSN Using the ODBC Administrator".
2. Add the following line to the new DSN in the `ODBC.INI` file:

   ```
   ReadOnly = True
   ```

Note: You can define a DSN which points to a file on a CD-ROM. Simply point the DSN to the CD-ROM drive and directory and provide the file name of the database file. Then modify the `ODBC.INI` file to add the line `ReadOnly=True` to the data source definition. ODBC programmers can call the following before opening the database to enable this feature (instead of adding the line to the `ODBC.INI` file):

```
SQLSetConnectOption( hdbc, SQL_ACCESS_MODE, SQL_
MODE_READ_ONLY )
```

Setting a database file to read-only suppresses the creation of log files. Updates, insertions, deletions, or commits appear to work on the in-memory image of tables. However, when you commit, these changes are not written to the database file. If you exit your application, reconnect, and issue your query, you see your original data.

A.8 ODBINFO

Description

You can use `ODBINFO` to find out the version number and volume ID of an Oracle Database Lite database. `ODBINFO` can also display and set several parameters.

Syntax

To display current information without making any changes use the syntax:

```
odbinfo [-p passwd]DSN DBName
```

You can also use:

```
odbinfo [-p passwd] NONE dbpath\dbanme.odb
```

For example:

```
odbinfo -p tiger polite polite
```

```
odbinfo NONE c:\orant\oldb40\polite.odb
```

If your database is encrypted you need to include the password.

Parameters

To set or clear parameters, use one or more "+" or "-" parameter arguments before the DSN or NONE. For example:

```
odbinfo +reuseoid -pagelog -fsync polite polite
```

You can use the parameters listed in Table A–6 with the `ODBINFO` utility:

Table A–6 ODBINFO Parameters

Parameter	Description
`pagelog`	By default, a commit backs up modified database pages to `filename.plg` before actually writing the changes to `filename.odb`. If an application or the operating system experiences a failure during a commit, the transaction is cleanly rolled back during the next connect. If `-pagelog` is specified, no backup is created and the database can become corrupted if a failure occurs.
`fsync`	Oracle Database Lite generally forces the operating system to write all the modified buffers associated with the database back to disk during a commit. If this option is disabled (`-fsync`), the operating system can keep the changes in memory until a later time. If the system (but not the application) crashes before the buffers are flushed, the database can become corrupted. Using `odbinfo -fsync -pagelog` improves the performance of applications that use many small transactions (with autocommit on) or ones with massive updates. However, if the database is corrupted, there is no straightforward way to repair it or recover the data. Therefore these two options should only be cleared during initial loading of the database, if (1) the `.ODB` file is backed up on regular basis, or (2) the data in the database can be recovered from some other source. Using this option has no effect on applications that seldom update the database. Setting the transaction isolation level to `SINGLE USER` has more impact in this case.
`reuseoid`	By default, Oracle Database Lite does not reuse the `ROWID` of any row that exists in a table until the table is dropped. The "Slot Deleted" error is returned when accessing a deleted object. This uses two bytes of storage for each deleted object, causing performance and disk space usage to degrade over time if rows are constantly inserted and deleted. If you use `odbinfo +reuseoid`, new rows can reuse `ROWID`s of previously deleted rows. However, this may not free all the space in a table that already has many deleted objects. For best results, you should set this option immediately after you create your database. This option is safe for pure relational applications. However, SQL applications that use `ROWID` and `OKAPI` applications that use direct pointers between objects need to verify that all references to an object are set to `NULL` before the object is deleted. Otherwise, dangling references may eventually point to some other, unrelated object.

Table A–6 (Cont.) ODBINFO Parameters

Parameter	Description
`compress`	This option (which is "on" by default) enables run-length compression of objects. Run-length compression takes very little CPU time, so you should only deselect (`-compress`) this option if: ■ Operating system-level file compression is used, such as `DriveSpace` or a `NTFS` compressed attribute. In this case not compressing the same data twice provides a better compression ratio. ■ Most objects in the database are frequently updated to a highly compressible state (for example, all columns set to `NULL`), and the data cannot be compressed well (such as binary columns with random data). In these cases, using this option (`+compress`) can result in highly fragmented tables. Changing this option does not compress or decompress any existing objects in the database.

A.9 VALIDATEDB

This command-line tool validates the structures within the database file and if the database structure is found to be corrupted, lists the errors found in a file designated by the user. The tool checks the following:

- Objects - Header information for database objects. Flags are checked for consistency in case the object was moved or compressed. Object length is checked against a valid range. If the object is a BLOB, the object's frames are checked against the volume page bitmap.
- Index page entries - Checks that the creation of an index page entry results in the correct number of nodes or list of object identifiers.
- Index pages - Checks that all key values on the page are sorted. All objects contained on the page are validated. Page descriptor information such as the number of objects, the number of free bytes, and the number of entries are checked against the actual objects on the page.
- Groups - As each page is validated, the group descriptor information is checked against the actual number of pages and objects.
- Indexes - All the pages are validated against the btree. The tool also validates all page pointers. All levels of the btree are checked to validate that key values are in the sorted order as a whole. For leaf elements of the btree, all OIDs from the leaf page entries are checked for consistency with the actual group objects.

Syntax

validatedb *DSName DBName* [-p *password*] [-l username:password] [-t *schemaname.tablename*] -file *outputfilename*

Keywords and Parameters

DSName

The data source name. This can also be `NONE` if no DSN is present.

DBName

If there is a DSN present, this is the database file name (without the .odb extension) if it is different from the default filename for the DSN. If there is no DSN, then `VALIDATEDB` uses the current directory unless the full path is specified. If there is a log file in the same directory as the database file, it is also validated.

-p password

Password for an encrypted database.

-l username:password

Optional. Provide the username/password to log into the Oracle Lite database that you are validating with the `-l username:password` option.

Note: This is not available on Windows CE.

The following details the behavior of this option:

1. If the database is encrypted, and the encryption password (the `-p` option) is not supplied, then the password included in the login (`-l`) option is used as the encryption password.
2. If you do supply an encryption password in the `-p` option as well as a login password with the `-l` option, then the login password is used to verify that the encryption password is correct.

-t schemaname:tablename

Optional.

- schema name. The default schema name is used unless this is specified.
- table name. The specified table is validated along with all of its indexes. If no table name is specified, the entire database is validated.

-file outputfilename

Optional filename for the text file where all errors and other related information revealed by `VALIDATEDB` are saved. The default is stdout.

Examples

```
validatedb polite polite -t emp -file out.txt
```

A.10 Transferring Data Between a Database and an External File

You can transfer data between an external file and the Oracle Lite database through either a command-line tool or programmatic APIs, as described in the following sections:

- Section A.10.1, "OLLOAD"
- Section A.10.2, "Oracle Database Lite Load Application Programming Interfaces (APIs)"

A.10.1 OLLOAD

The Oracle Database Lite Load Utility (`OLLOAD`) is a command-line tool, which enables you to load data from an external file into a table in Oracle Database Lite or to unload (dump) data from a table in Oracle Database Lite to an external file. Unlike SQL*Loader, `OLLOAD` does not use a control file in which you supply all data parameters and format information on the command-line.

When loading data, `OLLOAD` takes an input file that contains one record per line with a separator character between fields. The default field separator is a comma (,). These records can also include fields with values that are quoted strings. The default value is single quote ('). For more information on data parsing, see "Data Parsing".

A.10.1.1 Syntax

Loading a Datafile

To load a datafile, use the following syntax.

olload [options] -load *dbpath tbl* [*col1 col2* ...] [<*datafile*]

Unloading (dump) to an Outfile

olload [options] -dump *dbpath tbl* [col1 col2 ...] [>*outfile*]

A.10.1.2 Keywords and Parameters

This section describes keywords and parameters that are available for the `OLLOAD` utility.

[options]

For a list of options, see Section A.10.1.2.1, "Options".

-load

To use the load utility.

-dump

To use the unload (dump) utility.

dbpath

The path to the Oracle Database Lite (`.odb`) file.

tbl

The table name. `OLLOAD` first attempts to find a table name in the user-specified case. If this fails, it searches for the upper-case of the user-specified name.

Note: The default user is `SYSTEM`. To specify an `OLLOAD` operation for another user name's tables, prefix the `tbl` parameter with the user name and a dot (.).

col1 col2

The column names. `OLLOAD` first attempts to find a column name in the user-specified case. If this fails, it searches for the upper-case of the user-specified name.

[datafile] [outfile]

The source or destination file for the load or unload operations. If you do not specify a datafile or outfile, `OLLOAD` displays the output on the screen.

A.10.1.2.1 Options

This section describes keyword and parameter options that are available for the `OLLOAD` utility.

-sep *character*

The field separator. If you do not specify this option, `OLLOAD` assumes that the separator character is a comma (,).

-quote *character*

The quote character. If you do not specify this option, `OLLOAD` assumes that the quote character is a single quote (’).

-file *filename*

Use this option when loading and unloading data to specify the source or destination file name. When loading data, filename specifies the source file to load into Oracle Database Lite. When unloading (dumping) data, it is the destination file for the unloaded data.

> **Note:** To unload data from Oracle Database Lite and load (or pipe) it to another Oracle Database Lite, do not specify a file name for this option. For a description of sample syntax, see "Examples".

-log *logfile*

Specify this option if you want to produce a log file listing rows that OLLOAD could not insert during load. If you do not specify a log file, loading stops at the first error.

-passwd *passwd*

The connection password for an encrypted database. You need to supply this password so that loading and unloading can occur.

-nosingle

Specify this option when you do not want to use single user mode. This degrades performance but allows other connections to the database.

-readonly

Specify this option when unloading data from a read-only Oracle Database Lite, for example, one located on a CD-ROM.

-commit *count*

Use this option if you want OLLOAD to commit after processing a specified number of rows. The default is 10000. OLLOAD prints an asterisk (*) to the screen each time it commits the specified number of rows. To disable the commit operation specify 0.

-mark *count*

Use this option if you want OLLOAD to print a dot on the screen after processing the specified number of records. The default is 1000. To disable this feature specify 0.

Data Parsing

Table A–7 lists examples for `OLLOAD` data parsing.

Table A–7 Data Parsing Examples

Input	Data	Explanation
'Redwood Shores, CA'	Redwood Shores CA	Enclosing the input string in quotes preserves spaces and punctuations within a string.
'O''Brien'	O'Brien	Represent a single quote with its escape sequence, two single quotes.
fire fly	firefly	Spaces in data that is not quoted is ignored.
,	NULL,NULL	Empty fields are NULL.
1,,3	1,NULL,3,NULL	Empty fields are NULL.
	[no row inserted]	Completely empty lines are ignored.

If there are more values than database columns, extra values are ignored. Any missing values at the end of the line are set to NULL.

OLLOAD Utility Restrictions

`OLLOAD` does not support tab-delimited input files and LONG datatypes.

Examples

```
olload -quote \" -file p_kakaku.csv -load c:\orant\oldb40\polite.odb skkm01
olload -dump c:\orant\oldb40\polite.odb emp empno ename | olload -load myfile.odb
myemp
```

A.10.2 Oracle Database Lite Load Application Programming Interfaces (APIs)

This document describes the Oracle Database Lite Load APIs. Each section of this document presents a different topic. These topics include:

- Section A.10.2.1, "Overview"
- Section A.10.2.2, "Oracle Database Lite Load APIs"
- Section A.10.2.3, "File Format"
- Section A.10.2.4, "Limitations"

A.10.2.1 Overview

The Oracle Database Lite Load APIs allow you to load data from an external file into a table in Oracle Database Lite, or to unload (dump) data from a table in Oracle Database Lite to an external file. For information on using the command line tool `OLLOAD`, see Section A.10.1, "OLLOAD". You can use the API calls presented in this document to make your own customizations.

A.10.2.2 Oracle Database Lite Load APIs

The Oracle Database Lite Load APIs include:

- Section A.10.2.2.1, "Connecting to the Database: olConnect"
- Section A.10.2.2.2, "Disconnecting from the Database: olDisconnect"
- Section A.10.2.2.3, "Deleting All Rows from a Table: olTruncate"
- Section A.10.2.2.4, "Setting Parameters for Load and Dump Operations: olSet"
- Section A.10.2.2.5, "Loading Data: olLoad"

- Section A.10.2.2.6, "Dumping Data: olDump"
- Section A.10.2.2.7, "Compiling"
- Section A.10.2.2.8, "Linking"

The normal mechanism for unloading and loading a table is as follows:

1. Declare local variable, `DBHandle`.
2. Connect to the database using `olConnect`.
3. Optionally, set parameters for load or unload.
4. Dump or load the data using `olDump` or `olLoad`. You may optionally delete all rows from a table by calling `olTruncate`.
5. Disconnect from the database using `olDisconnect`.

A.10.2.2.1 Connecting to the Database: olConnect Use this API to connect to the database. This is the first API that you have to call. It creates a load and unload context that is used in subsequent APIs to influence the load and unload behavior. This returns an initialized database handle DBHandle.

Syntax

```
olError olConnect (char *database_path, char *password, DBHandle &dbh);
```

The arguments for `olConnect` are listed in Table A–8:

Table A–8 olConnect Arguments

Argument	Description
`database_path`	The full path to the database file (directory path and filename).
`password`	The password used for the encrypted database, for any other database the password = NULL.
`dbh`	The application handle for the current database connection. This allows multiple database connections for one application thread (each connection has a different handle).

Return Values

(short) integer error code

Values from -1 to -8999 are used for the error codes returned by the database, values from -9000 and below are used for olLoad-specific error codes.

A.10.2.2.2 Disconnecting from the Database: olDisconnect Disconnects from the database.

Syntax

```
olError olDisconnect (DBHandle dbh);
```

The arguments for `olDisconnect` are listed in Table A–9:

Table A–9 olDisconnect Arguments

Argument	Description
`dbh`	The current application handle.

Return Value

(short) integer error code

A.10.2.2.3 Deleting All Rows from a Table: olTruncate This API can be used to delete all rows from an existing table.

Note: Records removed from the server through a truncate command will not be removed from the client unless a complete refresh is triggered. The truncate command is considered a DDL operation. Consequently, the necessary DML triggers do not fire and therefore the operations are not logged for fast refresh.

Syntax

```
olError olTruncate (DBHandle  dbh, char* table );
```

The arguments for `olTruncate` are listed in Table A–10:

Table A–10 olTruncate Arguments

Argument	Description
`dbh`	The current application handle.
`tablename`	The name of the table in the form: owner_name.table_name. where owner_name is the name of the owner of the table.

Return Value

(short) integer error code

A.10.2.2.4 Setting Parameters for Load and Dump Operations: olSet This is an optional API. This sets optional parameters for load and unload.

Syntax

```
olError olSet (DBHandle  dbh, char * parameter_name, char *parameter_value);
```

The arguments for `olSet` are listed in Table A–11:

Table A–11 olSet Arguments

Argument	Description
`dbh`	The current application handle.
`parameter_name`	The name of the given parameter. This is not case sensitive. See Section A.10.2.3.2, "Parameters" for a list of parameter names and their default values.
`parameter_value`	The value to be set. This is not case sensitive for most parameters.

Return Value

(short) integer error code

A.10.2.2.5 Loading Data: olLoad `OlLoad` loads data from a file into a table using current parameter settings.

Syntax

```
olError olLoad (DBHandle dbh, char *table, char *file);
```

The arguments for `olLoad` are listed in Table A–12:

Table A–12 olLoad Arguments

Argument	Description
dbh	The current application handle.
table	The table information in the form: `owner_name.table_name(col1,col2,...)` where `col1,col2,...` is the list of column names to load. This allows you to load and dump certain columns instead of the entire table. If the entire table is to be dumped, the column list need not be specified.
file	The path to the file from which loading takes place.

Note: If table = NULL, `olLoad` tries to find the table description in the file header.

Return Value

(short) integer error code

A.10.2.2.6 Dumping Data: olDump `OlDump` dumps data from a table into a file using current parameter settings.

Syntax

```
olError olDump (DBHandle dbh, char *table, char *file);
```

The arguments for `olDump` are listed in Table A–13:

Table A–13 olDump Arguments

Argument	Description
`dbh`	The current application handle.
`table`	The table information in the same form as `olLoad`.
`file`	The file to which dump data is written.

Return Value

(short) integer error code

A.10.2.2.7 Compiling The declarations for the DBHandle, parameter constants and flags, and error message codes are given in the file **olloader.h** in the *ORACLE_HOME*\Mobile\SDK\include directory. For compilation of your product include olloader.h in your main source file.

A.10.2.2.8 Linking Linking use the file **olloader40.dll** and the library file **olloader40.lib**. Include these files in your project settings.

A.10.2.3 File Format

The Oracle Database Lite Load APIs support three file formats FIXEDASCII, BINARY and CSV. Each file contains an optional header followed by zero or more rows of data.

A.10.2.3.1 Header Format

The header has the following format (comments are in bold):

```
$$OL_BH$$ [begins header]
VERSION=xx.xx.xx.xx   [version number]
TABLE=T1(C1, C2, ...)... [table name with list of column names dumped]
FILEFORMAT=FIXEDASCII
SEPARATOR=,
[any other parameters in the parameter list can be listed here]
$$OL_EH$$ [ends header]
```

The following is a header example:

```
$$OL_BH$$
VERSION=01.01.01.01
TABLE=T1(EMPNO,SALARY)
FILEFORMAT=BINARY
BITARRAY=TRUE
HEADER=TRUE
RDONLY=FALSE
LOGFILE=
COMMITCOUNT=-1
NOSINGLE=TRUE
$$OL_EH$$
```

The header lines can be in any order and all lines except $$OL_BH$$ and $$OL_EH$$ can be considered optional. Although, during the dump, if the header flag is on, table information and all parameter settings are dumped into the header.

When executing load, parameter information in the header overwrites current parameter settings. If the table argument in `olLoad` is NULL, the table name and list of columns in the header prevails, otherwise the table argument of `olLoad` prevails over the header.

A.10.2.3.2 Parameters

Header file parameters listed in Table A–14 are not case sensitive.

Table A–14 Parameters

Parameter	Description
`FILEFORMAT`	Input and output file format. The following formats are supported: ▪ FixedASCII - text file with fixed field width for each datatype. ▪ CSV – comma separated values format. ▪ Binary - binary file format. These key word values are not case sensitive.
`SEPARATOR`	The separator between the values (one character), comma by default.
`QUOTECHAR`	The quote character for the string datatype values in the file, single quote (') by default.
`LOGFILE`	The log file name. NULL by default (no log file produced and loading stops at the first error).
`NOSINGLE`	FALSE for single user mode (the default), or TRUE for no single user mode.

Table A–14 (Cont.) Parameters

Parameter	Description
`READONLY`	FALSE (the default). TRUE to dump the data from read-only database (such as CD-ROM).
`COMMITCOUNT`	The number of rows processed after which `olLoad`, `olDump`, and `olTruncate` commit. The default value is -1, not to commit at all. Value 0 commits at the end of the operation, and values above 0 commit after the specified number of rows.
`HEADER`	FALSE (the default). TRUE to create a header in the beginning of the file during `olDump`.
`BITARRAY`	TRUE (the default) to support writing and reading nulls in binary format. During the dump, a bit array with the null information is dumped before each row. For FALSE `olDump` provides an error trying to write nulls in binary.
`NONULL`	TRUE (the default) when trying to read or write nulls `olLoad` and `olDump` return an error. When the flag is set to FALSE nulls are supported, including binary format since the default BITARRAY value is TRUE.
`DATEFORMAT`	The string for which date and timestamp columns should be written into the file and read from the file in FIXED ASCII and CSV formats. Such formats as "YYYYMMDD", "YYYY-MM-DD", and "YYYY/MM/DD" are supported. The default value is empty string (which can also be set using NULL), and the default date format is "YYYY-MM-DD". (In Oracle mode, date is treated the same as timestamp so that the date format is the default timestamp format which is "YYYY-MM-DD HH:MM:SS.SSSSSS".)

A.10.2.3.3 Data Format The data format can be comma separated value (CSV), fixed ASCII, or binary. The following cases apply:

- CSV Format: Each row of the table is represented as a separate line in the file. Each line is separated by a carriage return and a line feed character on the Windows platform. Each value in the row is separated by a separator character which by default is a comma.

  Each value is also quoted by a quote character. Nulls are represented by an empty quoted string " ". The number of quoted strings in the file should be the same as the number of columns in the table, `olLoad` gives an error otherwise.

- FixedAscii Format: Each row of the table is represented as a separate line in the file. Each line is separated by a carriage return and a line feed character on the Windows platform. Each line is of the same size. The datatype of a column governs its format or representation in the file. Nulls are represented by a string of *n* '\0' (null) characters, where *n* is the fixed size of the field. Table A–15 describes data representation for each data type. The total record length for each line in the file should be the same as the sum of field lengths (precision) of each column, otherwise `olLoad` returns an error.

Table A–15 Datatypes

Datatype	Description
`CHAR(n)`	Length of the field in *n* characters. Data is left aligned and padded with blanks on the right.
`VARCHAR(n)`	Length of the field in n characters. Data is left aligned. It is padded with a null byte ('\0').

Table A–15 (Cont.) Datatypes

Datatype	Description
`NUMERIC(p,s)`	The default mode: length of the field is p+1 characters if scale s is zero or is not present. Otherwise, the length of the field is (p+2) characters. The value is right aligned in the output field. Format is optional negative sign, followed by zeros if required, followed by significant digits. If there is no negative sign, then '0' instead, for example, Number(5,2) 12.3 -> ' 012.30' -12.3 -> '-012.30' 1.23 -> ' 001.23' -1.23 -> '-001.23' The custom mode: the field length is one less: p if scale is not present, or zero and p+1 otherwise. The actual number stored in the file is of type NUMERIC(p-1, s). Correspondingly, `olDump` gives an error trying to insert a number within the range of NUMERIC(p, s), but out of the range of NUMERIC(p-1, s). Therefore, the first character in the NUMERIC field must be '0' or '-'; `olLoad` gives an error otherwise.
`DECIMAL(p,s)`	The same as NUMERIC(p,s).
`INTEGER`	Length of the field is 11 characters. A negative sign or space followed by 10 digits. Leading digits are filled with zeros.
`SMALLINT`	Field length is 6 characters. Minus sign or space followed by 5 digits.
`FLOAT`	Field length is 23 characters. In Oracle mode, it is minus sign or space, followed by leading zeroes, followed by some number of digits, followed by dot, followed by some number of digits. For example: 0 -> ' 0000000000000000000000' -12.34 -> '-0000000000000000012.34' In SQL92 mode the E (exponent) is always present and there is only 1 digit before the decimal point. For example: 0 -> ' 00000000000000000000E0' -12.34 -> '-000000000000001.234E10'
`REAL`	The same format as for double precision except that the total field length is only 16 characters instead of 23.
`DOUBLE PRECISION`	Field length is 23 characters. Minus sign or space followed by 22 characters which are digits, dot, or E, floating point number followed by E, followed by the exponent digits. In Oracle mode, if the number is small enough to fit in the field without using the exponent, E is not used. In SQL92 mode, E is always used. There is always one meaningful digit before the floating point, except 0. For example, in SQL92 mode: 0 -> ' 00000000000000000000E0' -1.79E10 -> '-0000000000000001.79E10' 12 -> ' 000000000000000001.2E10' For example, in Oracle mode: 1.2E75 -> ' 000000000000000001.2E75' -1.33333 -> '-0000000000000001.33333' -1.79E10 -> '-0000000000017900000000'

Table A–15 (Cont.) Datatypes

Datatype	Description
`DATE`	In SQL92 mode: YYYY-MM-DD, 10 characters long, for example: October 1, 1999 -> 1999-10-01 In Oracle mode the date is dumped as timestamp. If it is not the default date format parameter, the date format corresponds to the specified date format string, for example: DATEFORMAT = "YYYYMMDD" October 1, 1999 -> 19991001
`TIME`	HH:MM:SS, 8 characters long, for example: 5:01:58 p.m. is 17:01:58
`TIMESTAMP`	Date format, space, time format, dot, 6 digits after dot (precision of microseconds), total length of 26 characters: YYYY-MM-DD HH:MM:SS.SSSSSS If it is not the default date format parameter, the timestamp format corresponds to the specified date format string. If no time is specified in the date format string, the time information in the timestamp is omitted when dumping into a file. Note: `TIMESTAMP WITH TIME ZONE` is not supported.

A.10.2.4 Limitations

Currently `olLoad` does not support the following features:

- Columns of the datatype Interval, Time with time zone, Timestamp with time zone, BLOB, and CLOB.
- Binary data is not supported.
- The only "var" type supported is varchar.

Glossary

Apache Server

The Apache Server is a public domain HTTP server derived from the National Center for Supercomputing Applications (NCSA).

Base Table

A source of data, either a table or a view, that underlies a view. When you access data in a view, you are really accessing data from its base tables.

Connected

Connected is a generic term that refers to users, applications, or devices that are connected to a server. The Mobile client for Web-to-go is "connected" when it is in online mode.

Database Object

A database object is a named database structure: a table, view, sequence, index, snapshot, or synonym.

Database Server

The Oracle database server is the third tier of the Mobile Server/Mobile Client Web model. It stores the application data.

Disconnected

Disconnected is a generic term that refers to users, applications, or devices that are not connected to a server.

Foreign Key

A foreign key is a column or group of columns in one table or view whose values provide a reference to the rows in another table or view. A foreign key generally contains a value that matches a primary key value in another table. See also "Primary Key".

Index

An index is a database object that provides fast access to individual rows in a table. You create an index to accelerate the queries and sorting operations performed against the table's data. You also use indexes to enforce certain constraints on tables, such as unique and primary key constraints.

Indexes, once created, are automatically maintained and used for data access by the database engine whenever possible.

Integrity Constraint

An integrity constraint is a rule that restricts the values that can be entered into one or more columns of a table.

Java Applets

Java applets are small applications that are executed in the browser that extend the functionality of HTML pages by adding dynamic content.

JDBC

JDBC (Java Database Connectivity) is a standard set of java classes providing vendor-independent access to relational data. Modeled on ODBC, the JDBC classes provide standard features such as simultaneous connections to several databases, transaction management, simple queries, manipulation of pre-compiled statements with bind variables, and calls to stored procedures. JDBC supports both static and dynamic SQL.

JavaServer Pages

JavaServer Pages (JSP) is a technology that enables developers to change a page's layout without altering the page's underlying content. JSP, which uses HTML and pieces of Java code to combine the presentation of dynamic content with business logic.

Java Servlets

Java servlets are protocol and platform-independent server-side components that are written in Java. Java servlets dynamically extend Java-enabled servers and provide a general framework for services built using the request-response paradigm.

Join

A relationship established between keys (both primary and foreign) in two different tables or views. Joins are used to link tables that have been normalized to eliminate redundant data in a relational database. A common type of join links the primary key in one table to the foreign key in another table to establish a master-detail relationship. A join corresponds to a WHERE clause condition in a SQL statement.

Master-Detail Relationship

A master-detail relationship exists between tables or views in a database when multiple rows in one table or view (the detail table or view) are associated with a single master row in another table or view (the master table or view).

Master and detail rows are normally joined by a primary key column in the master table or view that matches a foreign key column in the detail table or view.

When you change values for the primary key, the application should query a new set of detail records, so that values in the foreign key match values in the primary key. For example, if detail records in the EMP table are to be kept synchronized with master records in the DEPT table, the primary key in DEPT should be DEPTNO, and the foreign key in EMP should be DEPTNO. See also "Primary Key" and "Foreign Key".

MIME

MIME (Multipurpose Internet Mail Extensions) is a message format used on the Internet to describe the contents of a message. MIME is used by HTTP servers to describe the type of file being delivered.

MIME Type

MIME Type is a file format defined by Multipurpose Internet Mail Extension (MIME).

Mobile client for Web-to-go

The Mobile client for Web-to-go is the client tier of the Web-to-Go model. It contains the Mobile Server and Oracle Database Lite. Web-to-Go replicates the user applications and data to the Mobile device. When the user synchronizes, Web-to-Go replicates any data changes to the Oracle database.

Mobile Development Kit for Web-to-go

The Mobile Development Kit for Web-to-go enables application developers to develop and debug Web-to-go applications that consist of Java servlets, JavaServer Pages (JSP), or Java applets.

Mobile Server

The Mobile Server resides on the application server tier of the three-tier Web-to-go model and processes requests from the Mobile client for Web-to-go to modify data in the Oracle database server. The Mobile Server can be configured to run with the Oracle application server or as a standalone Mobile Server.

Mobile Server Repository

The Mobile Server repository is a virtual file system. It is a persistent resource repository that contains all application files and definitions of the applications.

ODBC

ODBC (Open Database Connectivity) is a Microsoft standard that enables database access on different platforms. You can enable ODBC support on the Mobile client for Web-to-go for troubleshooting purposes. ODBC support enables you to view the client's data, which is stored on the local Oracle Database Lite. To view this information, you can use Mobile SQL.

Oracle Database

The Oracle database is the database component of the Mobile Server.

Oracle Database Lite

Oracle Database Lite is a small footprint relational database.

Packaging Wizard

The Packaging Wizard enables administrators to package and publish Mobile applications to the Mobile Server repository. Developers can use the Packaging Wizard to create a new application or to edit an existing application definition.

Positioned DELETE

A positioned DELETE statement deletes the current row of the cursor. Its format is:

```
DELETE FROM table
   WHERE CURRENT OF cursor_name
```

Positioned UPDATE

A positioned UPDATE statement updates the current row of the cursor. Its format is:

```
UPDATE table SET set_list
   WHERE CURRENT OF cursor_name
```

Primary Key

A table's primary key is a column or group of columns used to uniquely identify each row in the table. The primary key provides fast access to the table's records, and is frequently used as the basis of a join between two tables or views. Only one primary key may be defined per table.

To satisfy a PRIMARY KEY constraint, no primary key value can appear in more than one row of the table, and no column that is part of the primary key can contain a NULL value.

Publication Item

A publication item is a SQL select statement that specifies which data subset a client can access. A publication item usually corresponds to a replica table on the client device. You can create publication items using the Mobile Server Admin API. This API contains Java functions that implement the publish/subscribe model. You can call the functions in this API from within Java programs as standard function calls.

Referential Integrity

Referential integrity is defined as the accuracy of links between tables in a master-detail relationship that is maintained when records are added, modified, or deleted.

Carefully defined master-detail relationships promote referential integrity. Constraints in your database enforce referential integrity at the database (the server in a client/server environment).

The goal of referential integrity is to prevent the creation of an orphan record, which is a detail record that has no valid link to a master record. Rules that enforce referential integrity prevent the deletion or update of a master record, or the insertion or update of a detail record, that creates an orphan record.

Replication

Replication is the process of copying and maintaining database objects in multiple databases that make up a distributed database system. Changes applied at one site are captured and stored locally before being forwarded and applied at each of the remote locations. Replication provides users with fast, local access to shared data, and protects the availability of applications because alternate data access options exist. Even if one site becomes unavailable, users can continue to query or even update the remaining locations.

Replication Conflict

Replication conflicts occur when contradictory changes to the same data are made.

Schema

A schema is a named collection of database objects, including tables, views, indexes, and sequences.

Sequence

A sequence is a schema object that generates sequential numbers. After creating a sequence, you can use it to generate unique sequence numbers for transaction processing. These unique integers can include primary key values. If a transaction generates a sequence number, the sequence is incremented immediately whether you commit or roll back the transaction.

Sequence Window

The sequence window contains a unique range of values. The range of values never overlaps with those of other clients. When a client uses all the values in the range of its sequence window, the Mobile client recreates the sequence with a new, unique range of values.

Snapshots

A snapshot is a subset of application data for a specific user. For each user, the publication (within the application) contains a SQL query that defines the information relevant to this user. This information is known as the snapshot.

The Mobile client retrieves the appropriate data from the Oracle database and downloads to the client when there is a connection to the back-end Oracle database. A snapshot can be a copy of an entire database table, or a subset of rows from the table, if specified within a parameterized SQL query. The first time a user synchronizes, the Mobile client automatically creates the snapshots on the client machine. Each subsequent time that a user synchronizes, the Mobile client either refreshes the snapshots with the most recent data or recreates them depending on the complexity of the snapshot.

SQL

SQL, or Structured Query Language, is a non-procedural database access language used by most relational database engines. Statements in SQL describe operations to be performed on sets of data. When a SQL statement is sent to a database, the database engine automatically generates a procedure to perform the specified tasks.

Synchronization

Synchronization is the process the Mobile client uses to replicate data between the Mobile client and the Oracle database. The Mobile client replicates the user applications and data to Oracle Database Lite when the user synchronizes. The Mobile client replicates any data changes made on the client to the Oracle database.

Synonym

A synonym is an alternative name, or alias, for a table, view, sequence, snapshot, or another synonym.

Table

A table is a database object that stores data that is organized into rows and columns. In a well designed database, each table stores information about a single topic (such as company employees or customer addresses).

Three-Tier Web Model

The three-tier Web model is an Internet database configuration that contains a client, a middle tier, and an Oracle database server. Web-to-go architecture follows the three-tier Web model.

Transaction

A set of changes made to selected data in a relational database. Transactions are usually executed with a SQL statement such as INSERT, UPDATE, or DELETE. A transaction is complete when it is either committed (the changes are made permanent) or rolled back (the changes are discarded).

A transaction is frequently preceded by a query, which selects specific records from the database that you want to change. See also "SQL".

Unique key

A table's unique key is a column or group of columns that are unique in each row of a table. To satisfy a UNIQUE KEY constraint, no unique key value can appear in more than one row of the table. However, unlike the PRIMARY KEY constraint, a unique key made up of a single column can contain NULL values.

View

A view is a customized presentation of data selected from one or more tables (or other views). A view is like a "virtual table" that allows you to relate and combine data from multiple tables (called base tables) and views. A view is a kind of "stored query" because you can specify selection criteria for the data that the view displays.

Views, like tables, are organized into rows and columns. However, views contain no data themselves. Views allow you to treat multiple tables or views as one database object.

Web-to-go

Oracle Web-to-go is a framework for the creation and deployment of Mobile, Web-based, database applications. Web-to-go contains a three-tier database architecture consisting of the Mobile client for Web-to-go, the Mobile Server and Oracle database. It is centrally managed from the server and Web-to-go applications can be run when Web-to-go connected to the server (online) or disconnected from the server (offline). When Web-to-go is disconnected from the back-end Oracle database, it caches data locally and synchronizes the data with the server when the client and server connect.

Workspace

The Mobile Server Workspace is a Web page that provides users with access to Web-to-go applications. Web-to-go generates the Workspace in the user's browser after the user logs in to Web-to-go. The Workspace displays icons, links, and descriptions of all applications that are available to the user. An application is available to the user after the administrator publishes it to the Web-to-go system and grants access privileges to the user.

Index