Oracle® Database Lite

Developer's Guide

Release 10.3

E12090-02

February 2010

Oracle Database Lite Developer's Guide Release 10.3

E12090-02

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 to all users, including users that are disabled. 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.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

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 for Designing Mobile Applications

The following sections provide an introduction to Oracle Database Lite 10*g* and an overview of the application development process:

- Section 1.1, "Introduction"
- Section 1.2, "Oracle Database Lite 10g Application Model and Architecture"
- Section 1.3, "Creating the Publish-Subscribe Model for Mobile Users"
- Section 1.4, "Mobile Development Kit (MDK)"
- Section 1.5, "Mobile Application Design"
- Section 1.6, "Supported Languages for Application Development"

1.1 Introduction

Oracle Database Lite 10*g* 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. The application requires a small, local database on the mobile device, whose content is a subset of data that is stored in the enterprise data server. This database can either be a SQLite database or the Oracle Lite database. Each database type stores data in files with an ODB extension. Modifications made to the local database by the application are reconciled with the back-end server data through data synchronization.

The Mobile client component in Oracle Database Lite is a preconfigured component to facilitate running a Mobile application. It contains synchronization and software components to manage the device.

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

- Web-to-Go applications can be built using Web technologies, such as servlet, Java Sever Pages (JSP), applet, HTML, and JDBC.
- 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 10*g* also offers a third way to develop Mobile database applications using the servlet based Web model called Web-to-Go.
- 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.

- 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 10*g* 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 10*g* 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 10*g* 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 10*g* 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, and form factors), many connectivity options (such as, LAN, Wireless LAN, cellular, and wireless data), 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 10*g* 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 10*g* 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 10*g* 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 to be running. You can use an existing test database as well. The Mobile Server stores its metadata in this database.

1.2 Oracle Database Lite 10*g* Application Model and Architecture

In the Oracle Database Lite 10*g* 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 Database Lite you receive snapshots, and, depending on your subscription parameter values, those snapshots are partitioned with data tailored for you.

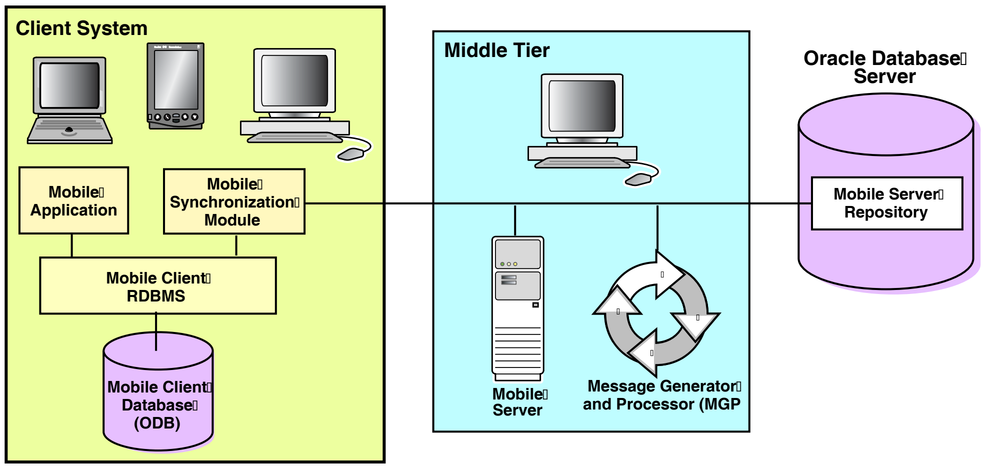
When a user synchronizes the Mobile client for the first time, the Mobile client creates the Mobile client database on the client machine for each subscription that is provisioned to the user. The Mobile client database could be a SQLite database or an Oracle Lite database, as set in the publication. 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 the Mobile client database 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 10*g* 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 10*g* applications.

Figure 1–1 Oracle Database Lite 10g Architecture



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

The following sections describe the separate components of Oracle Database Lite:

- Section 1.2.1, "Mobile Client Database"
- Section 1.2.2, "Mobile Sync"
- Section 1.2.3, "Mobile Server"
- Section 1.2.4, "Message Generator and Processor (MGP)"
- Section 1.2.5, "Mobile Server Repository"
- Section 1.2.6, "Device Manager"

1.2.1 Mobile Client Database

The Mobile client uses a client database, which can be either a SQLite database or Oracle Lite database, to store the relational data in one or more data files on the file system on the client. While the SQLite database is already installed on many client devices, you can install the Oracle Lite database on most any device from the Mobile Manager.

Both Mobile client databases are described in the following sections:

- Section 1.2.1.1, "SQLite Database"
- Section 1.2.1.2, "Oracle Lite Database"

1.2.1.1 SQLite Database

If you are using the SQLite database as the Mobile client database, if may already be installed on your device or you must install this independently. All details about the SQLite database are documented on the SQLite database Web site at `http://www.sqlite.org/`. After installing the SQLite database, install the SQLite Mobile client, which includes the Sync Engine for managing synchronization between the SQLite database and the back-end Oracle database.

1.2.1.2 Oracle Lite Database

The Oracle Database Lite database 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 2003/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 Lite RDBMS and the utilities are installed on your development machine. In a production system, when the Mobile Server installs Oracle Database Lite 10*g* applications, only the RDBMS, the Mobile Sync, and Mobile SQL applications are installed on the client machine.

Oracle Database Lite has its own JDBC driver on the client. With this driver, you can connect to the client Oracle Lite database.

1.2.2 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 authenticates locally, collects changes from the Oracle Lite database and sends them to the server, where the user is authenticated before the changes are uploaded. Mobile Sync runs on Windows 2003/XP, Windows CE/Windows Mobile, and Linux.

Use the `msync` executable for Mobile Sync.

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 (that is, 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.

1.2.3 Mobile Server

The installation of the Mobile Server requires an Oracle database to be running. You can use an existing test database as well. The Mobile Server stores its metadata in this database.

The Mobile Server is a Web middle-tier server that can exist on Windows, Solaris, HP-UX, AIX, and Linux. The Mobile Server uses the Oracle Containers for Java (OC4J) Web engine and provides the interface between the mobile infrastructure and the enterprise database. Most administration tasks are accomplished through the Mobile Server Web application—the Mobile Manager.

The Mobile Server provides the following features.

- application publishing
- application provisioning
- application installation and update
- data synchronization

The Mobile Manager application provides the capability to manage users, devices, publications and applications. This utility can provides the following:

- Monitors and manages synchronization between the client data store and the enterprise data store.
- Sends administrative commands to the Mobile clients. These commands capture data and logs from the client or instructs the client to carry out necessary tasks. For example, the Mobile Manager could send a command to a client to perform synchronization or to remove the entire client data store, if a device may have been compromised.

Note: If you wish, you can also accomplish the same tasks as the Mobile Manager with the Application Programming Interfaces (APIs).

As with any Web server tier, the Mobile Server may be configured within a Web farm for improved performance within the mobile infrastructure. This enables the use of a load balancer, such as the balancer included with Oracle Internet Application Server (OracleAS), or with one provided by a 3rd party vendor. The Mobile Server is designed to be fully integrated with OracleAS to take advantage of the features within OracleAS, such as the Oracle Internet Directory (OID) capabilities.

Note: As the Mobile Server is a Web-based environment, it is important to design for a proper security environment as would be done with any Web server.

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.

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.
2. Run the Mobile Sync (mSync) command on your Mobile device, which prompts for the Mobile username 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 msync or a custom command to synchronize your local database with the server database. This synchronization updates all applications that have changed.

1.2.4 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.

Note: The mobile infrastructure may allow for multiple Mobile Servers to be configured within a Web farm. However, there may only be one MGP application utilized for the entire Web farm.

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.

1.2.5 Mobile Server Repository

The Mobile Server Repository contains all the application data as well as all information needed to run the Mobile Server. The Mobile Repository contains the repository schema under which all the data mapping and internal tables utilized to maintain data synchronization exist. This schema also stores the application, application tables and its data published for use with a Mobile client.

The information is normally stored in the same database where the application data resides. 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.

The Mobile Repository contains some internal tables that the Mobile Server uses to perform its functions. You may query these tables to gain more details about the current state of the environment; however, most of the information needed from these tables is already accessible from the Mobile Manager. You should never alter any of the internal tables and their contents unless explicitly directed to by Oracle Support Services or Oracle Development.

Administration, backup, and recovery of the repository are no different then for any other Oracle database requiring standard Database Administrator (DBA) skills

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

1.2.6 Device Manager

The Device Manager manages client devices. On install of the Mobile client, the Device Manager registers a device with the Mobile Server. The Device Manager invokes the update executable after synchronization completes to determine if any mobile application updates are available, then downloads and installs these application updates to a Mobile client. You can request—through the Mobile Manager—that certain commands are invoked on the client. The Device Manager executes these commands. The Device Manager is responsible for most administrative actions between the Mobile Server and the Mobile client.

1.3 Creating the Publish-Subscribe Model for Mobile Users

ITo enable users to access their data, you need to first define the data in the snapshot. Then, subscribe the appropriate users to access only their data. On the client device, data is stored in a special type of relational table, called a snapshot table. A snapshot table behaves identical to a regular relational table, but has functionality that enables tracking changes made to the table.

A SQL query, called the publication item, can determine the record set that is downloaded to the snapshot table. The publication item is executed against the server database. The result set of the query defines the structure (columns) of the snapshot table on the client device as well as its contents.

A collection of publication items is called a publication, which corresponds to a single Oracle Lite database on a client device. All snapshot tables based that are based on publication items part of a single publication are stored in the same Oracle Lite database.

Oracle Database Lite operates within a publish-subscribe model. We use the example of the magazine as an effective way to explain the publish-subscribe model. A magazine is created with specific data that would be of interest to readers, such as sports, hunting, automobiles, and so on. Readers request a subscription for the specific magazine they feel would be in their interest to read. Once this subscription is created only the magazines to which the reader has been subscribed are sent to the reader.

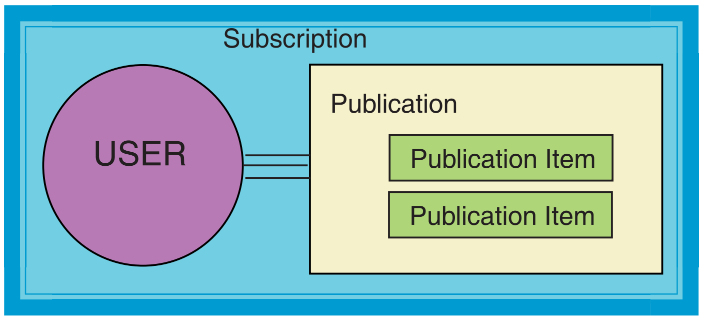
For Oracle Database Lite, the publication is the magazine, the publication items are the specific articles of data and the subscription is the granting of access to the publication for specific users. In the Oracle Database Lite 10*g* application model, each application defines its data requirements using a publication. Data subsets, known as publications items, are created and added to a publication. Application files are also uploaded to the same publication. Once these publications are deployed to the Mobile Server, any user may be granted a subscription to the publication.

Technically, a publication is like 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*.

As shown in Figure 1–2, a subscription defines the relationship between a user and a publication. 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 Database Lite you receive snapshots, and, depending on your subscription parameter values, those snapshots are partitioned with data tailored for you.

Subscription parameter values can be set by the administrator in order to tailor the snapshot data for each user.

Figure 1–2 Subscription Defines Relationship Between User and Publication



The subscription is the definition of how to retrieve data from the back-end database; the snapshot is the actual data that conforms to the definition within the subscription and which belongs to the user.

This process really forms a simple development cycle for mobile applications, as follows:

1. Create the publication and its publication items that contains the data subset for a particular application.
2. Grant users a subscription to a publication. This forms the specific dataset that is used on a Mobile client.
3. Develop and test the Mobile application to work with the specific data set.
4. Deploy the application to the Mobile Server and install it on the client.

Two of the more common questions and sources of confusion that comes up are what has to be done first:

1. Do you create the publication first or the publication items?

   It does not matter. You can create either the publication or the publication item first. Consider an article for a magazine. That article may have been written by a freelance author. The article exists before it belongs to any publication. The author submits this to two or three magazine publishers since it is relevant to the content they advertise. Two decide it is appropriate for the publication they are distributing currently while one does not include it since the content is not quite what their readers want.

2. Do you have to create a separate publication item for each publication?

   No, you can have one or more publication items in a publication.

Note: You can create publications with the Mobile Database Workbench or the Java APIs.

The following sections describes other pertinent information for publication items:

- Section 1.3.1, "Defining the Weight and Conflict Resolution for Publication Items"
- Section 1.3.2, "Behavior and Requirements for Primary Keys, Foreign Keys and Not Null Fields in Publication Items"

1.3.1 Defining the Weight and Conflict Resolution for Publication Items

The following important aspects of the publication item should be taken into account when you are designing your application:

- Weight—The publication item weight is used to control the order in processing publication items, which avoids conflicts. Changes made on the client are processed according to weight in order to prevent conflicts, such as foreign key violations. The weight determines what tables are applied to the enterprise database first. For example, the `scott.emp` table has a foreign key constraint to the `scott.dept` table. If a new department number is added to the `dept` table and a new record utilizing the new department number were added to the `emp` table, then the transaction would be placed in the error queue if the new record utilizing the new department in the `emp` table was applied to the repository before the new department in the `dept` table was applied. To prevent the violation of the foreign key constraint on the enterprise server, you set the `dept` snapshot to a weight of 1 and the `emp` snapshot to a weight of 2, which applies all updates to the `dept` table prior to any updates to the `emp` table as the lower weight is always processed first.
- Conflict Resolution—In the same scenario, what if someone already updated the enterprise server with the new department number? This causes a conflict when the client attempts to synchronize with the new department that utilizes the same number. To handle this, conflict resolution may be set to either "client wins" or "server wins". If set to "server wins", then the setting on the server takes precedence to the setting on the client. The client transaction is sent to the error queue. However, if "client wins" is set, then the new department number from the client overrides the setting on the server.

1.3.2 Behavior and Requirements for Primary Keys, Foreign Keys and Not Null Fields in Publication Items

Only primary keys and not null fields are replicated down to the client. Publication items require a primary key field or, as in the case of a view, primary key hints.

If a foreign key needs to be applied to the client, then the script for the foreign key needs to be added to the publication, so that it will be executed when the client synchronizes the first time. You can set the script for the foreign key within either the MDW scripts section or the API.

Constraints are not the only type of script that may be executed on the client. The script could execute any valid SQL DDL statement on the client.

1.4 Mobile Development Kit (MDK)

Before you develop an application using Oracle Database Lite 10*g*, you should install the Oracle Database Lite 10*g* 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 4.3, "Installing Oracle Databse Lite" in the *Oracle Database Lite Getting Started Guide*.

The Oracle Database Lite 10*g* Mobile Development Kit includes the following components.

- Oracle Database Lite RDBMS—A lightweight, object-relational database management system, including Mobile SQL (`msql.exe`). Mobile SQL is written in Java. It requires the Java runtime environment JRE to be installed on your system before you can use it.
- Mobile Database Workbench (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 that includes the executable (`msync.exe`) and the Java wrapper for it.
- 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 4.3, "Installing Oracle Database 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.

When you install the MDK, it performs the following:

- 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 6.3, "Synchronize or Execute Applications on the Mobile Client" in the *Oracle Database Lite Client Guide*.

- 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
  ```

1.4.1 Mobile SQL (mSQL)

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

- Create Oracle Lite 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. If the Oracle Lite database was created in the Mobile Server environment, then the default username is `SYSTEM` and the password is the same as the Mobile user password.

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 C.1, "The mSQL Tool in the *Oracle Database Lite Client Guide* for details.

Note: UTF8 SQL Scripts are not supported in mSQL.

1.4.2 Using the Mobile Database Workbench

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 6, "Using Mobile Database Workbench to Create Publications".

1.4.3 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".

1.5 Mobile Application Design

Before you start to design your Mobile application, it is important to read the following sections to understand the differences between an enterprise application and the mobile application as well as the choices you have in designing your application:

- Section 1.5.1, "Steps for Designing Your Mobile Application"
- Section 1.5.2, "Application Programming Interfaces"
- Section 1.5.3, "Application Deployment into the Mobile Environment"

1.5.1 Steps for Designing Your Mobile Application

With a proper design, you can avoid the most common causes for mobile project failure, not meeting the needs of the business, poor performance, or issues occurring within a production environment. Proper design of the Mobile System includes the infrastructure and the mobile application. Without proper design, a mobile architecture could end up costing more then it saves.

The following assumption is one of the most common misconceptions for taking an enterprise application and incorporating it into a mobile component:

The mobile application is a scaled down version of the enterprise application.

By taking an existing enterprise application, you may intend to provide the same functionality in remote or disconnected locations. Since the enterprise application has already undergone thorough requirements gathering, design, development, testing, and successful implementation, you may assume that it will automatically work seamlessly as a mobile application and so do not test it in this environment. This assumption may lead to project failure.

For example, take an enterprise form-based client/server application. You have a client connecting through a middle-tier connecting to a database on the back-end. Taking this to a mobile infrastructure, such as with Oracle Database Lite, adds a completely new tier that did not exist within the original infrastructure. The Mobile Server tier introduces new concerns, such as the following:

- Security—There is now a system in the infrastructure that potentially gives any outsider access to the entire organization if proper security configuration and implementation is not performed.
- Bandwidth—The Mobile Server may become a bottleneck for all remote locations without the implementation of a Web farm.
- Scalability—The applications are performing synchronization of hundreds to millions of records, which is not the same as providing static Web pages to a large number of users. A system that is fine for serving static Web pages may not be capable of servicing hundreds of users performing synchronization.

A complete redesign of the system specifications may be in order.

You may also need to re-evaluate the original design of the enterprise application. The following lists a few design considerations for the mobile application:

- Memory—An application designed, tested, and implemented on a multiprocessor system with several gigabytes of memory will not perform the same on a standard PC with only a single processor and maybe 512 megabytes of memory.
- Resource Limitations—Several years ago, limitations of available resources made the usage of data types an extreme concern. The storage space saved by using a small integer over an integer was crucial due to limited memory available. With advances in memory and system resources, this has not been a concern to most modern developers. Now, the mobile infrastructure brings resource limitations back to the list of chief concerns for the design and development of mobile applications. One of the most significant of these limitations is the bandwidth available for the mobile client. If a Mobile client is only able to synchronize over a cell phone network, you may not wish to bring a million records down to a client that only needs a few thousand records. This decision impacts the synchronization performance, as well as the costs associated with the synchronization. If the mobile client was only utilized to collect data, then you can create a data collection queue for synchronization and avoid the whole download phase of synchronization.
- Use of Indexes—You use indexes for avoiding full-table scans. So, if you use the same data subset originally designed for a Windows machine down on a client device and do not use an index, then the performance may be adversely effected. Oracle Database Lite uses two types of scans for queries: full table scans and index based scans.

Thus, we recommend the following steps:

- Section 1.5.1.1, "Read the Documentation Before Design"
- Section 1.5.1.2, "Gather Mobile Requirements"
- Section 1.5.1.3, "Proof of Concept"

- Section 1.5.1.4, "Prototype"
- Section 1.5.1.5, "Design for Data Subsets"
- Section 1.5.1.6, "Design for Indexing"
- Section 1.5.1.7, "Design for Sequences"
- Section 1.5.1.8, "Design for Synchronization"
- Section 1.5.1.9, "Design for Administration"
- Section 1.5.1.10, "Design for the Language Utilized for Handheld Devices"

1.5.1.1 Read the Documentation Before Design

To ensure that you develop your applications correctly, we recommend that you start with a light scan of all of the documentation followed by detailed reading of pertinent areas related to the specific project that are to be undertaken.

1.5.1.2 Gather Mobile Requirements

Gathering requirements is the key to successful project development; a mobile project is no exception.

Mobile users have a specific task they perform. Gather the specific requirements of each mobile user type that will use the mobile application. For example, an insurance application may have two types of mobile users:

- The insurance investigator may only drive to locations to photograph an incident for an insurance claim before any required evidence is altered or vanishes. This user needs a mobile application that captures and uploads the images of the incident.
- A claims agent may later be sent to the site to process a claim for the client. This user needs more details for the client, such as their account information and what they are covered for in order to properly process a claim.

Each user requires a different subset of the enterprise data and functionality within the mobile application.

1.5.1.3 Proof of Concept

Proof of concept testing determines if the Oracle Database Lite mobile architecture meets the needs of the business before architectural design starts.

When an aspect of the mobile solution is determined to meet a requirement, a quick proof of concept test ensures that the requirement may be satisfied by the potential solution. There is a difference in what can be done in a mobile environment versus what can be accomplished in the enterprise environment.

1.5.1.4 Prototype

Prototyping may be seen as too costly to utilize for the proper design of a mobile architecture and application. However, consider that a prototype may be anything from sketching out a design on paper to writing a full application prototype.

You may use one of the sample applications included with the Oracle Database Lite product, such as the transport demo, to gain a full understanding of the infrastructure and the design considerations it involves. Take the application through the setup of the environment, the creation of users, the deployment of the mobile application to multiple clients, the synchronization of multiple clients, and the testing of updates made to the application. This provides a better understanding of the infrastructure as actual hands on experience.

1.5.1.5 Design for Data Subsets

Data subsetting reduces data to be downloaded, which has a direct impact on performance.

What is Data Subsetting? Data subsets are a crucial part of the design for a mobile application. Data sub-setting is accomplished through variables that limit the data to a specific requirement, such as by region or department. An easy way to conceptualize data subsets is to consider a subset that is limited through a `WHERE` clause that uses bind variables—also known as snapshot template variables—to limit the snapshot data.

When you set up your publication item, you may have set up an input parameter that defines what snapshot of data is to be retrieved for this user. For example, if you have an application that retrieves the customer base for each sales manager, the application needs to know the sales manager's identification number to retrieve the data specific to each manager. Thus, if you set up each sales manager as a unique user and set their identification number in the data subsetting screen, then the application is provided that unique information for retrieving data.

1.5.1.6 Design for Indexing

Oracle Database Lite maintains two basic methods for accessing data within a table: full table scan and index-based access. The performance of an application can be affected if the design does not include the appropriate indexing. Use indexes to avoid full table scans.

You can execute the Explain Plan and SQL Trace capabilities within the product to determine the access method used by a statement and to determine if indexes are being utilized. Analyze your statements throughout the design and development cycles to ensure that they are being designed and developed for maximum efficiency.

1.5.1.7 Design for Sequences

Sequences guarantee uniqueness of a value, such as a primary key. Design how the sequences are generated within the mobile infrastructure. For example, if the enterprise database generates a sequence number and the Mobile client generates the same sequence number a conflict with the data occurs and causes an error.

Native sequences may be formed specifically for the Mobile clients. These sequences would never populate on the enterprise database itself, so there is no risk of a conflict occurring. This works well when data updates only occur from the Mobile clients and input to the database does not come from any other source. However, it is often necessary to have the sequences generated by both the database and the clients. To accomplish this, sequences must be designed so the database uses a range separate from the range used by the clients. For example, you could define the sequences where the database uses all odd numbers and the clients uses all even numbers.

You must also design sequences for the Mobile clients, so that each client uses a unique range of values without any two clients using the same range. For this you specify the sequence range for each client, such as sequences 1 through 1000 for client A and sequences 1001 through 2000 for client B. Using these ranges for the sequence numbers prevents each client from using the same sequence number as used by another client.

For full details on sequences, see Section 6.6, "Create a Sequence".

1.5.1.8 Design for Synchronization

If you are using the Mobile option, synchronization holds the mobile infrastructure together.

Analyze all of the data needed by the mobile user, as follows:

- Most snapshots are created where the data can be modified on either the client or the server, where the modifications are propagated to the other side through synchronization.
- If any snapshots require only the ability to read the data—that is, all modifications to the data are made on the server-level and not by the user—then create read-only snapshots.
- If all or a majority of the users use the same read-only snapshots, then create a cached user that shares the read-only data across multiple clients.

Analyze the type of synchronization that is appropriate for the user's needs, as described below:

- For optimal performance, use fast refresh for all publications, if appropriate.
- Only design publication items for a complete refresh if the following is true:
  - If it is absolutely critical for all changes to be processed and applied to the data store immediately.
  - If it is critical that any enterprise updates are immediately brought down to the client.

  Note: The complete refresh is the most resource intensive method and should only be utilized after full consideration of the performance hit is analyzed. Only time critical publication items should be specified for a complete refresh synchronization type.

- If a mobile user is only performing data collection and it is not necessary for server updates to be brought down to the user, then implement a push-only synchronization model for those publication items. For more information, see Section 2.13.2, "Creating Data Collection Queues for Uploading Client Collected Data".
- When a mobile user only requires specific table to be updated or synchronized, perform a selective synchronization methodology limiting the synchronization process to specific tables or specific publications.

1.5.1.9 Design for Administration

When designing for administration, you may create standardized groups and assign users to these groups. It is much easier to administer groups of users then it is to administer several thousand users each as individuals. The situations where groups simplify your administration is when your organizational architecture has data and the environment is structured into a hierarchy, such as by region or department. In this situation, all properties, access settings, and template variables may be set for groups of users and not the users individually.

1.5.1.10 Design for the Language Utilized for Handheld Devices

Any language selected for handheld development has limitations not existing in their fuller counterparts. The mobile application must be designed to meet these limitations. For example, when developing with WinCE utilizing the Microsoft Compact

Framework with C# or VB.Net, forms are not maintained the same way as they are in the full framework when running on a normal PC. Without designing for this limitation, it would be easy to accidentally leave forms sitting in the background that should have been closed when the application exits. This prevents crucial memory from being freed and also prevents any application updates from downloading and installing correctly.

When you properly research and design for the device platform, everything done on the device may be performed on the larger non-mobile applications. Thus, you can reuse and update code simply as it uses the same coding for both environments. Designing in the reverse may result in functionality being utilized in the non-mobile applications that does not have a matching functionality in the mobile version.

1.5.2 Application Programming Interfaces

When you are developing your application, you may decide that you want to control more aspects of Oracle Database Lite within your application—rather than relying on user interaction. In this case, you can use the Oracle Database Lite Application Programming Interfaces (APIs). Almost any task performed by the tools and utilities included with Oracle Database Lite may also be accomplished with the APIs. Some of the more advanced functionality within the product is only available through the use of the APIs. Except for the synchronization APIs which are provided for most languages utilized for application development, most of the APIs are Java interfaces that must be developed with the Java programming language.

The most common APIs utilized and their uses are as follows:

- Synchronization APIs: These APIs provide all of the basic synchronization functionality that is found within the mSync utility. The advantages of using these APIs are that the synchronization process can be fully integrated within the actual Mobile application. The APIs also provide the ability for a push-only synchronization, which allows Mobile clients to only upload data skipping the downloading of new data or applications. The push-only model is useful when bandwidth is limited and when the client just collects data—that is, it is not necessary for a remote client to have updated data from the enterprise.
- Consolidator APIs: The Consolidator APIs provide administrative functionality for creating users, setting the user properties, working with applications, and so on. You can automate common administration tasks and speed up some of the administration tasks required, such as the creation of a large amount of users. The only limitation is that application and user settings are not displayed in the Mobile Manager Web administration tool as these APIs directly access the Mobile Repository.
- Mobile Resource Manager APIs: The Mobile Resource Manager APIs also provides administration functionality for users and applications; however, this API actually updates the Mobile Manager administration tool as well as the repository. This utility may be used to create users, set user access, set the user template variables, and many other tasks.
- Device Manager APIs: The Device Manager APIs provide the ability customize the management of devices. These APIs may be used to gather information on devices, send commands to devices, register devices, and so on.

1.5.3 Application Deployment into the Mobile Environment

Deployment of applications includes setting up the server system so that end users can easily install and use the applications. Mobile applications are deployed to the Mobile Server.

Deployment consists of the following steps:

1. Create the publication with the Mobile Database Workbench (MDW). See Section 1.4.2, "Using the Mobile Database Workbench" for more information.
2. Publishing the application to the server includes installing all the components for an application on the Mobile Server with the Packaging Wizard tool. See Section 1.4.3, "Using the Packaging Wizard" for details.
3. Provisioning the applications to the Mobile users through the Mobile Manager, which is a GUI interface for the Mobile Server. This phase includes determining user accesses to applications with a specified subset of data. The Mobile Manager can 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 the following:
   - Multiple Mobile device client technologies—such as, operating systems, form factors, and so on.
   - Multiple connectivity options—such as, LAN, Wireless LAN, cellular, wireless data, and other technologies.
   - Multiple server configuration options.

   When testing, pay particular attention 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 10*g* client, the application code, and the initial Oracle Lite 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 given 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 10*g* provides a tool for offline instantiation.

After deployment, both the application and the data schema may change because of enhancements or defect resolution. The Mobile Server manages 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.

1.6 Supported Languages for Application Development

Oracle Database Lite is an integrated framework that simplifies the development, management, and deployment of mobile applications. For a list of the supported on mobile platforms, operating systems, and hardware, see Chapter 3, "Requirements before Installation or Development" in the *Oracle Database Lite Getting Started Guide*.

Platform	Programming Languages
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.Net 2003 or 2005 for development. If you use Visual Studio.Net 2005, you must install the ODBC 3.5 driver. See Section 5.1.3, "ODBC" for details.
PocketPC	C, C++, Visual Basic, Java applications (no Servlet or JSP support), SODA, ADO.NET. You can use Visual Studio.Net 2005 for development. If you use Visual Studio.Net 2005, you must install the ODBC 3.5 driver. See Section 5.1.3, "ODBC" for details.
Linux	Java, C, C++
Symbian OS on Nokia and Motorola	C, C++

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 1.6.1, "Native Applications"
- Section 1.6.2, "Standalone Java Applications"
- Section 1.6.3, "Web Applications"

1.6.1 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 5.1.2, "JDBC" and Section 5.1.3, "ODBC" for more information on accessing the database with either of these interfaces.

See Section 5.1, "Data Access APIs" for more information on C and C++.

1.6.2 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 5, "Application Development" and Chapter 7, "JDBC Programming" in the *Oracle Database Lite Client Guide* for more information.

1.6.3 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 2003 or XP. 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 5.5, "Developing Mobile Web-to-Go Applications".

2 Synchronization

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

The following sections describe how synchronization functions between the Mobile clients 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.

2.1 How Oracle Database Lite Synchronizes

When most people think of synchronizing data, they think of their Palm Pilot. When you hit the synchronization button for the Palm Pilot, any changes are added to the database of information on the Windows machine immediately. This is not the case for Oracle Database Lite, in that the synchronization is used for multiple clients—rather than a single user. In order to accommodate a large number of concurrent users, the application tables on the back-end database cannot be locked by a single user. Thus, the synchronization process involves using queues to manage the information between the Mobile clients and the application tables in the database.

Oracle Database Lite uses a synchronization model that maintains data integrity between the Mobile Server and the Mobile client. In addition, the synchronization is asynchronous and that as a result, change propagation is not immediate. The benefit, however, is that the clients do not stay connected for long while the changes are being applied.

You can specify if the synchronization occurs automatically or by manual request. For more details, see Section 2.1.3, "Deciding on Automatic or Manual Synchronization".

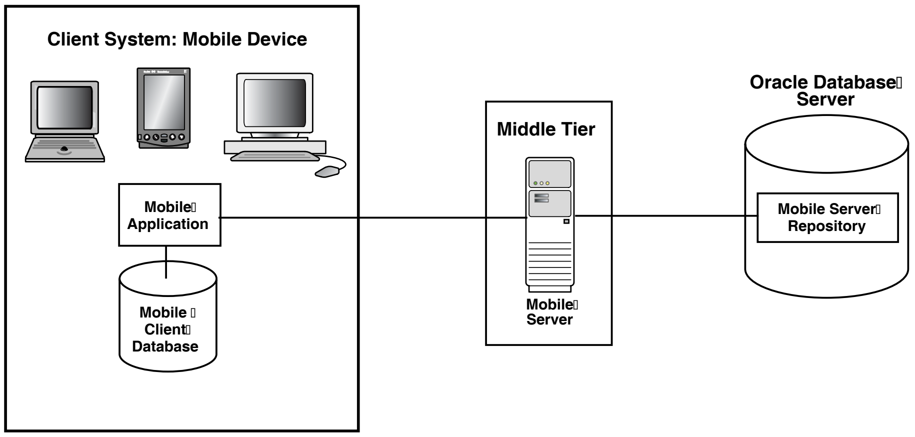
A simplified view of Mobile synchronization is as follows:

- On the client—The Mobile application communicates through the Mobile Sync Server 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.
- On 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 2–1 illustrates the architecture for Oracle Database Lite 10*g* applications.

Note: This section describes how the synchronization is performed across several components and enterprise tiers to complete successfully. For more details on each component, see Section 1.2, "Oracle Database Lite 10g Application Model and Architecture".

Figure 2–1 Oracle Database Lite 10g Architecture

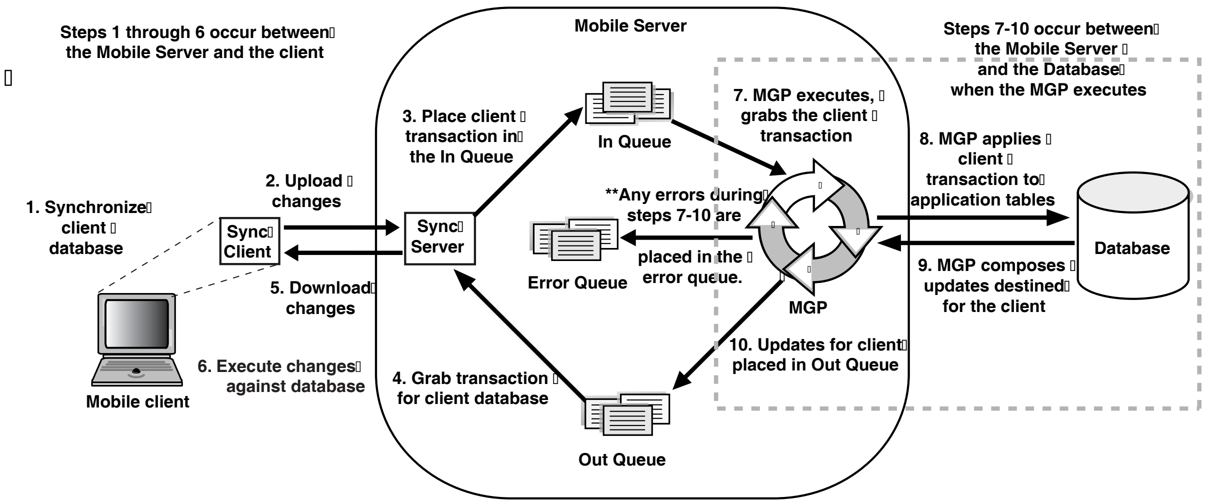


Note: Web-to-Go clients have one additional component, a light weight HTTP listener that is not shown in the diagram.

Oracle Database Lite uses the Mobile Server to replicate data between the Mobile clients with their client databases and the application tables, which are stored on a back-end Oracle database.

Thus, the more detailed description of how synchronization is performed within the separate components of Oracle Database Lite is demonstrated by Figure 2–2.

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

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 PDA, a Windows platform client, or a supported Linux platform client.
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 database.
6. Mobile client applies all changes for client database. For Oracle Lite Mobile clients, 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 6.3, "Synchronize or Execute Applications on the Mobile Client"in the *Oracle Database Lite Client Guide*.

---

7. All transactions uploaded by all Mobile clients are gathered by the MGP out of the In-Queue. The MGP executes independently and periodically based upon an interval specified in the Job Scheduler in the Mobile Server.
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. If any conflicts occur during this phase, most are resolved by the MGP or by the conflict resolution rules. If the conflict cannot be resolved, the transaction is moved into the Error Queue. See Section 1.3.1, "Defining the Weight and Conflict Resolution for Publication Items" for more information.

Note: The behavior of the apply/compose phase can be modified. See Section 6.1.1, "Defining Behavior of Apply/Compose Phase for Synchronization" in the *Oracle Database Lite Administration and Deployment Guide* for more information.

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, where the Sync Server downloads these updates to the client on the next client synchronization.

Overall, synchronization involves two parties: the Mobile client using the Sync Client/Server to upload and download changes and the MGP process interacting with the queues and the application tables to apply and compose transactions. These are displayed separately in the Data Synchronization section of the Mobile Manager.

The following sections describe synchronization activity:

- Section 2.1.1, "Oracle Lite Mobile Client Database Created on First Synchronization"
- Section 2.1.2, "Using Multiple Databases for Application Data"
- Section 2.1.3, "Deciding on Automatic or Manual Synchronization"
- Section 2.1.4, "Deciding on Synchronization Refresh Option"
- Section 2.1.5, "Synchronizing to a File With File-Based Sync"
- Section 2.1.6, "How Downloaded Data is Processed on the Mobile Client"
- Section 2.1.7, "How Updates Are Propagated to the Back-End Database"
- Section 2.1.8, "How Modified BLOB Data is Synchronized"

2.1.1 Oracle Lite Mobile Client Database Created on First Synchronization

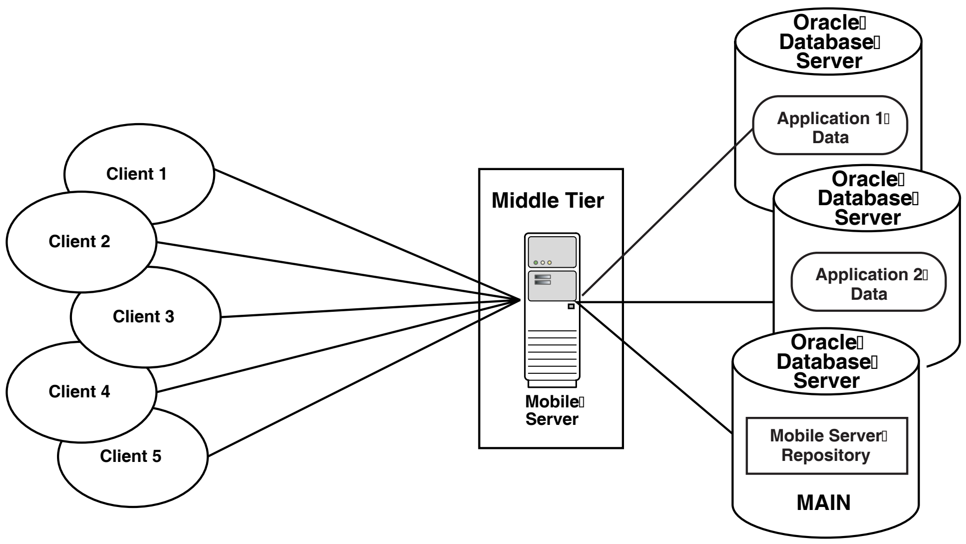
When a user synchronizes an Oracle Lite Mobile client for the first time, the Mobile client creates an Oracle Lite database on the client machine for each subscription that is provisioned to the user. The Mobile client then creates a snapshot in this database for each publication item contained in the 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 the data in the Oracle Lite database are captured by the snapshots. When the connection to the Mobile Server is available, the changes can be synchronized with the Mobile Server.

2.1.2 Using Multiple Databases for Application Data

By default, the Mobile repository metadata and the application data are stored on the same database. However, if for performance or other reasons, you may store application data on a separate database other than the main database where the Mobile repository exists. In this manner, the Mobile repository exists on the main database and the data for one or more applications may exist on the main database or another database of your choosing.

Figure 2–3 Separating Application Data from Mobile Repository



You can register one or more databases to host the application data. Once registered, you can specify during publication creation where to host the application data. Synchronization is executed on a per publication basis rotating through the databases. For more information, see Section 3.2, "Register a Remote Oracle Database for Application Data".

2.1.3 Deciding on 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.

Manual Synchronization may be initiated, as follows:

- The user initiates the Oracle Database Lite 10*g* Mobile Synchronization (mSync) application directly.
- The application programmatically invokes the Mobile Synchronization API.
- Oracle Database Lite has a Web-based application model, known as Web-to-Go. For this type of application, the synchronization option can be defined from the Web-to-Go workspace to synchronize the data. See Section 1.6, "Supported Languages for Application Development" for more information.

Automatic Synchronization can be configured 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.

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.

Synchronization is closely tied to how you define the snapshot for your application. See Section 1.3, "Creating the Publish-Subscribe Model for Mobile Users" for a description of a snapshot and its components. One of the components is a publication item. If you want automatic synchronization, you define it at the publication item level.

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. Alternatively, you can stop the automatic synchronization to allow the manual synchronization to occur. If you choose to do this, then after the manual synchronization is finished, re-start the automatic synchronization.

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

Table 2–1 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.
Objects synchronized	All	The following objects are not synchronized by an automatic synchronization: sequences, DDL scripts, resources—such as Java stored procedures—indexes and automatic synchronization rules and conditions.

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.

To enable high priority synchronization for automatic synchronization, Oracle Database Lite adds a hidden column (`MSG$PRIO`) to all automatic synchronization snapshots to designate if this data has a higher priority for synchronization if conditions are right. If users need to indicate that a particular record is high priority, they can set the column value to (0). Then the sync agent automatically schedules a high priority synchronization for the transaction that contains this record.

2.1.4 Deciding on Synchronization Refresh Option

How or when data changes are applied to either the Mobile Server or the Mobile client depends upon the synchronization refresh option at the publication item level. Synchronization refresh options may ease the cost burden for resources, such as wireless connectivity, bandwidth and network availability, personnel loss of time during the synchronization process, and so on.

Oracle Database Lite employs synchronization refresh options that may be utilized to synchronize data between the Oracle enterprise database and the Mobile client. With the following Oracle Database Lite refresh options, you can maintain data accuracy and integrity between the Oracle database and Mobile client:

- Section 2.1.4.1, "Fast Refresh"

- Section 2.1.4.2, "Complete Refresh"
- Section 2.1.4.3, "Queue-Based Refresh"
- Section 2.1.4.4, "Forced Refresh"

2.1.4.1 Fast Refresh

The most common method of synchronization is a fast refresh publication item where changes are uploaded and downloaded by the client. Meanwhile, the MGP periodically collects changes uploaded by all clients and applies them to the back-end Oracle database tables. Then, the MGP composes new data, ready to be downloaded to each client during the next synchronization, based on pre-defined subscriptions.

2.1.4.2 Complete Refresh

During a complete refresh, all data for a publication is downloaded to the client. For example, during the 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.

The complete refresh model is resource intensive as all aspects of synchronization are performed. This model should only be utilized for snapshots/publication items where it is an absolute requirement.

2.1.4.3 Queue-Based Refresh

The developer creates their own queues to handle the synchronization data transfer. There is no synchronization logic created with a queue-based refresh; instead, the synchronization logic is implemented solely by the developer. A queue-based publication item is ideally suited for scenarios that require synchronization to behave in a different manner than normally executed. For instance, data collection on the client; all data is collected on the client and pushed to the server.

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.

2.1.4.4 Forced Refresh

This is not a refresh option; however, we discuss it here because it is often mistaken for a refresh option—specifically, it is often confused with the complete refresh option. The Forced Refresh is a one-time execution request made from within Mobile Manager, the GUI interface for the Mobile Server. The forced refresh option may result in a loss of critical data on the client.

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

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

2.1.5 Synchronizing to a File With File-Based Sync

There are times when you do not have network access to the Mobile Server, but there is a way you can use removable media to transport a file between the Mobile Server and the client. In this instance, you may want to use File-Based Sync, which saves all transactions in an encrypted file either for the upload from the client for the Mobile Server or the download from the Mobile Server for the client.

Once saved within the encrypted file, the file is manually transported and copied onto the desired recipient—whether Mobile client or Mobile Server. This file is uploaded and the normal synchronization steps are performed. The only difference is that the interim transmission of the data is through a file copied to the correct machine—rather than transmitted over a network.

For full details on file-based synchronization, see Section 6.8, "Synchronizing to a File with File-Based Sync" in the *Oracle Database Lite Administration and Deployment Guide*. To enable and perform file-based synchronization through the APIs, see Chapter 4, "Invoking Synchronization in Applications With the Mobile Sync APIs".

2.1.6 How Downloaded Data is Processed on the Mobile Client

The client processes the downloaded data. By default, the steps taken to process the received data on the client is as follows:

1. Process each publication item
2. Commit
3. Process each DDL statement
4. Commit

> **Note:** The acknowledgment is sent only in the subsequent synchronization.

Alternatively, the configuration could effect how the data is processed on the client. The following are two scenarios where the commit may occur before all of the publication items are processed:

- Auto commit—Invoke the `auto_commit_count` feature for publication items that use manual synchronization. If this parameter is set to 0, Oracle Database Lite calls a commit at the end of processing for each publication.

  If you set the `auto_commit_count` variable in the `[SYNC]` section in the `polite.ini` file and the number of records in a transaction is greater than the `auto_commit_count`, then a commit is issued at that time. This occurs only on the first synchronization, complete refresh, or a low memory condition. However, for the fast refresh option, if the auto commit is performed, then a data mismatch will happen between client and server.

  > **Note:** During synchronization when `auto_commit_count` is set, the user should not add, update, or delete a database record.

  If this parameter is set to 1000, Oracle Database Lite calls commits for every 1000 inserts. This value should be more than 100. The default value for `auto_commit_count` is 250 records on WINCE; this variable is not valid on WIN32 and LINUX platforms.

- Low memory—If the client is on a WIN32 device and available memory is running low, then an auto commit is performed.

  If the client is on a WinCE device, then if memory is getting low, the synchronization throws and error and exits error. In this situation, the commit is not performed.

2.1.7 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 2.4.1.7.2, "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 2.4.1.7.2, "Using Table Weight".

Note: This order of executing operations can cause constraint violations. See Section 2.10, "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 Sync Server only takes the final result and makes a single insert.

2.1.8 How Modified BLOB Data is Synchronized

If you update the contents of a BLOB belonging to a particular row in particular table, then the row is not synchronized. The row only stores a handle to the BLOB object—not the object itself. So, the handle does not change. Therefore, the changes made to the BLOB are not propagated to the server. To cause the BLOB contents to be synchronized to the server, the row must be explicitly updated by the application. Any update of the row should be sufficient—even updating the column to the same value.

2.2 Enabling Automatic Synchronization

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 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 client 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 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 2–2 Automatic Synchronization

Steps for Automatic Synchronization	See the Following for Details
The developer enables the publication item to use automatic synchronization.	Section 2.2.1, "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 2.2.2, "Enable/Disable Automatic Synchronization on the Mobile Client"
You can configure under what rules the automatic synchronization occurs.	Section 2.2.3, "Define the Rules Under Which the Automatic Synchronization Starts"
The server can notify the client of data waiting for download.	Section 2.13.3, "Selecting How/When to Notify Clients of Composed Data"
The client application can request status of the outcome of an automatic synchronization.	Section 2.2.6, "Notify Application on Completion of Automatic Synchronization Cycle"

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

- Section 2.2.1, "Enable Automatic Synchronization at the Publication Item Level"
- Section 2.2.2, "Enable/Disable Automatic Synchronization on the Mobile Client"
- Section 2.2.3, "Define the Rules Under Which the Automatic Synchronization Starts"
- Section 2.2.4, "Setting Data as High Priority in Automatic Synchronization"
- Section 2.2.5, "Enable the Server to Notify the Client to Initiate a Synchronization to Download Data"
- Section 2.2.6, "Notify Application on Completion of Automatic Synchronization Cycle"

2.2.1 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 6.4, "Create a Publication Item" for details of how to enable synchronization in a publication item using MDW or Section 2.4.1.3, "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 synchronization 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, unless the synchronization in in the middle of a large data commit or is executing over a slow network . If the automatic synchronization does not stop immediately due to these exceptions and you want the manual synchronization to start immediately, 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 2.2.2, "Enable/Disable Automatic Synchronization on the Mobile Client" for full details.

2.2.2 Enable/Disable Automatic Synchronization on the Mobile Client

Automatic synchronization is enabled by default if a publication is enabled for automated synchronization. However, you may turn on and off automatic synchronization—either temporarily or permanently—as follows:

- Start/Stop—If you decide to stop automatic synchronization temporarily; then, if you restart the client, automatic synchronization is restarted. Use start/stop for temporarily stopping automatic synchronization.
- 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—until enabled again.

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

- Section 2.2.2.1, "Start or Stop Automatic Synchronization"
- Section 2.2.2.2, "Enable or Disable Automatic Synchronization"
- Section 2.2.2.3, "Sync Control APIs to Start or Enable Automatic Synchronization"

2.2.2.1 Start or Stop Automatic Synchronization

Use the start/stop methods to temporarily start or stop the Sync Agent. The user may want to stop the Sync Agent for many reasons, such as aborting an automatic synchronization that may be running longer than desired, freeing up system resources, or de-fragmenting or backing up an Oracle Lite database. If the automatic synchronization is not re-started with the control API, then it restarts by either a manual synchronization or the device management agent after reboot.

By default, if you are using the mSync GUI to initiate a synchronization, the underlying code stops and restarts the automatic synchronization for you, as described below:

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.

You can stop or 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 6.4.2, "Start, Stop, or Get Status for Automatic Synchronization" in the *Oracle Database Lite Administration and Deployment Guide* for more details.

2.2.2.2 Enable or Disable Automatic Synchronization

The start and stop methods only control the automatic synchronization temporarily. Use the disable method to fully disable automatic synchronization, so that it is not restarted when a device is powered on. Turn the automatic synchronization engine back on with the enable method. You can enable or disable automatic synchronization in one of three ways, as described below:

- Enable/disable automatic synchronization using the Client Workspace. For details, see Section 6.4.1.1.2 "Configuration Tab" in the *Oracle Database Lite Client 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` executable UI. See the following sections for details: Section E.4.1, "SYNC_AGENT" or Section 6.4.2, "Start, Stop, or Get Status for Automatic Synchronization," which are located in the *Oracle Database Lite Administration and Deployment Guide*.
- Enable/disable automatic synchronization by invoking the enable or disable methods, as described in Section 2.2.2.3, "Sync Control APIs to Start or Enable Automatic Synchronization".

2.2.2.3 Sync Control APIs to Start or Enable Automatic Synchronization

The following sections describe how to start/stop or enable/disable automatic synchronization from the Sync Control API:

- Section 2.2.2.3.1, "C/C++ Sync Control APIs to Start or Enable Automatic Synchronization"
- Section 2.2.2.3.2, "C# Sync Control APIs to Start or Enable Automatic Synchronization"
- Section 2.2.2.3.3, "Java Sync Control APIs to Start or Enable Automatic Synchronization"

2.2.2.3.1 C/C++ Sync Control APIs to Start or Enable Automatic Synchronization The following sections describe the Sync Control APIs for C/C++ applications.

To start or stop the Sync Agent, use the following APIs:

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

To enable or disable the Sync Agent, use the following APIs:

```
typedef struct _olSyncOpt {
olBool bDisable;
} olSyncOpt;
olError olGetSyncOptions(olSyncOpt *opt);
olError olSetSyncOptions(const olSyncOpt *opt);
```

The `olGetSyncOptions` and `olSetSyncOptions` methods take a pointer to the `olSyncOpt` structure as a parameter. The `olSyncOpt` structure contains the `bDisable` boolean, which is `true` if the Sync Agent is disabled.

To enable the Sync Agent, perform the following:

```
olSyncOpt opt.bDisable = FALSE;
olSetSyncOptions(&opt);
```

To disable the Sync Agent, perform the following:

```
olSyncOpt opt.bDisable = TRUE;
olSetSyncOptions(&opt);
```

Use `olGetSyncOptions` method to retrieve the current value of the `bDisable` boolean.

2.2.2.3.2 C# Sync Control APIs to Start or Enable Automatic Synchronization The following `BGSyncControl` class has the following methods:

- `Start`—Start automatic synchronization that was previously stopped.
- `Stop`—Stop automatic synchronization. Normally, this is used to stop automatic synchronization before a manual synchronization is invoked. Then, use the `start` method to restart automatic synchronization.
- `Enabled` property—Set `Enable` to `TRUE` to enable automatic synchronization that was previously disabled. Set to `FALSE` to disable automatic synchronization. Even if the client is restarted, automatic synchronization is not enabled unless you enable synchronization. This property returns a boolean where true states that automatic synchronization is enabled and false that it is disabled.

```
public class BGSyncControl
{
  public void Start();
  public void Stop(int timeout);
public bool Enabled{}
}
```

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

2.2.2.3.3 Java Sync Control APIs to Start or Enable Automatic Synchronization The following `BGSyncControl` class has the following methods:

- `start`—Start automatic synchronization that was previously stopped.
- `stop`—Stop automatic synchronization. Normally, this is used to stop automatic synchronization before a manual synchronization is invoked. Then, use the `start` method to restart automatic synchronization.
- `enable`—Enables automatic synchronization that was previously disabled.
- `disable`—Disables automatic synchronization on a client. Even if the client is restarted, automatic synchronization is not enabled unless you enable synchronization.
- `isEnabled`—Returns a boolean where true states that automatic synchronization is enabled.

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

2.2.3 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
```

So, if an event occurs, the conditions are evaluated. If the conditions are valid, then the synchronization occurs; if the conditions are not met, then the synchronization is queued until the conditions are valid.

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

By default, the client will automatically synchronize after every commit if network bandwidth is detected. To change these defaults, you must modify the Client commit condition in the publication rules and the event rule for network bandwidth in either the platform or publication rules.

If after defining these rules and publishing the application, you want to modify the rules, you can do so through MDW. However, you must perform a manual synchronization. The manual synchronization restarts the automatic Sync Agent, which will then use the new rules The new settings will NOT be applied during automatic synchronization.

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

- Section 2.2.3.1, "Configure Publication-Level Automatic Synchronization Rules"
- Section 2.2.3.2, "Configure Platform-Level Automatic Synchronization Rules"

2.2.3.1 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 6.5, "Define the Rules Under Which the Automatic Synchronization Starts" ; to add publication-level automatic synchronization rules with the API, see Section 2.4.1.4, "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 2–3 describes the publication level events for automatic synchronization. The lowest value that can be provided is 1.

Table 2–3 Automatic Events for the Publication

Events	Description
Client commit	For Oracle Lite Mobile client only. Upon commit to the Oracle Lite database, the Oracle Lite 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. You must modify this rule if you do not want the automatic synchronization to occur after every commit.
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**.

2.2.3.2 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 2.2.3.2.1, "Event Rules for Platforms".
   - Conditions—See Section 2.2.3.2.2, "Condition Rules for Platforms".
   - Network settings—See Section 2.2.3.2.3, "Network Configuration for the Client Platform".

2.2.3.2.1 Event Rules for Platforms Table 2–4 shows the platform events for automatic synchronization.

Table 2–4 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.
AC Power	As soon as AC power is detected, then synchronization is automatically triggered.

Table 2–4 (Cont.) Automatic Event Rules for the Client Platform

Event	Description
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.

2.2.3.2.2 Condition Rules for Platforms Table 2–5 shows the platform conditions for automatic synchronization.

Table 2–5 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.

2.2.3.2.3 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. For dial-up, Oracle Database

Lite can automatically establish the network connection before initiating the synchronization.

2.2.4 Setting Data as High Priority in Automatic Synchronization

Oracle Database Lite adds a hidden column (MSG$PRIO) to all automatic synchronization snapshots to designate if this data has a higher priority for synchronization if conditions are right. If users need to indicate that a particular record is high priority, they can set the column value to (0). Then sync agent automatically schedules a high priority synchronization for the transaction that contains this record.

2.2.5 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 2.13.3, "Selecting How/When to Notify Clients of Composed Data".

2.2.6 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 2–6 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_CHANGED`: 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

Table 2–6 (Cont.) The Sync Message Variables

Variable	Description
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.

2.2.7 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 2–7 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.

2.3 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 2.3.1, "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 2.3.2, "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 2.8, "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 2.6, "Customize the Compose Phase Using MyCompose".

2.3.1 Creating a Snapshot Definition Declaratively

Use the Mobile Database Workbench (MDW), a GUI based tool of Oracle Database Lite—described fully in Chapter 6, "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 6, "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. In addition, for Oracle Lite Mobile client, create the Oracle Lite database automatically.

2.3.1.1 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 2.3.1.1.1, "Read-only Snapshots"
- Section 2.3.1.1.2, "Updatable Snapshots"
- Section 2.3.1.1.3, "Refresh a Snapshot"
- Section 2.3.1.1.4, "Snapshot Template Variables"

2.3.1.1.1 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 6.9.2, "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.

Also, because read-only does not upload any data from the client, there are no conflicts. Thus, when specified within MDW, you will only be able to select Custom for conflict resolution.

2.3.1.1.2 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 6.9.2, "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 2–8 shows each refresh method type and whether it is updatable or read-only depending on primary key or virtual primary key:

Table 2–8 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

2.3.1.1.3 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 6.4, "Create a Publication Item" and Section 2.9, "Understanding Your Refresh Options"

2.3.1.1.4 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 6.4.1, "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 5.5, "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 2–9 provides a sample set of snapshot query values specified for separate users.

Table 2–9 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 2–10 provides another sample snapshot query value.

Table 2–10 Snapshot Query Value for User Names

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

2.3.2 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 2.4, "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 6, "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.

2.4 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 Mobile clients. Basic functions, such as creating publication items and publications, can be implemented easily using the Mobile Development Workspace (MDW). See Chapter 6, "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 6, "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 2.4.1, "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.

2.4.1 Defining a Publication With Java Consolidator Manager APIs

While we recommend that you use MDW (see Chapter 6, "Using Mobile Database Workbench to Create Publications") 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 6, "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 2.4.1.1, "Create the Mobile Server User"
2. Section 2.4.1.2, "Create Publications"
3. Section 2.4.1.3, "Create Publication Items"
4. Section 2.4.1.4, "Define Publication-Level Automatic Synchronization Rules"
5. Section 2.4.1.5, "Data Subsetting: Defining Client Subscription Parameters for Publications"
6. Section 2.4.1.6, "Create Publication Item Indexes"

7. Section 2.4.1.7, "Adding Publication Items to Publications"
8. Section 2.4.1.8, "Creating Client-Side Sequences for the Downloaded Snapshot"
9. Section 2.4.1.9, "Subscribing Users to a Publication"
10. Section 2.4.1.10, "Instantiate the Subscription"
11. Section 2.4.1.11, "Bringing the Data From the Subscription Down to the Client"
12. Section 2.4.1.12, "Modifying a Publication Item"
13. Section 2.4.1.13, "Callback Customization for DML Operations"
14. Section 2.4.1.14, "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`

2.4.1.1 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. If you want members to be created for this device, perform the tasks described in Section 2.4.1.1.2, "Create Member Users for Sharing Application and Data on Device".
4. 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.

2.4.1.1.1 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");
```

2.4.1.1.2 Create Member Users for Sharing Application and Data on Device

The Member user enables you to define multiple users on a device using the same application and data. Each member is created and associated with a user. After the user grants access to the member to its data, each member can log on with his/her username and password and can access the data as defined by the user. This enables multiple people, such as shift workers, to use the same device, without needing to use the same username and password or the same access privileges. For more details on how the member is used and instructions on how to create the member with the Mobile Manager, see Section 5.5.3, "Adding New Members and Associating Them With Users" in the *Oracle Database Lite Administration and Deployment Guide*.

To create a member user and associate it with a user with the APIs, perform the following:

1. After you have created the `MobileResourceManager` object described in Section 2.4.1.1, "Create the Mobile Server User", you can issue the createUser method to create a member. Provide member name, password, full name and 'M' as the privilege parameter to designate the creation of a member.
2. After you have created all members, associate one or more members with the user by executing the `MobileResourceManager.associateMemberToUser`

method. Provide the existing user name and member name or a Vector containing all member names.

After you complete creation of all users and members, end the transaction and close the connection.

The following detail other methods that support member users:

- Drop a member with the `dropUser` method.
- Retrieve all members in a vector with the `getUserMembers` method.
- Remove one or more members from a user with the `removeMemberFromUser` method.
- Batch initialize several members with the `enableMembers` method. You can disable multiple members with the `disableMembers` method. To see which members are initialized, use the `initializeMembers` method, which returns a hashtable that indicates if the user was initialized successfully or not.

  For full details on batch initialization, see **Member Initialization** in Section 6.4.1.1.2, "Configure the Mobile Client" in the *Oracle Database Lite Client Guide*.

  The syntax for these methods are as follows:

  ```
  Hashtable oracle.lite.web.client.enableMembers(String userName,
   String password, Vector members) throws ResException
  Hashtable oracle.lite.web.client.disableMembers(String userName,
   String password, Vector members) throws ResException
  Hashtable oracle.lite.web.client.initializeMembers(String userName,
   String password, Hashtable members) throws ResException
  ```

  Where:

  - `userName`—Device owner username.
  - `password`—Password of the device owner.
  - `members`—For the `enable/disableMembers` methods, this contains a vector with the list of member users that need to be enabled/disabled. For the `initializeMembers` method, this contains a hashtable containing a list of usernames and password represented as (key, value) pairs. The key is the member name and value contains the member password.
  - Returns a hashtable containing a list of usernames and status codes represented as (key, value) pairs. The key is the member-name and value contains the status code indicating if the user was enabled, disabled, or is initialized.

2.4.1.2 Create Publications

A subscription is an association 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.
3. Create the publication with the `createPublication` method, which creates an empty publication. An example of the `createPublication` method syntax is as follows:

   ```
   createPublication(
   ```

```
java.lang.String name,
java.lang.String db_inst,
int client_storage_type,
java.lang.String client_name_template,
java.lang.String enforce_ri,
int dev_types_flg)
```

The `createPublication` method can have some of the following input parameters:

- **name**—A character string specifying the new publication name.
- **db_inst**—Null, unless you are using a registered database for application data, as described in Section 3.2, "Register a Remote Oracle Database for Application Data". If using a registered database, provide the application database name in this field.
- **client_storage_type**—An integer specifying the client storage type for all publication items in the new publication. If you are defining a publication exclusively for a SQLite Mobile client, you must specify the `Consolidator.SQLITE_CREATOR_ID` as the storage type.

  Other values are `Consolidator.DFLT_CREATOR_ID and Consolidator.OKPI_CREATOR_ID`.
- **client_name_template**—A template for publication item instance names on client devices. This parameter contains the following predefined values:
  - `%s`—Default.
  - `DATABASE.%s`—Causes all publication items to be instantiated inside an OKAPI database with the name `DATABASE`.
  - `SFT-EE_%s`—Must be used for Satellite Forms-based applications.
- **enforce_ri**—Reserved for future use. Use null or an empty string.
- **dev_types_flg**—Specifies which device types or platforms the publication supports. The default flag is set to `Consolidator.DEV_FLG_GEN`, which includes all device platforms. If a publication is for more than one platform, use the sum of the platform flags.

  Available platforms are as follows:
  - SQLite DB: "SQLite LINUX", "SQLite WCE", "SQLite WIN32", "SQLiteJava"
  - Oracle Lite DB: "EPOC", "LINUX", "WCE", "WIN32", "WTG"

  To retrieve the device flag for a platform, call the `getPlatformDevFlg` function. The syntax for this function is as follows:

  ```
  int getPlatformDevFlg(java.lang.String platform)
  ```

  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", null
             Consolidator.SQLITE_CREATOR_ID, "OrdersODB.%s", null);
```

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

2.4.1.3 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.

Note: You can create a publication using MDW. To see more details on publications and publication items, refer to Section 6.4, "Create a Publication Item".

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

- Column data: When you specify column data in 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 3.8, "Datatype Conversion Between the Oracle Server and the Oracle Lite Database".

  Also, the publication item query must select primary keys in the same order as they are defined in the base table.
- Automatic or Manual Synchronization: Whether the publication item is to be synchronized 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.
- If you are using a registered database for application data, as described in Section 3.2, "Register a Remote Oracle Database for Application Data".

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

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. However, if you are using a SQLite Mobile client, then you need to set the `setDfltColOptions` flag to false, as SQLite does not support the same SQL functions as Oracle. If `setDfltColOptions` is set to true (default) when the publication item is created, synchronization automatically uses the default clause from Oracle meta data, which is not supported by SQLite. Alternatively, you can execute the `ConsolidatorManager.setPubItemColOption` method to set a supported SQLite expression.
- 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.db_inst = null;          // Provide registered db instance name or null
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);
```

2.4.1.3.1 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 2.9, "Understanding Your Refresh Options" for more information.
- All base tables of the view must be published.

2.4.1.4 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 2–11, " 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 2–11 Automatic Synchronization Rule Info Objects

Rule Info Object	Description
`RULE_MAX_DB_REC_ri`	For Oracle Lite Mobile clients only. 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.
`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 2.4.1.4.1, "Retrieve All Publications Associated with a Rule"
- Section 2.4.1.4.2, "Retrieve Rule Text"
- Section 2.4.1.4.3, "Check if Rule is Modified"
- Section 2.4.1.4.4, "Remove Rule"

2.4.1.4.1 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);
```

2.4.1.4.2 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 );
```

2.4.1.4.3 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 );
```

2.4.1.4.4 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.

2.4.1.5 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.

2.4.1.6 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 the snapshot table for a publication item. If you do not want the primary index, you must explicitly drop it from the publication item.

If you want to create and associate other indexes on any columns in your application tables in the publication item, then use the `createPublicationItemIndex` method. You can drop an index from the publication item and from the snapshot table with the `dropPublicationItemIndex` 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");
```

Note: All indexes created by this API can be viewed within the `CV$ALL_PUBLICATIONS_INDEXES` view.

2.4.1.6.1 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 is an index based off of the primary keys.
- U - Unique enforces the unique constraint on the indexed columns, which ensures that duplicate values will not exist in the columns being indexed.
- I - Regular does not provide the UNIQUE constraint on the indexed columns.

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.

2.4.1.7 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 2.4.1.12, "Modifying a Publication Item" for details on how to change the definition.

2.4.1.7.1 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 2.11, "Resolving Conflicts with Winning Rules" for more information.

2.4.1.7.2 Using Table Weight

Table weight is an integer associated with publication items that determines in what order the transactions for all publication items within the publication are processed. For example, if three publication items exist—one that contains SQL to modify the `emp` table, one that modifies the `dept` table, and one that modifies the `mgr` table, then 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 the publication item that contains the `dept` table, table weight of 2 to the publication item that contains the `mgr` table, and table weight of 3 to the publication item that contains the `emp` table. In doing this, you ensure that the publication item that contains the master table `dept` is always processed first, followed by the publication item that modifies the `mgr` table, and lastly by the publication item that modifies the `emp` table.

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.

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 publication items when you add it to the publication.

2.4.1.8 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. For full details of what a sequence is and how Oracle Database Lite creates them, see Section 6.6, "Create a Sequence".

If you do not want to use MDW to create a sequence, then use the Consolidator Manager API 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. All of the same behavior exists for the Consolidator Manager APIs as are available through MDW.

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* for a complete listing of the APIs to define and administrate sequences.

2.4.1.9 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");
```

2.4.1.10 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 2.4.1.5, "Data Subsetting: Defining Client Subscription Parameters for Publications" for more information.

2.4.1.11 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.

2.4.1.12 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 3.6, "Facilitating Schema Evolution" and the `alterPublicationItem` method definition in the *Oracle Database Lite API Specification* for more information.

2.4.1.13 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..)
```

Note: You can use the `generateMobileDMLProcedure` to generate the procedure specification for a given publication item. This specification can be used as a starting point in creating your own custom DML handling logic in a PL/SQL procedure. See the *Oracle Database Lite API Specification* for more information.

The parameters for customizing a DML operation are listed in Table 2–12:

Table 2–12 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.

2.4.1.13.1 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.

2.4.1.14 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. For using a restricting predicate, see Section 1.2.10 "Priority-Based Replication" in the *Oracle Database Lite Troubleshooting and Tuning Guide*.

2.5 Client Device Database DDL Operations

The first time a client synchronizes, Oracle Database Lite automatically creates the snapshot tables for the user subscriptions on the Mobile client. For the Oracle Lite Mobile client, the Oracle Lite database is also created. 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.

2.6 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 `ConsolidatorManager` APIs to register and load the customized class.

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

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

1. Section 2.6.1, "Create a Class That Extends MyCompose to Perform the Compose"
2. Section 2.6.2, "Implement the Extended MyCompose Methods in the User-Defined Class"
3. Section 2.6.3, "Use Get Methods to Retrieve Information You Need in the User-Defined Compose Class"
4. Section 2.6.4, "Register the User-Defined Class With the Publication Item"

2.6.1 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.

2.6.2 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 2.6.2.1, "Implement the needCompose Method"
- Section 2.6.2.2, "Implement the doCompose Method"
- Section 2.6.2.3, "Implement the init Method"
- Section 2.6.2.4, "Implement the destroy Method"

Note: To retrieve information, use the methods described in Section 2.6.3, "Use Get Methods to Retrieve Information You Need in the User-Defined Compose Class".

2.6.2.1 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, Connection rmt_conn, String clientid)
 throws Throwable
```

The parameters for the `needCompose` method are listed in Table 2–13:

Table 2–13 needCompose Parameters

Parameter	Definition
`conn`	Database connection to the Main Mobile Server repository.
`rmt_conn`	Database connection to the remote database for application. Set to NULL if the base tables are on the Main database where the Mobile repository exists. For details on remote databases, see Section 3.2, "Register a Remote Oracle Database for Application Data".
`clientid`	Specifies the client that is being composed.

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(Connection conn, Connection rmtConn, 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 2.6.3, "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.

2.6.2.2 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, Connection rmt_conn,
   String clientid) throws Throwable
```

The parameters for the `doCompose` method are listed in Table 2–14:

Table 2–14 doCompose Parameters

Parameter	Definition
`conn`	Database connection to the Main Mobile Server repository.
`rmt_conn`	Database connection to the remote database for application. Set to NULL if the base tables are on the Main database where the Mobile repository exists. For details on remote databases, see Section 3.2, "Register a Remote Oracle Database for Application Data".
`clientid`	Specifies the client that is being composed.

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, Connection rmtConn, String clientid)
              throws Throwable {
        int rowCount = 0;

        Connection auxConn = rmtConn;
        if(auxConn == null)
            auxConn = rmtConn;
        String[][] baseTables = getBaseTables();
        String baseTableDMLLogName =
            getBaseTableDMLLogName(baseTables[0][0], baseTables[0][1]);
        String baseTablePK =
            getBaseTablePK(baseTables[0][0], baseTables[0][1]);
        String pubItemDMLTableName = getPubItemDMLTableName();
        String pubItemPK = getPubItemPK();
        String mapView = getMapView(clientid);

        Statement st = auxConn.createStatement();
        String sql = null;

        // insert
        sql = "INSERT INTO " + pubItemDMLTableName + " SELECT " + baseTablePK +
             ", DMLTYPE$$ FROM " + baseTableDMLLogName;

        rowCount += st.executeUpdate(sql);

        st.close();
        return rowCount;
    }
```

This code uses subsidiary methods discussed in Section 2.6.3, "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.

2.6.2.3 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 2–15:

Table 2–15 init Parameters

Parameter	Definition
`conn`	Database connection to the Main Mobile Server repository.

2.6.2.4 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 2–16:

Table 2–16 destroy Parameters

Parameter	Definition
`conn`	Database connection to the Main Mobile Server repository.

2.6.3 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 2.6.3.1, "Retrieve the Publication Name With the getPublication Method"
- Section 2.6.3.2, "Retrieve the Publication Item Name With the getPublicationItem Method"
- Section 2.6.3.3, "Retrieve the DML Table Name With the getPubItemDMLTableName Method"
- Section 2.6.3.4, "Retrieve the Primary Key With the getPubItemPK Method"
- Section 2.6.3.5, "Retrieve All Base Tables With the getBaseTables Method"
- Section 2.6.3.6, "Retrieve the Primary Key With the getBaseTablePK Method"
- Section 2.6.3.7, "Discover If Base Table Has Changed With the baseTableDirty Method"
- Section 2.6.3.8, "Retrieve the Name for DML Log Table With the getBaseTableDMLLogName Method"
- Section 2.6.3.9, "Retrieve View of the Map Table With the getMapView Method"

2.6.3.1 Retrieve the Publication Name With the getPublication Method

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

```
public String getPublication()
```

2.6.3.2 Retrieve the Publication Item Name With the getPublicationItem Method

The `getPublicationItem` method returns the publication item name.

```
public String getPublicationItem()
```

2.6.3.3 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 2–17:

Table 2–17 getPubItemDMLTableName View Structure Parameters

Parameter	Definition
`PubItemPK`	The value returned by `getPubItemPK()`

Table 2–17 (Cont.) getPubItemDMLTableName View Structure Parameters

Parameter	Definition
`DMLTYPE$$`	This can have the values 'I' for insert, 'D' for delete, or 'U' for Update.

2.6.3.4 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
```

2.6.3.5 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
```

2.6.3.6 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 2–18:

Table 2–18 getBaseTablePK Parameters

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

2.6.3.7 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 2–19:

Table 2–19 baseTableDirty Parameters

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

2.6.3.8 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 2–20:

Table 2–20 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 2–21:

Table 2–21 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.

2.6.3.9 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 2–22:

Table 2–22 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.

2.6.4 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.

2.7 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 2.7.1, "Customize What Occurs Before and After Every Phase of Each Synchronization"
- Section 2.7.2, "Customize What Occurs Before and After Compose/Apply Phases for a Single Publication Item"

2.7.1 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 2.7.2, "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 and any remote database that has publication items that are relevant for the customization. For more information on remote databases, see Section 3.2, "Register a Remote Oracle Database for Application Data".

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 2.7.1.1, "NullSync"
- Section 2.7.1.2, "BeforeProcessApply"
- Section 2.7.1.3, "AfterProcessApply"
- Section 2.7.1.4, "BeforeProcessCompose"
- Section 2.7.1.5, "AfterProcessCompose"
- Section 2.7.1.6, "BeforeProcessLogs"
- Section 2.7.1.7, "AfterProcessLogs"
- Section 2.7.1.8, "BeforeClientCompose"
- Section 2.7.1.9, "AfterClientCompose"
- Section 2.7.1.10, "BeforeSyncMapCleanup"
- Section 2.7.1.11, "AfterSyncMapCleanup"
- Section 2.7.1.12, "Example Using the Customize Package"

- Section 2.7.1.13, "Error Handling For CUSTOMIZE Package"

2.7.1.1 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);
```

2.7.1.2 BeforeProcessApply

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

```
procedure BeforeProcessApply;
```

2.7.1.3 AfterProcessApply

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

```
procedure AfterProcessApply;
```

2.7.1.4 BeforeProcessCompose

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

```
procedure BeforeProcessCompose;
```

2.7.1.5 AfterProcessCompose

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

```
procedure AfterProcessCompose;
```

2.7.1.6 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;
```

2.7.1.7 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;
```

2.7.1.8 BeforeClientCompose

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

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

2.7.1.9 AfterClientCompose

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

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

2.7.1.10 BeforeSyncMapCleanup

For every publication item, Oracle Database Lite maintains a map table, where the MGP inserts the DML operations to be carried out on the Oracle Lite database or new records to be inserted in the case of a complete refresh. At the end of the every synchronization session, the map tables are cleaned up where all old entries are deleted.

During this cleanup, if the connection properties are not ideal, then you may have performance issues. The callbacks added before and after the map cleanup operation enable you to optimize the connection properties and revert back to old connection properties after the operation is complete.

The `BeforeSyncMapCleanup` procedure is called at the beginning of the cleanup; the `AfterSyncMapCleanup` procedure is called after cleanup is finished. You can configure the connection settings can be changed in the `BeforeSyncMapCleanup` and reverted back in the `AfterSyncMapCleanup` procedure. These methods are invoked only once during the synchronization cycle.

The properties you can manage in these callback procedures are as follows:

- Any session level hints
- You can set the `OPTIMIZER_INDEX_CACHING` and `OPTIMIZER_INDEX_COST_ADJ` session parameters, as follows:
  - `ALTER SESSION SET OPTIMIZER_INDEX_CACHING=0;`
  - `ALTER SESSION SET OPTIMIZER_INDEX_COST_ADJ=100;`

> **Note:** In the `CONSOLIDATOR` section of the `webtogo.ora` file, you may want to modify the `MAX_U_COUNT` parameter before the synchronization starts.
>
> The `MAX_U_COUNT` parameter controls the number of SQL statements that are executed together in a SQL batch statement while performing the map cleanup. The default value for the `MAX_U_COUNT` parameter is 256. However, if the value is 256 during the map cleanup, then a maximum of 256 SQL statements can be executed together in a batch. Modify this parameter and restart the Mobile Server to enable a larger batch of SQL statements to be processed during map cleanup.

2.7.1.11 AfterSyncMapCleanup

The `AfterSyncMapCleanup` procedure is called at the end of the map cleanup. If you set any parameters in the BeforeSyncMapCleanup callback, you can set them back to the original settings in this procedure. See Section 2.7.1.10, "BeforeSyncMapCleanup" for more information.

2.7.1.12 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.

2.7.1.13 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.

2.7.2 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.

Note: If you are using a remote database for application data, then the callbacks must be defined on the same database as the application. See Section 3.2.4, "Using Callbacks on Remote Databases" for more details.

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);`

2.8 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 2–1) 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 2.8.1, "Starting Synchronization Upload and Download Phases With C or C++ Applications"
- Section 2.8.2, "Starting Synchronization Upload and Download Phases With Java Applications"
- Section 2.8.3, "Starting Synchronization Upload and Download Phases With the ADO.NET Provider"

2.8.1 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.

2.8.2 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.

2.8.3 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 5.1.4, "ADO.NET" for full details.

2.9 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 client 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 2.9.1, "Fast Refresh"
- Section 2.9.2, "Complete Refresh for Views"
- Section 2.9.3, "Queue-Based Refresh"
- Section 2.9.4, "Forced Refresh"

2.9.1 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.

2.9.2 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. On Oracle Lite Mobile clients, 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.

2.9.3 Queue-Based Refresh

You can create your own queues. 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 2.13, "Customizing Synchronization With Your Own Queues" for more information.

2.9.4 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.

2.10 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.

Note: For more details on database constraints in Oracle Database Lite, refer to Section 1.10, "Database Constraints" in the *Oracle Database Lite SQL Reference*.

The following sections detail each constraint and what issues you must take into account:

- Section 2.10.1, "Synchronization And Database Constraints"
- Section 2.10.2, "Primary Key is Unique"
- Section 2.10.3, "Foreign Key Constraints"
- Section 2.10.4, "Unique Key Constraint"
- Section 2.10.5, "Not Null Constraint"
- Section 2.10.6, "Generating Constraints on the Mobile Client"

2.10.1 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 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 2.4.1.7.2, "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 2.10.3.2, "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 Sync 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 2.10.6, "Generating Constraints on the Mobile Client".

2.10.2 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 Mobile clients.

For more information on the sequence number, see Section 2.4.1.8, "Creating Client-Side Sequences for the Downloaded Snapshot".

2.10.3 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 2.10.3.1, "Set Update Order for Tables With Weights"
- Section 2.10.3.2, "Defer Constraint Checking Until After All Transactions Are Applied"

2.10.3.1 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 2.4.1.7.2, "Using Table Weight".

2.10.3.2 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;
```

2.10.4 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 reponsibility 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 2.4.1.8, "Creating Client-Side Sequences for the Downloaded Snapshot" for more information.

2.10.5 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.

2.10.6 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 the Mobile 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.

2.10.6.1 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.

2.11 Resolving Conflicts with Winning Rules

When you have a conflict, you need to determine which party wins. The following are the settings that you can choose for conflict resolution on the server:

- 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.
- Two separate clients update the same row.
- Two clients insert a row with the same primary key.
- One client deletes a row that a second client updates.

> **Note:** In the case where two clients conflict, then the client whose data gets applied first effectively becomes the server and the other client becomes the client in resolving this conflict.

- 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 2.10, "Synchronizing With Database Constraints" for more information.

On the server, synchronization errors and conflicts 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 resolve the errors, and then re-execute or purge transactions from the error queue using either of the following:

- Resolve errors and conflicts in the error queue using the Mobile Manager GUI—See Section 6.10.4.3, "Viewing Transactions in the Error Queue" 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.

- Resolve errors and conflicts programmatically with the Consolidator Manager API. You can access the Mobile Server error queue tables directly and customize the conflict rules, as described in the following sections:
  - Section 2.11.1, "Resolving Errors and Conflicts on the Mobile Server Using the Error Queue"
  - Section 2.11.2, "Viewing Client-Side Synchronization Conflicts from Automatic Synchronization"
  - Section 2.11.3, "Customizing Synchronization Conflict Resolution Outcomes"

2.11.1 Resolving Errors and Conflicts on the Mobile Server 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 6.11.4.3 "Viewing Transactions in the Error Queue" in the *Oracle Database Lite Administration and Deployment Guide* for directions.

2.11.2 Viewing Client-Side Synchronization Conflicts from Automatic Synchronization

For automatic synchronization publication items only, any synchronization conflict error information that occurs on the client is stored in the `CONF$`*`<snapshot>`* table in the same client database as the snapshot.

Note: The rules for the client conflict resolution is the same as what was set for the server. These tables show the errors that occured on the client during automatic synchronization.

This table has the same columns and the snapshot, plus some additional columns. Some of the additional columns in this table are reserved, but the columns that you can use to determine what course to take in resolving the conflict are as follows:

- `MSG$PRIO`—Lists the priority of the data where priority 0 is the highest.

  Oracle Database Lite adds the `MSG$PRIO` column to all automatic synchronization snapshots to designate if this data has a higher priority for synchronization if conditions are right. If users need to indicate that a particular record is high priority, they can set the column value to (0). Then Sync Agent automatically schedules a high priority synchronization for the transaction that contains this record.
- `MSG$TYPE`—`I` for insert, `U` for update, or `D` for delete DML.

- `MSG$CONFTYPE`—`S` for server-side DML or `C` for client-side DML.

2.11.3 Customizing Synchronization Conflict Resolution Outcomes

You can customize synchronization conflict resolution by doing the following:

1. Configure the winning rule to Client Wins.
2. Perform only ONE of the following:
   - Create and attach one or more triggers on the back-end Oracle database base tables to execute before the `INSERT`, `UPDATE`, or `DELETE` DML statements. The triggers should be created to evaluate the data and handle the conflict. Triggers are created to compare old and new row values and resolve client changes as defined by you. See the *Oracle Database* documentation for full details on how to create and attach triggers.
   - Create a custom DML procedure. See Section 2.4.1.13, "Callback Customization for DML Operations" for an example of how to create a custom DML procedure.

     You can use the `generateMobileDMLProcedure` to generate the procedure specification for a given publication item. This specification can be used as a starting point in creating your own custom DML handling logic in a PL/SQL procedure. You use the `addMobileDMLProcedure` API to attach the PL/SQL procedure to the publication item. See the *Oracle Database Lite API Specification* for more information.

2.12 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 2.12.1, "getDownloadInfo Method"
- Section 2.12.2, "DownloadInfo Class Access Methods"
- Section 2.12.3, "PublicationSize Class"

2.12.1 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.

2.12.2 DownloadInfo Class Access Methods

The access methods provided by the `DownloadInfo` class are listed in Table 2–23:

Table 2–23 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.

2.12.3 PublicationSize Class

The access methods provided by the `PublicationSize` class are listed inTable 2–24:

Table 2–24 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();
      }
   }
}
```

2.13 Customizing Synchronization With Your Own Queues

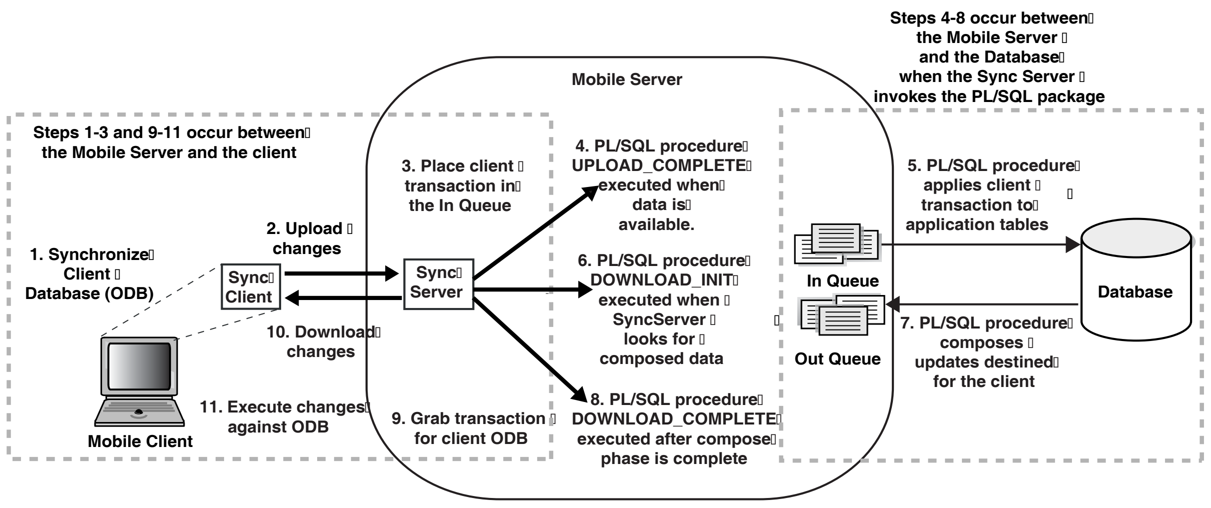
Application developers can manage the synchronization process programmatically by using queue-based publication items. By default on the server-side, the MGP manages both the In Queues and the Out Queues by gathering all updates posted to the In Queue, applying these updates to the relevant tables, and 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 2.1, "How Oracle Database Lite Synchronizes".

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 2–4 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 2–4 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>*.

The following sections describe the methods for customizing the server-side apply/compose phases-++:

- Section 2.13.1, "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 2.13.2, "Creating Data Collection Queues for Uploading Client Collected Data"—You use the data collection queue for uploading data from the client. The queues are optimized for when a client collects data to upload to the server and never receives data from the server.
- Section 2.13.3, "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.

2.13.1 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 2.13.3, "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.

Note: Normally, you define the package on the Main database where the Mobile repository is located. However, if you are using a remote database for your application data, then the package must be defined on the remote database. For more information on remote databases, see Section 3.2, "Register a Remote Oracle Database for Application Data".

2.13.1.1 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 2–25:

Table 2–25 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 2–26:

Table 2–26 Out Queue Interface Creation Parameters

Parameter	Description
`CLID$$CS`	A unique string identifying the client.
`TRANID$$`	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 2–27:

Table 2–27 Queue Control Table Parameters

Parameter	Description
CLID$$CS	A unique string identifying the client.
TRANID$$	A unique number identifying the transaction.
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 2–28:

Table 2–28 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.

2.13.1.2 Queue-Based PL/SQL Callouts

The PL/SQL package for the queue-based publication callouts is in a package where the UPLOAD_COMPLETE, DOWNLOAD_INIT, DOWNLOAD_COMPLETE, and POPULATE_Q_REC_COUNT 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
     );

PROCEDURE POPULATE_Q_REC_COUNT(
     CLIENTID     IN     VARCHAR2
     );

END CONS_QPKG;
/
```

2.13.1.2.1 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;
```

2.13.1.2.2 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.

2.13.1.3 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 6.4, "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.

2.13.1.4 Register the PL/SQL Package Outside the Repository

Once you finish developing the PL/SQL package, register the package in the `MOBILEADMIN` schema with 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.

2.13.2 Creating Data Collection Queues for Uploading Client Collected Data

If you have an application that collects data on a client, such as taking inventory or the amount collected on 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 2.13.1, "Customizing Apply/Compose Phase of Synchronization with a Queue-Based Publication Item" for more information.

Data collection queues can be used for the following two types of data collection:

- New records that are inserted on the client.
- Existing records that are downloaded to the client in order that the user can modify and upload these records.

An example of the second type is a supply counting application. If you want to count the number of items in stock, then you could design the application table with the columns: Item and Count. Initially, populate the Item column and synchronize the data to the device, as follows:

Table 2–29 Stock Inventory Table

Item	**Count**
Apples	-
Pears	-
Oranges	-

The user on the client updates each item with the inventory amount, as follows:

Table 2–30 Stock Inventory Table

Item	**Count**
Apples	2
Pears	3
Oranges	1

The Data Collection Queue is lightweight and simple to create. Data collection queues are the same as regular queues with the exception that they provide automatic apply of the data uploaded by the client. However, 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 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>`.

Just like for regular queues, users can create their own Out Queue logic. By default, the Out Queue created is an empty view with the name of (`CTM$`*`<publication_item>`*). An empty view is a view that selects zero records. Therefore, by default, data collection queues do not pick up any data from the server.

You can modify how the data collection queue behaves when you create it using the `ConsolidatorManager.createDataCollectionQueue` method. The following parameters effect the behavior of your data collection queue:

- Specify an Out Queue—Out Queue creation is affected by the `isOutView` boolean input parameter. If `isOutView` is TRUE, then creates the Out Queue as an empty view; if FALSE, then creates the Out Queue as a table.
- Automatic Removal of Data on the Client—Users can customize the default behavior of data purging on the client by setting the `purgeClientAfterSync` parameter to either true or false.

- If TRUE, then the client uploads its data changes and removes the records from the client database. At this point, the table on the client is empty. If the Out Queue on the server is empty, the client will no longer have any records. If the Out Queue is not empty, the client downloads these records and the table on the client contains only these records.
- If FALSE, then the client records remain on the device after synchronization unless the server explicitly sends the `DELETE` command, in the same manner as a normal publication item.

2.13.2.1 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.
   - `purgeClientAfterSync`—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.

2.13.3 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 2–31 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.

2.14 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.

2.15 Troubleshooting Synchronization Errors

The following section can assist you in troubleshooting any synchronization errors:

- Section 2.15.1, "Foreign Key Constraints in Updatable Publication Items"

2.15.1 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 2.15.1.1, "Foreign Key Constraint Violation Example"
- Section 2.15.1.2, "Avoiding Constraint Violations with Table Weights"
- Section 2.15.1.3, "Avoiding Constraint Violations with BeforeApply and After Apply"

2.15.1.1 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 2.15.1.2, "Avoiding Constraint Violations with Table Weights" for more information.

2.15.1.2 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 2.15.1.1, "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 2.4.1.7.2, "Using Table Weight".

2.15.1.3 Avoiding Constraint Violations with BeforeApply and After Apply

You can use a PL/SQL procedure to 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 2.10.3.2, "Defer Constraint Checking Until After All Transactions Are Applied" for more information.

3 APIs for Client and Database Administration

The Mobile client database contains a subset of data stored in the Oracle database., which are synchronized with the Oracle database.

The following sections describe how you can programmatically configure the features for the back-end Oracle database.

3.1 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.

3.2 Register a Remote Oracle Database for Application Data

By default, the Mobile repository metadata and the application schemas are present in the same database. However, it is possible to place the application schemas in a database other than the MAIN database where the Mobile repository exists. This can be an advantage from a performance or administrative viewpoint.

Thus, you can spread your application data across multiple databases.

Note: We refer to the database where the application schema resides as remote because it is separate from the MAIN database that contains the Mobile repository. It does not mean that the database is geographically remote. It can be local or remote. For performance reasons, the Mobile Server must have connectivity to all databases involved in the synchronization—MAIN and remote.

This section describes how to register a remote Oracle database containing application schemas, using the `ConsolidatorManager` APIs. However, it is recommended that you use the Oracle Database Lite GUI tools for this task unless you have a specific need to use the API. For concepts and description of how to perform this with the Oracle Database Lite GUI tools, see Section 6.6, "Register or Deregister an Oracle Database for Application Data" in the *Oracle Database Lite Administration and Deployment Guide*.

To use an Oracle database other than the Oracle database used for the Mobile repository, perform the following:

1. Use the `apprepwizard` script to setup a remote application repository. See Section 3.2.1, "Set up a Remote Application Repository With the APPREPWIZARD Script" for details.
2. Register the Oracle database as described in Section 3.2.2, "Register or Deregister a Remote Oracle Database for Application Data".
3. When creating the publication and publication items, specify the name of the registered Oracle database that contains the application schemas. All data for a single application—that is, all publication items for the publication—must be contained in the same Oracle database.

3.2.1 Set up a Remote Application Repository With the APPREPWIZARD Script

Use the `apprepwizard` script to setup a remote application repository. This script creates and initializes an administrator schema with the same name as the adminstrator schema in the Main database. For example, if the administrator schema name in the Main database is `mobileadmin`, then the `apprepwizard` script will create a `mobileadmin` schema on the remote database.

The `apprepwizard` script is located in the *ORACLE_HOME*`/Mobile/Server/admin`. The usage of this script is as follows:

```
apprepwizard.bat <MAIN_Repository_Schema_Name> <MAIN_Repository_Schema_Password>
 <Application_Database_Administrator_User_Name>
 <Application_Database_Administrator_Password>
 <Application_Database_JDBC_URL> <Application_Database_Schema_Password>
 [<DB_name>]
```

Where each parameter is as follows:

- `MAIN_Repository_Schema_Name`: Provide the Mobile repository schema name, which exists on the Main database. The default is `MOBILEADMIN`.
- `MAIN_Repository_Schema_Password`: Provide the password for the Mobile repository administrator schema.
- `Application_Database_Administrator_User_Name`: Any user with administrator privileges at the application database. such as `SYSTEM`.
- `Application_Database_Administrator_Password`: Password of the administrator user for the application database.
- `Application_Database_JDBC_URL`: JDBC URL of the application database.
- `Application_Database_Schema_Password`: Password of the schema, which will be created at the application database. The username is the same as the Mobile repository schema name.
- `DB_Name`: Optionally, the user can provide a name to identify this database. This name is used in logging. By default, the log is sent to the console. If this name is provided as the last parameter, then the log is generated in the By default, the log is sent to the console. If the database name is provided as the last parameter, then the log is generated in the *ORACLE_HOME*`/Mobile/Server/`*<DB_NAME>*`/apprepository.log` file.

This script installs silently. Thus, If you execute this script without any arguments, nothing is performed.

3.2.2 Register or Deregister a Remote Oracle Database for Application Data

Use the following `ConsolidatorManager` APIs to register, deregister, or alter the properties of the remote Oracle database:

```
void registerDatabase(String name, Consolidator.DBProps props)
void deRegisterDatabase(String name)
void alterDatabase(String name, Consolidator.DBProps props)
```

Where:

- **Name**—An identifying name for the database where the application schema resides. Once defined, this name cannot be modified. This name must be unique across all registered database names.
- **DBProps**—A class that contains the JDBC URL, password and description, as follows:

  ```
  public static class DBProps {
   public String jdbcUrl;
   public String adminPassword;
   public String description;
  }
  ```

  - **JDBC URL**—The JDBC URL can be one of the following formats:

    * The URL for a single Oracle database has the following structure: `jdbc:oracle:thin:@<host>:<port>:<SID>`

    * The JDBC URL for an Oracle RAC database can have more than one address in it for multiple Oracle databases in the cluster and follows this URL structure:

    ```
    jdbc:oracle:thin:@(DESCRIPTION=
     (ADDRESS_LIST=
    ```

```
      (ADDRESS=(PROTOCOL=TCP)(HOST=PRIMARY_NODE_HOSTNAME)(PORT=1521))
      (ADDRESS=(PROTOCOL=TCP)(HOST=SECONDARY_NODE_HOSTNAME)(PORT=1521))
    )
    (CONNECT_DATA=(SERVICE_NAME=DATABASE_SERVICENAME)))
```

- **Password**—The administrator password is used to logon to the database. The administrator name is the same as what was defined for the main database.

  When defining, the password must conform to the following restrictions:

  – not case sensitive

  – cannot contain white space characters

  – maximum length of 28 characters

  – must begin with an alphabet

  – can contain only alphanumeric characters

  – cannot be an Oracle database reserved word

- **Description—**A user-defined description to help identify this database.

Refer to the `ConsolidatorManager` Javadoc in the *Oracle Database Lite API Specification* for more details.

The following code example registers a database as `APP1`. The `registerDatabase` API stores access information for the application repository and provides a name so that publications, publication items, and MGP Jobs can be created against this repository. It does not define the administrator schema.

```
Consolidator.DBProps props = new Consolidator.DBProps();
props.jdbcUrl = "jdbc:oracle:thin:@apphost:1521:app1";
props.description="App database 1"
props.adminPassword = "secret";
consMgr.registerDatabase("APP1", props);
```

The following code example deregisters the `APP1` database.

```
consMgr.deRegisterDatabase("APP1");
```

You can retrieve the names of all of the registered databases with the `getDatabaseInstances` method, which is as follows:

```
Map getDatabaseInstances()
```

The `Map` returned by `getDatabaseInstances` method contains a keyset of the application database names and the entry for each key is a `Consolidator.DBProps` class where the `adminPassword` is always null for security purposes.

3.2.3 Create Publication, Publication Item, Hints and Virtual Primary Keys on a Remote Database

You must have already registered the remote database before defining publications, publication items, hints, and virtual primary keys that use the application data schemas and tables on the remote database. In the `ConsolidatorManager` API calls, the registered name of the remote database is required.

Note: The publication and publication item names are unique irrespective of where the data resides.

All publication items within a publication must be defined on tables within the same database.

The following example illustrates the creation of a publication and a publication item against a remote database registered as `APP1`. Refer to the `ConsolidatorManager` Javadoc in the *Oracle Database Lite API Specification* for more details.

```
ConsolidatorManager consMgr = new ConsolidatorManager();
consMgr.openConnection("mobileadmin", "mobileadmin",
 "oracle:jdbc:thin:@host1:1521:master");
consMgr.createPublication( "PUB1","APP1",Consolidator.DFLT_CREATOR_ID,
 "ddb.%s", null);

Consolidator.PubItemProps taskPIProps = new Consolidator.PubItemProps();
taskPIProps.db_inst = "APP1"; // Remote App database name as registered
taskPIProps.owner = "APPUSER1";
taskPIProps.store = "TASKS";
taskPIProps.refresh_mode = "F";
taskPIProps.select_stmt = "select id, emp_id, cust_id, stat_id, notes
 from APPUSER1.TASKS";
taskPIProps.cbk_owner = "MOBILEADMIN";
taskPIProps.cbk_name = "TASKSPI_PKG";
consMgr.createPublicationItem( "PI_1_TASKS", taskPIPProps);

consMgr.addPublicationItem("PUB1", "PI_1_TASKS", null, null, "S", null, null);
consMgr.createSubscription( "PUB1", "USER1");
consMgr.instantiateSubscription("PUB1", "USER1");
consMgr.closeConnection();
```

Other API calls for managing data collection queues, hints, and virtual primary keys that require the remote database name are shown below. Refer to the `ConsolidatorManager` Javadoc in the *Oracle Database Lite API Specification* for more details.

- **Data Collection Queue**

  ```
  void createDataCollectionQueue(String name, String db_inst,
     String owner, String store, String inq_cols, String pk_columns,
     boolean purgeClientAfterSync, boolean isOutView)
  ```

- **Hint**

  ```
  void parentHint(String db_inst, String owner, String store, String owner_d,
      String store_d)
  void dependencyHint(String db_inst, String owner, String store,
      String owner_d, String store_d)
  void removeDependencyHint(String db_inst, String owner, String store,
      String owner_d, String store_d)
  ```

- **Virtual Primary Key**

  ```
  public void createVirtualPKColumn(String db_inst, String owner,
      String store, String column)
  public void dropVirtualPKColumns(String db_inst, String owner,
      String store)
  ```

The APIs used for creating a publication and publication item is the same except for the addition of the remote database name. Following is an example that provides the remote database name, APP1, in bold for creating a publication and publication item:

```
ConsolidatorManager consMgr = new ConsolidatorManager();
consMgr.openConnection("mobileadmin", "mobileadmin",
```

```
"oracle:jdbc:thin:@host1:1521:master");
consMgr.createPublication( "PUB1","APP1",Consolidator.DFLT_CREATOR_ID,
       "ddb.%s", null);
Consolidator.PubItemProps taskPIProps = new Consolidator.PubItemProps();
taskPIProps.db_inst = "APP1"; // Remote APP instance name as registered
taskPIProps.owner = "APPUSER1";
taskPIProps.store = "TASKS";
taskPIProps.refresh_mode = "F";
taskPIProps.select_stmt = "select id, emp_id, cust_id, stat_id, notes from
APPUSER1.TASKSî;
taskPIProps.cbk_owner = "MOBILEADMIN";
taskPIProps.cbk_name = "TASKSPI_PKG";
consMgr.createPublicationItem( "PI_1_TASKS", taskPIPProps);
consMgr.addPublicationItem("PUB1", "PI_1_TASKS", null, null, "S", null, null);
consMgr.createSubscription( "PUB1", "USER1");
consMgr.instantiateSubscription("PUB1", "USER1");
consMgr.closeConnection();
```

3.2.4 Using Callbacks on Remote Databases

The following sections describe how the synchronization callbacks, described in Section 2.7, "Customize What Occurs Before and After Synchronization Phases", must be handled for the remote database:

- Section 3.2.4.1, "Customize Callbacks on the Remote Database"
- Section 3.2.4.2, "Publication Item Level Callbacks for the MGP Apply/Compose Phases"

3.2.4.1 Customize Callbacks on the Remote Database

The Customize callbacks, as described in Section 2.7.1, "Customize What Occurs Before and After Every Phase of Each Synchronization", are created to perform defined tasks before or after any phase of synchronization.

Most of the callbacks pertain to MGP processing. Since an MGP Job executes against a database, these callbacks are invoked separately by each job against the corresponding database. Callbacks that are not related to the MGP are invoked against the MAIN database. Thus, the callback PL/SQL package must be created on the MAIN database as well as on the appropriate remote databases.

3.2.4.2 Publication Item Level Callbacks for the MGP Apply/Compose Phases

Define the MGP publication item level callbacks on the database against which the publication item is defined. Then, these can access the base tables on that database.

For full details on the MGP publication item level callbacks, see Section 2.7.2, "Customize What Occurs Before and After Compose/Apply Phases for a Single Publication Item".

3.2.4.3 Customizing the Apply/Compose Phase for a Queue-Based Publication Item on a Remote Database

When you customize the apply/compose phase for a queue-based publication item, as described in Section 2.13.1, "Customizing Apply/Compose Phase of Synchronization with a Queue-Based Publication Item", then these packages must be defined on the database where the queue-based publication item base tables exist. Thus, if the base tables exist on a remote database, then the packages must be defined on the remote database.

3.3 Create a Synonym for Remote Database Link Support For a Publication Item

Publication items can be defined for database objects existing on remote databases outside of the Mobile Server repository. Local private synonyms of the remote objects can be created in the Oracle database. However, we recommend that you use the remote database functionality as described in Section 3.2, "Register a Remote Oracle Database for Application Data".

If you still decide to use database links for defining publication items on remote databases, then you can 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 3.3.1, "Publishing Synonyms for the Remote Object Using CreatePublicationItem"
- Section 3.3.2, "Creating or Removing a Dependency Hint"

3.3.1 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`.

3.3.2 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.

3.4 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

3.4.1 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.

3.4.2 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.

3.5 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 3.5.1, "Creating Secondary Indexes on Client Device"
- Section 3.5.2, "Virtual Primary Key"

3.5.1 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 2.4.1.6, "Create Publication Item Indexes" for an example.

3.5.2 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. A virtual primary key can consist of a single or multiple columns, where each column included in the virtual primary key must not null. If a null value is entered into any column of a virtual primary key, this results in an error. If the virtual primary key is on a single column, it must be unique; if the virtual primary key consists of a composite of multiple columns, then the composite must be unique.

If you want to create a virtual primary key for more than one column, then the API must be called separately for each column that you wish to assign to that 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.

3.6 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 are 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 using this method. Instead, use the directions provided in** Section 3.6.1, "Schema Evolution Involving a 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 synchronization 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.

3.6.1 Schema Evolution Involving a Primary Key

What if you want to perform a schema evolution that does include a modification to the primary key. Normally, you would drop the entire publication and recreate it. However, there is a way that you can modify the primary key constraint and recreate the publication item without dropping the entire publication.

The following steps describe how to remove the primary key constraint, add a new column and identify it as the primary or virtual primary key and then recreate the publication item. The steps below must be followed in the order listed:

1. Using MDW, remove the publication item from the publication and drop the publication item from the repository.
2. Modify the table in the back-end Oracle database, as described in the following steps:
   a. Drop the Primary Key constraint. For example, if `table1` has primary key constraint of `pk_constraint`, then drop this constraint, as follows:

      ```
      alter table table1 drop constraint pk_constraint;
      ```

   b. Add a new column to perform as the new primary key or virtual primary key, as follows:

      ```
      alter table table1 add my_new_col number(5,0) not null;
      ```

   c. Populate the new column with values that can be used (solely or as part of) the new the primary key or virtual primary key.
   d. Alter the table to create a primary key or virtual primary key constraint on the new column. If you want to create the primary key constraint on the new column `my_new_col` for `table1`, use the `ALTER TABLE` SQL command. If you want to define a virtual primary key on `my_new_col` for `table1`, use MDW.
3. In MDW, create a new publication item for `table1`. This should be a duplicate of the previously dropped publication item, but with teh new column included. When creating the publication item, verify the my_new_col appears as teh primary key.
4. Add the publication item to the publication.
5. Reset the Metadata Cache using the Mobile Manager by selecting Data Synchronization -> Administration -> Reset Metadata Cache.
6. Verify in the Parent Table Primary Key and Base table Primary Key fields in the Publication Item detail screen in the Mobile Manager that the new primary key is in effect.
7. Synchronize on the existing client device to bring down the new publication.
8. After the synchronization is complete, then verify that the new column is present and that it is functioning as the primary key on the client device.

3.7 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.

3.8 Datatype Conversion Between the Oracle Server and the Oracle Lite Database

Before you publish your application, create the tables for your applications in the Oracle database. Thus, when the first synchronization occurs for an Oracle Lite Mobile client, Oracle Database Lite takes the Oracle database datatypes and converts them to corresponding allowed datatypes in the Oracle Lite database on the client. Table 3–1 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 E, "Oracle Database Lite Datatypes" in the *Oracle Database Lite SQL Reference*.

Table 3–1 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										

Table 3–1 (Cont.) 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
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 in Applications With the Mobile Sync APIs

The following sections describe the Mobile Sync APIs available to start synchronization programmatically within your application on the Mobile client, whether the application is C, C++, C#, or Java:

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:

4.1.1 Overview of C/C++ Synchronization API

For starting synchronization, the application should perform the following:

1. Create, memset, and initialize the `ocEnv` structure.
2. Invoke the `ocSessionInit()` method.
3. Set any optional fields in the `ocEnv` structure, such as username and password. If you want to preserve all optional fields set in the `ocEnv` structure for future synchronization sessions, then execute the `ocSaveUserInfo` method.
4. Optionally, you can set proxy information with the `ocSetSyncOption` method or specify the synchronization type for each table with the `ocSetTableSyncFlag` function.
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 `SimpleSync.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 you want to save the user preferences for future sessions, then invoke the `ocSaveUserInfo` method.

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 `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. If set to `OC_FILE_TRANSPORT`, then the synchronization uses file-based sync. See Section 4.1.11, "Enable File-Based Synchronization through C or C++ APIs" for more information.

Table 4–2 (Cont.) ocEnv Structure Field Parameters

Field	Type	Usage	Description
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.
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. You can only use fast refresh with a high priority restricting predicate. If you use any other type of refresh, the high priority restricting predicate is ignored. See Section 1.2.10, "Priority-Based Replication" in the *Oracle Database Lite Troubleshooting and Tuning Guide* for more information.
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.

Table 4–2 (Cont.) ocEnv Structure Field Parameters

Field	Type	Usage	Description
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.
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. `ocDoSynchronize` 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 2.4, "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 2.4, "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.

Table 4–7 (Cont.) Description of the Stage Values

Stage Value	Description
`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.
`OC_RETRY_CALL`	Resume synchronization is restarted.
`OC_SYNC_FINISH`	Last callback after the synchronization.

- `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.

4.1.11 Enable File-Based Synchronization through C or C++ APIs

When you want to use file-based synchronization, you must enable file-based synchronization. Once enabled, then when you initiate manual synchronization, then the synchronization file is created. See Section 6.8, "Synchronizing to a File Using File-Base Sync" in the *Oracle Database Lite Administration and Deployment Guide* for more details on file-based synchronization.

To enable file-based synchronization programmatically with the `ocEnv` structure, perform the following:

1. Ensure that any previous settings of the File-Based Sync properties are set to `NULL`.
2. Initialize the environment with the `ocSessionInit` method.
3. Set the username and password for the user that is initializing the synchronization.
4. Specify the synchronization direction and directory and filename for the synchronization file. The synchronization direction is either send, which creates the synchronization file, or receive, which takes in a file from the Mobile Server. These are configured in the `SEND_FILE_PROP` and `RECEIVE_FILE_PROP` properties with the `ocSetSyncProperty` method.
   - When you set the `SEND_FILE_PROP` property, specify the filename—including the relative or full path—where you want the Mobile client to save the upload data for the Mobile Server. This file is created with the Mobile client transactions destined for the Mobile Server.
   - When you set the `RECEIVE_FILE_PROP` property, specify the filename—including the relative or full path—where the data file that was received from the Mobile Server. This file is loaded and processed within the Mobile client.

The following code example sets the direction, filename, username and password. Notice that the `ocEnv` structure is memset to zero to ensure that if a previous direction and filename were specified, then these are invalidated for the next file-based synchronization. The `SEND_FILE_PROP` property is set with the filename and direction, which tells the Sync Client to marshall the Mobile client transactions that are to be uploaded to the Mobile Server into this file. If you were receiving a synchronization file from the Mobile Server, you would have set the `RECEIVE_FILE_PROP` property with the location and name of this file.

Finally, the `ocEnv` structure is provided to the `ocDoSynchronize` method, which performs the file-based synchronization.

```
ocEnv env;
memset(&env, 0, sizeof(ocEnv));
ocSessionInit(&env);
strcpy(env.username, "S11U1");
strcpy(env.password, "manager");
```

```
ocSetSyncProperty(&env, SEND_FILE_PROP, "C:\\temp\\send1.bin");
ocDoSynchronize(&env);
ocSessionTerm(&env);
```

4.2 Synchronization API for Java Applications

The following sections describe how you can use Java API 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, "SyncProgressListener Service"
- Section 4.2.8, "Manage What Tables Are Synchronized With Selective Sync"
- Section 4.2.9, "Enable File-Based Synchronization through Java APIs"

4.2.1 Overview

The Java interface for Mobile Sync client-side synchronization 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`. These files are located in the `<`*`ORACLE_HOME`*`>\Mobile\Sdk\bin` directory.

The following are the classes and interface for the Java API:

- `Sync` Class
- `SyncException` Class
- `SyncOption` Class
- `SyncProgressListener` 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 transportType,
     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.
`transportType`	A string containing the name of the transport driver. Currently, only `HTTP` or `FILE` are 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.

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)`	This is used to set and get the current user.
`String getuser()`	

Table 4–15 (Cont.) Sync Option Public Method Parameters

Parameter	Description
`void setPassword(String password)` `String getPassword()`	This is used to set and get the password.
`void setSyncParam(String syncParam)` `string getSyncParam()`	This is used to set and get the synchronization parameters.
`void setTransportType(String driverName)` `String getTransportType()`	This is used to set and get the driver name, which defaults to `HTTP`. Set to `FILE` if using file-based synchronization.
`void setTransportParam(String transportParam)` `String getTransportParam()`	Set and get the transport parameters.

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;proxyPort=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 transportType = "HTTP";
    String trasportParam = "server=localhost";

    /**
     * Constructor
     */
    public JavaSyncClient() throws Exception{
        //Create the SyncOption class
        SyncOption syncOpt = new SyncOption(user, password,
            syncParam, transportType, 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.
`"highPriority"`	String	A string parameter that forces the server to append a restricting predicate to the publication item querys where restricting predicate exists. This limits the number of records client downloads and should be used in combination with selective sync which selects only high priority snapshots. You can only use fast refresh with a high priority restricting predicate. If you use any other type of refresh, the high priority restricting predicate is ignored. See Section 1.2.10, "Priority-Based Replication" in the *Oracle Database Lite Troubleshooting and Tuning Guide* for more information.
`"pushOnly"`	A boolean value which makes synchronization push only.	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.

Table 4–16 (Cont.) Java Interface SyncParamSettings

Name	Value/Options	Description
`[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 Agent 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 SyncProgressListener 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 4–18:

Method

```
void progress

   (int progressType,
    int completed);
```

Table 4–18 SyncProgressListener Abstract Method

Parameter	Description
`progressType`	This is set to one of the constants listed in Table 4–19.
`completed`	This is the percentage of completion for specific `progressType`.

The names of the constants which report the synchronization progress are listed in Table 4–19.

Table 4–19 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.

Note: Some codes are returned from the OCI layer. If you receive a status code not listed here, see Table 4–7.

Example

This simple class implements the `SyncProgressListener`.

```
class myProgressTracker implements SyncProgressListener;

{
  public void progress
     (int progressType,
     int completed)
    {
      System.out.println( "Status: "+progressType+"="+ completed+"%" );
     } //progress
 }
```

4.2.8 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 4–20 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 2.4, "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 2.4, "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
```

4.2.9 Enable File-Based Synchronization through Java APIs

When you want to use file-based synchronization, you must enable file-based synchronization. Once enabled, then when synchronization occurs—either through automatic or manual synchronization—then the synchronization file is created. See Section 6.8, "Synchronizing to a File Using File-Base Sync" in the *Oracle Database Lite Administration and Deployment Guide* for more details on file-based synchronization.

To enable file-based synchronization programmatically with the `SyncOption` class, specify the following:

1. Create the `SyncOption` class with the username and password.
2. Specify the synchronization direction and directory and filename for the synchronization file with the `setSyncProperty` method of the `SyncOption` class. The synchronization direction is either send, which creates the synchronization file, or receive, which takes in a file from the Mobile Server. These are configured in the `SEND_FILE_PROP` and `RECEIVE_FILE_PROP` properties with the `setSyncProperty` method.
   - When you set the `RECEIVE_FILE_PROP` property with the filename and directory, the intended file is uploaded and processed within the Mobile client.
   - When you set the `SEND_FILE_PROP` property with the filename and directory, the intended file is created with the Mobile client transactions destined for the Mobile Server.

The following code example sets the direction, filename, username and password. In this example, the `SEND_FILE_PROP` property is set with the filename and direction, which tells the Sync Client to marshall the Mobile client transactions that are to be uploaded to the Mobile Server into this file. If you were receiving a synchronization file from the Mobile Server, you would have set the `RECEIVE_FILE_PROP` property with the location and name of this file.

Finally, the `Sync` class is instantiated with the `SyncOption` settings and a synchronization is performed.

```
SyncOption sync_op = new SyncOption("S11U1", "manager", "", "", "");

op.setSyncProperty(SyncOption.SEND_FILE_PROP,"C:\\temp\\send1.bin");
Sync mSync = new Sync(op);
mSync.doSync();
```

4.3 Synchronization API for Java Applications on SQLite Mobile Clients

The following sections describe how you can use Java APIs to build your own client synchronization initiation on SQLite Mobile clients:

- Section 4.3.1, "Overview"
- Section 4.3.2, "OSESession Class"
- Section 4.3.3, "OSEProgressListener Interface"
- Section 4.3.4, "Enable Selective Synchronization"
- Section 4.3.5, "OSEException Class"

4.3.1 Overview

The Java interface for SQLite Mobile client synchronization resides in the `oracle.opensync.ose` 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

For Win32, Windows Mobile, and Linux clients, the Java interface is implemented using JNI and consists of two files: `jsync.jar` and `msync_java.dll`. The pure Java interface is implemented in the `msync.jar` file. To use the JNI synchronization interface, include the `jsync.jar` file in the `CLASSPATH` and `msync_java.dll` file in the `PATH`. These files are located in the `<`*`ORACLE_HOME`*`>\Mobile\Sdk\sqlite` directory.

The pure Java library for the Blackberry RIM platform is located in the `<`*`ORACLE_HOME`*`>\Mobile\Sdk\sqlite\rim\lib` directory; the pure Java library for the Android platform is located in the `<`*`ORACLE_HOME`*`>\Mobile\Sdk\sqlite\android\lib` directory.

The following are the classes and interface for the Java API for SQLite Mobile clients:

- `OSESession` Class
- `OSEProgressListener` Interface

4.3.2 OSESession Class

`OSESession` enables setting synchronization parameters and options. This class exposes APIs to invoke and control synchronization by using the provided synchronization options.

Synchronization progress is reported through the `OSEProgressListener` interface, which is set by the `OSESession setProgress(OSEProgressListener)` method.

The parameters for the constructor are listed in Table 4–10.

Constructors

```
OSESession( )

OSESession( String user )

OSESession( String user, char[] pwd)
```

Table 4–21 OSESession Class Constructor

Parameter	Description
`user`	A string containing the name used for authentication by the Mobile Server.
`password`	A character array containing the user password.

Public Methods

The public methods and their parameters for the `OSESession` class are listed in Table 4–22:

Table 4–22 OSESession Class Public Method Parameters

Parameter	Description
`void cancelSync( )`	Attempts to cancel the sync process with a non-blocking call. If successful, throws `OSEException` with error code `OSEExceptionConstants.SYNC_CANCELED`.
`void close()`	Closes any active database connections that the session maintains. This method is called before application exits.

`void setAppRoot(String appRoot)`
`String getAppRoot( )`
	Sets or retrieves the current root directory, as set in the `DATA_DIRECTORY` parameter, for internal synchronization and database files for the application.

`void setEncryptionType(int type)`
`int getEncryptionType( )`
	Sets or retrieves the current encryption type. Possible types can are as follows:
	■ `ENC_AES` - AES encryption, which is the default.
	■ `ENC_SSL` - SSL over HTTP.
	■ `ENC_NONE` - No encryption.

`void setForceRefresh(boolean on)`	Set to wipe out all of the client data and replace it with server data, if true.
`boolean getForceRefresh( )`	Retrieves value of force refresh.

`void setSavePassword(boolean on)`
`boolean getSavePassword()`
	This is used to set and get the flag for persistently saving the user password. If true, the password will be saved.

`void setNewPassword(char[] pwd)`	Allows clients to modify their password on the server. After a successful synchronization, the client's password on the server will be changed to the new password.
`void setPassword(char[] pwd)`	Provide or modify the SQLite Mobile client password.

`void setSyncNewPub( )`
`boolean getSyncNewPub( )`
	Sets flag for enabling synchronization of new publications. By default, this is set to true and all publications are synchronized. However, if you set this to false, any new subscribed publications on the server are not downloaded to the client.

`void setURL(java.lang.String url)`
`java.lang.String getURL( )`
	Sets or retrieves the HTTP URL of the Mobile Server.

Table 4–22 (Cont.) OSESession Class Public Method Parameters

Parameter	Description

`void setUseFiles(boolean on)`
`boolean getUseFiles()`
	Set flag to switch between using streaming or files to transport synchronization data. If set to true, synchronization stores uploaded and downloaded data in a file; otherwise, data will be streamed.
	When using files, the `ose$in.bin` file contains the data received from the server. The `ose$out.bin` file contains the data sent to the server. These files are located in the `<`*`mobileclient_root`*`>\bin` directory on Win32, WinCE, Windows Mobile and Linux platfoms or in the directory specified by the `SQLITE.DATA_DIRECTORY` on the Android or Blackberry platforms.
	Note: streaming requires that the underlying client transport stack implements HTTP 1.1. Thus, if a platform does not support streaming, `setUseFiles` must be congifigured as TRUE.

`void saveUser()`
`String getUser()`
	The `saveUser` method saves user information, such as users specific information, and the last sync user id.
	The `getUser` method retrieves current synchronization client name.

`void selectPub(String name)`	Provided the publication name, adds the publication to the list of publications to be synchronized selectively. See Section 4.3.4, "Enable Selective Synchronization" for more information.
`void setProgress(OSEProgressListener p)`	Set synchronization progress listener.
`void sync()`	Initiates a manual synchronization from within the application.

Example

The following example sets the username and password to `JOHN/john`. The Mobile Server URL is identified as `localhost:88`. And a synchronization is initiated with the `sync` method.

```
/* set up username and password */
String user = "JOHN";
String pwd  = "john";

/* create OSESession with user John */
OSESession sess = new OSESession(user, pwd.toCharArray());

/* Identify Mobile Server URL */
sess.setURL("localhost:88");

/* Identify the progress listener, myProgressTracker */
sess.setProgress(myProgressTracker);

/* Initiate Sync */
sess.sync();
```

4.3.3 OSEProgressListener Interface

The `OSEProgressListener` interface enables progress updates to be trapped during synchronization.

Sync calls the progress function to report the current stage and the percent of completion of that stage. The parameters for the progress method are listed in Table 4–23:

Method

```
void progress (int stage, int val);
```

Table 4–23 OSEProgress Method Parameters

Parameter	Description
`stage`	This is set to one of the constants listed in Table 4–24.
`val`	This is the percentage of completion for specific `stage`.

The names of the constants which report the synchronization progress are listed in Table 4–24.

Table 4–24 OSEProgressListener Interface Constants

Constant Name	Progress Type
`PREPARE`	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.
`SEND`	States that the synchronization engine is sending data to the network.
`RECEIVE`	States that the synchronization engine is receiving data from the server.
`PROCESS`	States that the synchronization engine is applying the newly received data from the server to the local data stores.
`IDLE`	States that the synchronization engine has completed the synchronization process.

Example

This simple class implements the `OSEProgressListener`.

```
class myProgressTracker implements OSEProgressListener;

{
  public void progress
      (int state,
      int val)
     {
      System.out.println( "Status: "+state+"="+ val+"%" );
      } //progress
 }
```

4.3.4 Enable Selective Synchronization

Selective sync specifies whether a publication should be synchronized or not for the next session. Set the flag with the `selectPub` method to indicate whether the publication is to be synchronized on the next execution of the `sync` method. The default setting is NULL for all publications.

> **Note:** Automatic synchronization selectively synchronizes only publications that contain automatic publication items.

Table 4–5 lists the name and description of parameter for the `selectPub` method.

Table 4–25 selectPub 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, which turns off selective sync. For more information, see Section 2.4, "Creating Publications Using Oracle Database Lite APIs".

4.3.5 OSEException Class

This class signals a non-recoverable error during the synchronization process. The `OSEException()` class constructs a `clear` object. The parameters for the constructor are listed in Table 4–26:

Constructors

`OSEException()`

`OSEException(int errorCode, string errorMessage)`

Table 4–26 OSEException Constructor Parameter Description

Parameter	Description
`errorCode`	Error codes are provided within the `OSEExceptionConstants` class, which are listed in "Error Codes and Messages". See the Javadoc for full details.
`errorMessage`	A readable text message, which are listed in "Error Codes and Messages", that provides extra information.

Public Methods

The methods for the `OSEException` are listed in Table 4–27.

Table 4–27 OSEException Class Public Methods

Parameters	Description
`int getErrorCode()`	Gets the error code.
`String getErrorMessage`	Gets the error message.

Error Codes and Messages

Table 4–28 lists the error codes and messages that can be returned in the `OSEException` class.

Table 4–28 OSEException Error Messages

Error Code	Error Message	Error number
DATABASE_NOT_FOUND	Could not find database <*database_name*>.	-12002
EMPTY_PASSWORD	Blank password is not allowed.	-12101
EMPTY_USER	User name cannot be blank.	-12105
ENCRYPTION_ID_MISMATCH	Sent encryption id <*id*> + 1 does not match received <*id*>.	-12030
ERR_CREDENTIALS	Failed to get credentials from the server (the current credentials are invalid or missing).	-12104
HTTP_RESPONSE	Unsuccessful HTTP response	-12035
INTERNAL_ERROR	Internal error has occured (see the cause).	-12039
INVALID_DML_TYPE	Received invalid record DML type <*dml_type*>.	-12005
INVALID_ENCRYPTION_TYPE	Invalid encryption type specified: <*encryption_type*>.	-12034
INVALID_OPCODE	Received invalid opcode <*opcode*>.	-12006
INVALID_PRIORITY	Invalid priority specified: <*priority*>.	-12047
INVALID_STRING_LENGTH	Invalid string length <*number*> received in opcode <*opcode*>.	-12103
INVALID_SYNC_DIRECTION	Invalid sync direction specified: <*sync_direction*>.	-12012
INVALID_TRANSPORT_TYPE	Invalid transport type specified: <*transport_type*>.	-12016
MISSING_DEFAULT_DATABASE	Plugin is missing default database needed to create snapshot.	-12040
OPCODE_LENGTH_UNDERRUN	<*numbytes*> bytes for opcode <*opcode*> have not been read.	-12008
OPCODE_OUT_OF_SEQUENCE	Opcode <*opcode*> was not expected at this time.	-12024
PASSWORD_NOT_SPECIFIED	Password is not specified and was not saved for user <*username*>.	-12004
PLUGIN_CLASS_INIT_FAILED	Failed to initialize plugin class <*class_name*>.	-12100
PLUGIN_EXCEPTION	Plugin has thrown an exception, see the cause.	-12011
PLUGIN_CLASS_NOT_FOUND	Could not find plugin class <*class_name*>.	-12044
PLUGIN_ID_NOT_FOUND	Could not find plugin with id <*id*>.	-12017
PUBLICATION_ID_MISMATCH	Publication id <*id*> for snapshot with id <*id*> does not match publication id <*id*> in the current transaction.	-12102
PUBLICATION_ID_NOT_FOUND	Could not find publication with id <*id*>.	-12043
PUBLICATION_NOT_FOUND	Could not find publication <*publication_name*>.	-12019
SERVER_ERROR	Server error, id = <*id*>.	-12038
SNAPSHOT_ID_EXISTS	Snapshot with id <*id*> already exists.	-12023
SNAPSHOT_ID_NOT_FOUND	Could not find snapshot with id <*id*>.	-12021
SYNC_CANCELED	Sync was canceled.	-12000
UNCOMPRESSED_DATA	Received erroneous uncompressed data.	-12032
UNENCRYPTED_DATA	Received erroneous unencrypted data.	-12031

Table 4–28 (Cont.) OSEException Error Messages

Error Code	Error Message	Error number
UNEXPECTED_BLOB_DATA	Got a record with BLOBs for a plugin that does not support BLOBs.	-12018
UNEXPECTED_OPCODE	Expecting opcode *<opcode>*, received *<opcode>*.	-12001
USER_NOT_SPECIFIED	User is not specified and the last user was not saved.	-12003

4.4 Synchronization API for C#

The C# interface for Mobile Sync client-side synchronization resides in the `Oracle.DataAccess.Lite` package.

The C# 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 C# interface is contained in the `Oracle.DataAccess.Lite.dll`. The `OracleSync` class, defined in the `OracleSync.cs` file, contains the C# API for synchronization.

The following sections describe how to use the C# API:

- Section 4.4.1, "Use the OracleSync Class for Synchronization"
- Section 4.4.2, "Using the OracleEngine to Synchronize"
- Section 4.4.3, "Exception Handling and Reading Log Files"
- Section 4.4.4, "Monitor Synchronization Progress With the SyncEventHandler"
- Section 4.4.5, "Manage What Tables Are Synchronized With Selective Sync"
- Section 4.4.6, "Enable File-Based Synchronization through C# APIs"

4.4.1 Use the OracleSync Class for Synchronization

You use the `OracleSync` class to initialize and perform synchronization using the C# APIs.

Table 4–29 shows the properties you can set for initializing the environment before invoking synchronization:

Table 4–29 OracleSync Properties

Properties	Type	Description
UserName	String	The name of the user who is initiating the synchronization.
Password	String	The password for the user who is initiating the synchronization.
ServerURL	String	The hostname or IP address of the Mobile Server. If you want to use SSL, set the URL using the `HTTPS` prefix instead of the `HTTP` prefix.
ProxyHost	String	The hostname or IP address of the proxy server.
ProxyPort	Integer	The hostname or IP address of the proxy port.

Table 4–29 (Cont.) OracleSync Properties

Properties	Type	Description
PushOnly	Boolean	Upload changes from the client to the server only, do not download. This is useful when data transfer is one way, client to server. A boolean value which makes synchronization push only. TRUE sets PushOnly to on; FALSE is the default value.
HighPriority	Boolean	FALSE turns high priority to OFF, which is the default. TRUE turns high priority to ON; Only high priority table or rows are synchronized when set to TRUE. You can only use fast refresh with a high priority restricting predicate. If you use any other type of refresh, the high priority restricting predicate is ignored. See Section 1.2.10, "Priority-Based Replication" in the *Oracle Database Lite Troubleshooting and Tuning Guide* for more information.
Option	Option	Set one or more of the appropriate sync options with an OR statement from the `SyncOption` enumerator. Provide a bitset of the following flags: ■ `FORCE_REFRESH` Forces a purge of all existing data and do a full refresh.
Secure	Boolean	Should not be used. To use an SSL connection, prefix the ServerURL property with `https://`.

The synchronization public methods of the `OracleSync` class are described in Table 4–30.

Table 4–30 OracleSync Public Methods

Method	Description
`void save()`	This saves the username, password, URL, proxy, and proxy-port settings for the `OracleSync` class. The next time that the `OracleSync` class is created, it is pre-loaded automatically with these saved settings. The password is not saved.
`void Synchronize ()`	Starts a synchronization session and blocks that thread until synchronization is complete.
`void Close ()`	Closes and performs clean-up for the synchronization session.
`void SetCancel()`	Set cancel during any synchronization. The synchronization returns.
`String GetLogFileName()`	Used to retrieve the error log filename. For details, see Section 4.4.3, "Exception Handling and Reading Log Files".
`void SetTableSyncFlag (string pubName, string tableName, bool flag)`	Used for setting selective sync. For details, see Section 4.4.5, "Manage What Tables Are Synchronized With Selective Sync".

Table 4–30 (Cont.) OracleSync Public Methods

Method	Description
`int SetSyncProperty (string prop, string val)`	Used to set Sync properties. The types of Sync properties that you can set are as follows: ■ `HTTP_USER_PROP` —HTTP authentication user name. ■ `HTTP_PASS_PROP` — HTTP authentication password. ■ `USER_NAME_PROP` —User context when a member is synchronizing. ■ `USER_PASS_PROP` —User context password, when a member is synchronizing. ■ `SEND_FILE_PROP` — Send file name for file based synchronization ■ `RECEIVE_FILE_PROP` —Receive file name for file based synchronization. For details on file-based synchronization, see Section 4.4.6, "Enable File-Based Synchronization through C# APIs".
`int GetSyncProperty (string prop, out string val)`	Used to retrieve Sync property settings. See the SetSyncProperty description for details.

To create the `OracleSync` class in preparation for the synchronization, perform the following:

1. Instantiate the `OracleSync` object.
2. Set relevant properties, such as username, password and URL. The username and password are limited to 28 characters each.
3. If you want to preserve all `OracleSync` properties, except for the password, then execute the `Save` method. The format of the `Save` method is as follows:

   ```
   void Save()
   ```

4. Perform the synchronization with the `Synchronize` method. The format of the `Synchronize` method is as follows:

   ```
   void Synchronize()
   ```

Note: A `DataException` is thrown if synchronization fails. Also, you must close all database connections before doing a synchronization.

5. Close the `OracleSync` object when finished. The `Close` method is as follows:

   ```
   void Close()
   ```

The following code demonstrates these methods by setting the properties off of user entries on a GUI screen:

```
// Instantiate the object
OracleSync m_sync = new OracleSync();

// Set the appropriate sync options
```

```
m_sync.UserName  = userName.Text;
m_sync.Password  = password.Text;
m_sync.ServerURL = url.Text;

if (enableProxy.Checked == true)
{
  m_sync.ProxyHost = proxyHost.Text;
  m_sync.ProxyPort = 80;

  try {m_sync.ProxyPort = System.Convert.ToInt32 (proxyPort.Text);}
  catch (System.ArgumentNullException) {}
}
if (forceRefresh.Checked == true)
{
 m_sync.Option = SyncOption.FORCE_REFRESH;
}
// save the options before synchronization
m_sync.Save();
//Synchronize
m_sync.Synchronize();
//Close the OracleSync object
m_sync.Close();
m_sync = null;
```

4.4.2 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.

- Section 4.4.2.1, "Launch the MSYNC Tool for User Input"
- Section 4.4.2.2, "Set the Environment and Synchronize With the OracleEngine"

4.4.2.1 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.

4.4.2.2 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 4–31 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 4–31.

```
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 4–31 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.
`/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.

4.4.3 Exception Handling and Reading Log Files

For any synchronization error or database error that occurs during synchronization, then the `Oracle.DataAccess.Lite.OracleException` is thrown. The `OracleException` object contains the error code and error messages.

If an error occurs during synchronization, you can view errors returned by the server in the error log file. To retreve the error log filename, execute the `GetLogFileName` method. Then, open and evaluate the log file, which is written into the working directory for the application. The syntax for the `GetLogFileName` method is as follows:

```
static String GetLogFileName();
```

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.

The following code example shows how to retrieve the filename:

```
String file = OracleSync.GetLogFileName();
```

4.4.4 Monitor Synchronization Progress With the SyncEventHandler

To monitor the progress of the synchronization process, the `SetEventHandler` and `SyncEventHandler` methods and the `SyncEventArgs` object of the `OracleSync` class enable the user to create an event handler to return a progress report on the state of the synchronization.

The `SyncEventArgs` object is generated in the `OracleSync` object during the `Synchronize` method.

The following sections describe and show how to use the object and the methods to monitor the synchronization stage and progress:

- Section 4.4.4.1, "Using the SyncEventArgs Object"
- Section 4.4.4.2, "Executing the SetEventHandler Method"
- Section 4.4.4.3, "Creating the SyncEventHandler Object"

4.4.4.1 Using the SyncEventArgs Object

The `SyncEventArgs` is an object that contains the state or the synchronization.

The following is the definition for the object:

```
public class SyncEventArgs : EventArgs
{
 public readonly int stage;
 public readonly int percentage;
 // Synchronization progress stages
 //
 public const int SYNC_PREPARE_START  = 0;
 public const int SYNC_PREPARING      = 1;
 public const int SYNC_PREPARE_FINISH = 2;
 public const int SYNC_SEND_START     = 3;
 public const int SYNC_SENDING        = 4;
 public const int SYNC_SEND_FINISH    = 5;
 public const int SYNC_RECEIVE_START  = 6;
 public const int SYNC_RECEIVING      = 7;
 public const int SYNC_RECEIVE_FINISH = 8;
 public const int SYNC_PROCESS_START  = 9;
 public const int SYNC_PROCESSING     = 10;
 public const int SYNC_PROCESS_FINISH = 11;
}
```

Note: Some codes are returned from the OCI layer. If you receive a status code not listed here, see Table 4–7.

There are two parameters, as follows:

Table 4–32 SyncEventArgs Parameters

Parameter	Description
stage	The stage in which the synchronization is acting on the Mobile client. The possible stages are as follows: ■ SYNC_PREPARE_START ■ SYNC_PREPARING ■ SYNC_PREPARE_FINISH ■ SYNC_SEND_START ■ SYNC_SENDING ■ SYNC_SEND_FINISH ■ SYNC_RECEIVE_START ■ SYNC_RECEIVING ■ SYNC_RECEIVE_FINISH ■ SYNC_PROCESS_START ■ SYNC_PROCESSING ■ SYNC_PROCESS_FINISH
percentage	The percentage of completion for this stage.

The following code demonstrates how you can determine the state and percentage of the synchronization from the `SyncEventArgs`. For each state group indicated in the `args.stage` parameter, it modifies a display to show the state and the percentage of completion by invoking the `moveProgressBarDelegate` method, as follows:

```
private void DisplayProgress (object sender, SyncEventArgs args)
{
 switch (args.stage)
 {
  case SyncEventArgs.SYNC_PREPARE_START:
  case SyncEventArgs.SYNC_PREPARING:
  case SyncEventArgs.SYNC_PREPARE_FINISH:
  this.Invoke(new moveProgressBarDelegate(moveProgressBar),
        prepareBar, args.percentage );
  break;
  case SyncEventArgs.SYNC_SEND_START:
  case SyncEventArgs.SYNC_SENDING:
  case SyncEventArgs.SYNC_SEND_FINISH:
  this.Invoke(new moveProgressBarDelegate(moveProgressBar),
        sendBar, args.percentage);
  break;
  case SyncEventArgs.SYNC_RECEIVE_START:
  case SyncEventArgs.SYNC_RECEIVING:
  case SyncEventArgs.SYNC_RECEIVE_FINISH:
  this.Invoke(new moveProgressBarDelegate(moveProgressBar),
        receiveBar, args.percentage);
  break;
  case SyncEventArgs.SYNC_PROCESS_START:
  case SyncEventArgs.SYNC_PROCESSING:
```

```
  case SyncEventArgs.SYNC_PROCESS_FINISH:
  this.Invoke(new moveProgressBarDelegate(moveProgressBar),
        processBar, args.percentage);
  break;
 }
}
```

4.4.4.2 Executing the SetEventHandler Method

Before you can determine the state, you must create the event handler that monitors the synchronization progress. This is performed by executing the `SetEventHandler` method and providing a new `SyncEventHandler` object for it to track.

The following is the definition for the method:

```
public void SetEventHandler (SyncEventHandler handler, bool add);
```

The parameters for the `SetEventHandler` method are described in Table 4–33.

Table 4–33 SetEventHandler Method Parameters

Parameter	Description
object sender	The `SyncEventHandler` object that is created for this event handler.
boolean add	This boolean, if true, registers the `SyncEventHandler` object. If false, it de-registers the event handler.

The following code demonstrates how to create a new `SyncEventHandler` and delegates the `SyncEventHandler` method to the `DisplayProgress` method, which is the application implemented callback method that processes the state and percentage.

The `SetEventHandler` takes in the `SyncEventHandler` delegate, which is assigned as `DisplayProgress`, and whether to monitor the progress with a `TRUE` or to not monitor the progress with `FALSE`. Then the `Synchronize` method is called to initiate the synchronization.

```
//Create the SyncEventHandler and put it in the event handler
m_sync.SetEventHandler (new
        OracleSync.SyncEventHandler (DisplayProgress), true);
//Perform the synchronize
 m_sync.Synchronize();
//Once the synchronization is complete, remove this SyncEventHandler
 m_sync.SetEventHandler (new
        OracleSync.SyncEventHandler (DisplayProgress), false);
```

4.4.4.3 Creating the SyncEventHandler Object

The `SyncEventHandler` object is the event handler. It is also a delegate method. The following is the definition for the method:

```
delegate void SyncEventHandler (object sender, SyncEventArgs args);
```

In the application implementation, create the delegate method with the same arguments and how you want the delegated method to handle the event. In our example, the `DisplayProgress` method is defined as follows:

```
void DisplayProgress (object sender, SyncEventArgs args)
```

It has the same arguments as the delegate definition, and is defined as the delegate when the `SyncEventHandler` is created, as shown in the code below:

```
m_sync.SetEventHandler (new
        OracleSync.SyncEventHandler (DisplayProgress), true);
```

The `SyncEventHandler` object, and thus the `DisplayProgress` method, takes in two parameters, as shown in Figure 4–34.

Table 4–34 SetEventHandler Method Parameters

Parameter	Description
`object sender`	The `SyncEventHandler` object that is created for this event handler.
`SyncEventArgs args`	The `SyncEventArgs` object that monitors the state and progress of the synchronization.

On each synchronization event—such as send complete, receive complete, synchronization complete—the `OracleSync` object raises the `SyncEventHandler` event and invokes the `DisplayProgress` method with the `SyncEventArgs` object.

4.4.5 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 `Synchronize` 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

```
void SetTableSyncFlag (string pubName, string tableName, bool sync_flag)
```

Table 4–5 lists the name and description of parameters for the `setTableSyncFlag` function.

Table 4–35 setSyncFlag Parameters

Name	Description
`pubName`	The name of the publication which is being synchronized. If the value for the `pubName` is `NULL`, it means all publications in the database. This string is the same as the client name supplied to the Consolidator Manager when creating the publication. In most cases, you will use `NULL` for this parameter. For more information, see Section 2.4, "Creating Publications Using Oracle Database Lite APIs".

Table 4–35 (Cont.) setSyncFlag Parameters

Name	Description
`tableName`	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 2.4, "Creating Publications Using Oracle Database Lite APIs".
`sync_flag`	If the `sync_flag` is set to TRUE, you must synchronize the publication. If the `sync_flag` is set to FALSE, then do not synchronize. The value for the `sync_flag` is not stored persistently. Each time before `Synchronize()`, you must call `SetTableSyncFlag()`.

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` = FALSE, the table is not synchronized. To synchronize specific tables only, you must perform the following steps:

1. Disable the default setting by setting it to FALSE (0). By default, the setting is set to TRUE (1) for all the tables. Setting them all to FALSE then enables you to select which tables are to be synchronized, which is performed in step 2.

   Example:

   ```
   SetTableSyncFlag(<publication_name>,null,0)
   ```

   Where `<pubName>` 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:

   ```
   SetTableSyncFlag(<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.

```
m_sync.SetTableSyncFlag("","",(short)1); //turn on sync flag for all the tables
m_sync.SetTableSyncFlag("","OrdersODB.TEST",(short)0);
                      //turn off sync flag for OrdersODB.TEST
```

4.4.6 Enable File-Based Synchronization through C# APIs

When you want to use file-based synchronization, you must enable file-based synchronization. Once enabled, then when synchronization occurs—either through automatic or manual synchronization—then the synchronization file is created. See Section 6.8, "Synchronizing to a File Using File-Base Sync" in the *Oracle Database Lite Administration and Deployment Guide* for more details on file-based synchronization.

To enable file-based synchronization programmatically with the `OracleSync` class, specify the following:

1. Instantiate the `OracleSync` class and set the username and password.
2. Specify the synchronization direction and directory and filename for the synchronization file with the `SetSyncProperty` method of the `OracleSync` class. The synchronization direction is either send, which creates the synchronization file, or receive, which takes in a file from the Mobile Server. These

are configured in the SEND_FILE_PROP and RECEIVE_FILE_PROP properties with the SetSyncProperty method.

- When you set the RECEIVE_FILE_PROP property with the filename and directory, the intended file is uploaded and processed within the Mobile client.
- When you set the SEND_FILE_PROP property with the filename and directory, the intended file is created with the Mobile client transactions destined for the Mobile Server.

Note: You can retrieve the filename with the GetSyncProperty method.

The SetSyncProperty and GetSyncProperty methods are as follows:

```
int SetSyncProperty (string prop, string val)
int GetSyncProperty (string prop, out string val)
```

- For SetSyncProperty, provide the property and the filename for setting the direction of the file-based synchronization and the destination/origination filename.
- For GetSyncProperty, provide the direction property and receive the filename in the OUT value.

The following code example sets the direction, filename, username and password in the SetSyncProperty method. In this example, the SEND_FILE_PROP property is set with the filename and direction, which tells the Sync Client to marshall the Mobile client transactions that are to be uploaded to the Mobile Server into this file. If you were receiving a synchronization file from the Mobile Server, you would have set the RECEIVE_FILE_PROP property with the location and name of this file.

Finally, perform the synchronization with the Synchronize method.

```
OracleSync m_sync = new OracleSync();
m_sync.UserName = "S11U1";
m_sync.Password = "manager";
m_sync.SetSyncProperty(OracleSync.SEND_FILE_PROP,"C:\\temp\\send1.bin");
m_sync.Synchronize();
```

4.5 mSync/OCAPIs/mSyncCom

For more information, refer to the *Oracle Database Lite API Specification*.

5 Application Development

The following sections discuss how to develop applications for Oracle Database Lite:

- Section 5.1, "Data Access APIs"
- Section 5.2, "Supported Native APIs for Oracle Database Lite"
- Section 5.3, "Developing Java Applications"
- Section 5.4, "Using Stored Procedures in Oracle Database Lite"
- Section 5.5, "Developing Mobile Web-to-Go Applications"

5.1 Data Access APIs

Table 5–1 lists the supported APIs for accessing data within the Oracle Lite database:

Table 5–1 Supported APIs

Native API	Description
JDBC	Use JDBC to access the Oracle Lite database. See Section 5.1.2, "JDBC" or Oracle Database JDBC manuals for more information.
ODBC	Use ODBC to access the Oracle Lite database. See Section 5.1.3, "ODBC".
.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 5.1.4, "ADO.NET" for more information.
SODA	Simple Object Data Access (SODA) for object and relational database development. See Chapter 12, "Using Simple Object Data Access (SODA)" in the *Oracle Database Lite Client Guide* 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 5.1.4, "ADO.NET" for more information.

These data access APIs are described in Chapter 5, "Oracle Database Lite Data Access APIs"" in the *Oracle Database Lite Client Guide*. The focus in the client guide is for accessing the Oracle Lite database on the client without the Mobile Server synchronization feature. However, if you are using the Mobile Server product and providing synchronization, then the following sections describe the differences to take into account for these APIs in the Mobile Server environment.

Note: After you perform a complete installation of Oracle Database Lite, you can view samples that show how to access data within the Oracle Lite database. The samples are available in your <*ORACLE_HOME*>`\Mobile\Sdk` directory. The tools, locations for samples, and descriptions are described in Section 2.5, "Using Oracle Lite Samples" in the *Oracle Database Lite Client Guide*. .

- Section 5.1.1, "Data Source Name"
- Section 5.1.2, "JDBC"
- Section 5.1.3, "ODBC"
- Section 5.1.4, "ADO.NET"

5.1.1 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.

5.1.2 JDBC

For Mobile clients within the Mobile Server environment, all JDBC drivers are provided for you to use within the Oracle Database Lite binaries. See Chapter 7, "JDBC Programming" in the *Oracle Database Lite Client Guide* for more information on programming with JDBC.

5.1.3 ODBC

The Mobile Server supports Level 3 compliant ODBC 2.0 and the ODBC 3.5 drivers. The ODBC 2.0 driver is installed and used by default for all Oracle Database Lite components. The ODBC 3.5 driver should be used solely for the standalone application that uses an embedded Oracle Lite database.

Note: You cannot use ODBC 3.5 for any multi-user listener application or Branch Office scenario.

If you want to use the ODBC 3.5 driver for an embedded application on a Mobile client, you must install and configure the ODBC 3.5 driver on the client in one of the following methods:

- Automatic installation: Configure the INF file for automatic installation and configuration: Modify the INF file to instruct the client install to automatically download the ODBC 3.5 DLLs to the Mobile client and register the ODBC 3.5 driver and the DSN in the `ODBC.INI` file. See the following sections on how to modify the INF file: Section 8.10.5.2, "FILE Section" and Section 8.10.5.5, "ODBC Section" in the *Oracle Database Lite Administration and Deployment Guide*.

- Manual installation: See Chapter 6, "ODBC Drivers" in the *Oracle Database Lite Client Guide* on how to install and configure the ODBC 3.5 DLL from the MDK to the Mobile client.

For a full description of the ODBC drivers, see Chapter 6, "ODBC Drivers" in the *Oracle Database Lite Client Guide*.

5.1.4 ADO.NET

The Oracle Database Lite ADO.NET provider resides in the `Oracle.DataAccess.Lite` namespace. Use this programming interface to access Oracle Database Lite and trigger data synchronization in .NET applications. The ADO.Net classes are described in Chapter 11, "Oracle Database Lite ADO.Net Provider" in the *Oracle Database Lite Client Guide*. However, since the *Oracle Database Lite Client Guide* describes only the Oracle Lite client, it does not include any details on how to enable the client as a Mobile client with synchronization. The following sections describe the ADO.Net classes that enable synchronization and create a client database for testing:

- Section 5.1.4.1, "Data Synchronization With the OracleSync or Oracle Engine Classes"
- Section 5.1.4.2, "Creating a Database for Testing"
- Section 5.1.4.3, "Developing an ADO.NET Application on WinCE"

5.1.4.1 Data Synchronization With the OracleSync or Oracle Engine Classes

You can perform a synchronization programatically with the `OracleSync` or `OracleEngine` classes. For full details, see Section 4.4, "Synchronization API for C#".

5.1.4.2 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 4.4.2, "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).

The following is the signature of the `CreateDatabase` method:

```
OracleEngine.CreateDatabase (string dsn, string db, string pwd)
```

5.1.4.3 Developing an ADO.NET Application on WinCE

For an example of how to develop an ADO.NET application, see Chapter 14, "Tutorial for Building Mobile Applications for Windows CE".

5.2 Supported Native APIs for Oracle Database Lite

The C, C++, C# APIs use ODBC to access the Oracle Lite database. Use Oracle-specific APIs for programmatic synchronization. See Section 4.1, "Synchronization APIs For C or C++ Applications" for more information.

5.3 Developing Java Applications

The following sections describe how to develop and test Java applications:

- Section 5.3.1, "Java Support for Applications"
- Section 5.3.2, "Oracle Database Lite Java Development Environment"
- Section 5.3.3, "Java Development Tools"

5.3.1 Java Support for Applications

Table 5–2 lists the Java support provided for each platform in Oracle Database Lite.

Table 5–2 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 5.3.1.1, "JDBC Drivers".	Yes	Yes	Yes On Linux, only JDBC and ODBC access is supported.	Chapter 7, "JDBC Programming" in the *Oracle Database Lite Client Guide*
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	See Chapter 10, "Using Stored Procedures and Triggers" in the *Oracle Database Lite Client Guide*
Java Server Pages	1.1	N/A	N/A	N/A	Section 5.5.2.4, "Developing Java Server Pages"
Java Servlet	2.2	N/A	N/A	N/A	Section 5.5.2.5, "Developing Java Servlets for Web-to-Go" and Section 5.5.2.8, "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 2, "Synchronization".

5.3.1.1 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 connnection. 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.

5.3.2 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 5.3.2.1, "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`.

5.3.2.1 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.

5.3.3 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.

5.4 Using Stored Procedures in Oracle Database Lite

A stored procedure is a method that is stored in Oracle Database Lite. The procedure can be invoked by applications that access the database. Stored procedures can return a single value, a row, or multiple rows.

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.

The description of how to create, load and define stored procedures for the Mobile client are contained in Chapter 10, "Using Stored Procedures and Triggers" in the *Oracle Database Lite Client Guide*. However, if you are using the Mobile Server, then there are a few instructions on how to load and define stored procedures when using the Mobile Server.

The following sections describe how to load and define stored procedures in a Mobile Server environment.

- Section 5.4.1, "Load and Define Java Stored Procedures"
- Section 5.4.2, "Load and Define C, C++, or C# Stored Procedures"

5.4.1 Load and Define Java Stored Procedures

You can either use Java stored procedures in a standalone Oracle Lite database or integrated within an enterprise Mobile system that uses synchronization. Each of these require a different method to load and define the Java stored procedure, as described in the following sections:

- Section 5.4.1.1, "Load and Define Java Stored Procedures on the Mobile Client in an Oracle Lite Database"
- Section 5.4.1.2, "Load and Define Java Stored Procedures in an Enterprise Mobile Server Environment"

5.4.1.1 Load and Define Java Stored Procedures on the Mobile Client in an Oracle Lite Database

The methods for loading and defining the Java stored procedures are described in Section 10.2.1, "Load and Define Java Stored Procedures" in the *Oracle Database Lite Client Guide*.

5.4.1.2 Load and Define Java Stored Procedures in an Enterprise Mobile Server Environment

When you are in an enterprise environment where you are using the Mobile option including synchronization between the Mobile Server and the Mobile clients, then you can have the Java stored procedure automatically downloaded to each client and loaded into the client Oracle Lite database.

You can load and define the Java stored procedure in an enterprise Mobile environment in one of following ways:

- Section 5.4.1.2.1, "Using MDW to Store Java Stored Procedure in the Repository"

- Section 5.4.1.2.2, "Using the Consolidator Manager API to Store the Java Stored Procedure in the Repository"

5.4.1.2.1 Using MDW to Store Java Stored Procedure in the Repository When you are creating the publication using MDW, part of the process is a section to load the Java stored procedure, as detailed below:

1. Archive the Java stored procedure into a JAR file. You cannot upload the CLASS file to MDW.
2. Upload the JAR file as a Resource for the publication within MDW. Once loaded as a Resource, then it will be loaded into the Oracle Lite database on the client on the first synchronization.
3. Create a SQL script with DDL that uses one of the definition methods on the Java stored procedure: `ATTACH`, `CREATE METHOD`, `CREATE PROCEDURE` or `CREATE TRIGGER`. Once created, add the script to the publication as a Script. The script is automatically executed after download to the client.

Note: You can also perform these functions through the Consolidator Manager API. See the *Oracle Database Lite API Specification* documentation for more information.

5.4.1.2.2 Using the Consolidator Manager API to Store the Java Stored Procedure in the Repository On the server, you can invoke the following ConsolidatorManager APIs to create the Java stored procedure and to add it to the publication.

1. Invoke the `createStoredProc` API to load and define the Java stored procedure in the server. Provide the name of the Java class and the method that you want defined.

   The Consolidator loads the Java stored procedure into the repository and creates the call specification for defining the method on the client within the repository.

   Note: You cannot use the attach for definition; this method creates the call specification only with `CREATE FUNCTION` or `CREATE PROCEDURE`.

   ---

2. Invoke the `addJavaResourceAndScripts` API to add the Java stored procedure to the publication.

When the client first synchronizes, the application and Java stored procedure contained within the publication are downloaded onto the client. The Java stored procedure is then loaded into the Oracle Lite database and the call specification is used to define the method within the Java stored procedure.

5.4.2 Load and Define C, C++, or C# Stored Procedures

Once you create the C, C++ and C# stored procedures, you still need to load and define them on the client Oracle Lite database. This section describes how to perform this manually for a standalone client Oracle Lite database or how you can implement a solution for automatically loading and defining these stored procedures to multiple clients.

- Section 5.4.2.1, "Defining the C, C++, or C# Stored Procedure"

- Section 5.4.2.2, "Loading the C, C++ or C# Stored Procedure to the Oracle Lite Database"

5.4.2.1 Defining the C, C++, or C# Stored Procedure

For details on how to define the C, C++, or C# stored procedure, see Section 10.10, "Defining the C, C++, or C# Stored Procedures" in the *Oracle Database Lite Client Guide*.

5.4.2.2 Loading the C, C++ or C# Stored Procedure to the Oracle Lite Database

When you have a standalone client, then you can load the stored procedure by copying it directly to the client. Define the stored procedure by executing the `CREATE` statements on the Mobile device directly, as described in Section 5.4.2.1, "Defining the C, C++, or C# Stored Procedure".

However, if you are using the Mobile Server and have multiple clients on which you want to install and use these stored procedures, then you may want to automate the load and definition process for these stored procedures.

Use one of the following methods to download the DLL containing the stored procedures and to define the stored procedures:

- Option 1: Extend the Client Platform and Use Scripts
  1. To download the DLL, extend the client platform and adding the required DLL. When the users download the `setup.exe` for your extended platform, then the DLL is automatically downloaded and installed. When you use this option, then you should install the DLL in the same directory as the `msync.exe` and `ocapi.dll` files on the client.
  2. Create a script that issues the `CREATE` commands, as defined in Section 5.4.2.1, "Defining the C, C++, or C# Stored Procedure", that defines the stored procedures or triggers. Once you create the SQL script, then you can load the script into the publication using either the Add Script function within MDW or the Consolidator APIs. Once added to the publication, then the synchronization downloads the script with the application and it is executed upon completion of the download. In this case, the DLL containing the stored procedures must already be loaded onto the device before the script is executed. When loading the DLL, ensure that it is within the PATH or for C#, within the Global Assembly Cache.
- Option 2: Issue the Definition From Within Your Application
  1. Package the DLL containing the stored procedures with the other application files. Then, when you load the package, the DLL is loaded when the application is loaded.
  2. Implement the application to issue the definition `CREATE` statements within an initialization routine that is executed the first time the application runs.

  This option is simpler than extending an existing platform, because the DLL can be part of the application itself.

5.5 Developing Mobile Web-to-Go Applications

The following sections describe how to develop and test Web-to-Go applications:

- Section 5.5.1, "Choose the Type of Web-to-Go Mobile Client to Use"
- Section 5.5.2, "Developing and Testing the Application"

5.5.1 Choose the Type of Web-to-Go Mobile Client to Use

To install and set up the Mobile Client, see Section 11.5.1, "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.

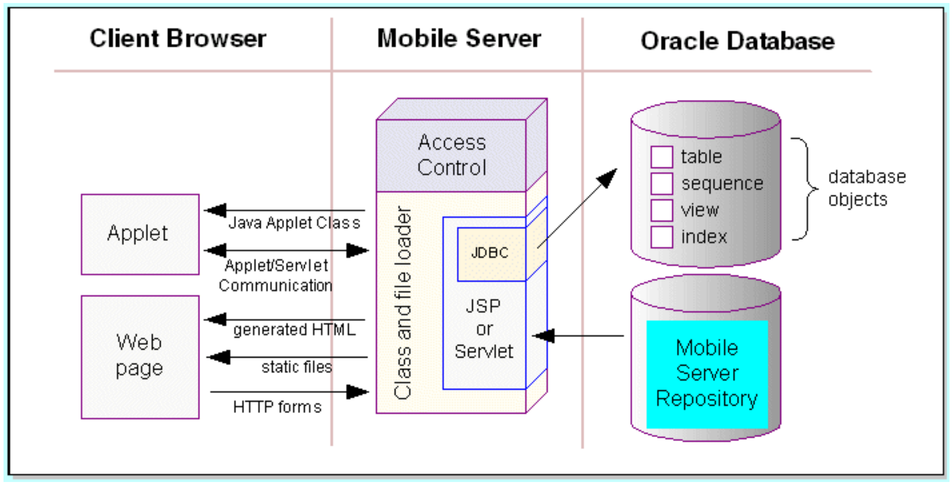
5.5.2 Developing and Testing the Application

Web-to-Go provides a Java API for 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 enables developers to develop and debug Web-to-Go applications that contain Java applets, Java servlets, and JavaServer Pages (JSP).

Figure 5–1 displays the development architecture of the Mobile Server and the Oracle database.

Figure 5–1 Development Architecture



The following sections provide a discussion on how to develop Mobile applications for Web-to-Go. Topics include:

- Section 5.5.2.1, "Building Web-to-Go Applications"
- Section 5.5.2.2, "Database Connections"
- Section 5.5.2.3, "Application Roles"

- Section 5.5.2.4, "Developing Java Server Pages"
- Section 5.5.2.5, "Developing Java Servlets for Web-to-Go"
- Section 5.5.2.6, "Using Web-to-Go Applets"
- Section 5.5.2.7, "Developing Applets to use JDBC Communication"
- Section 5.5.2.8, "Developing Applet Servlet Communication"
- Section 5.5.2.9, "Debugging Web-to-Go Applications"
- Section 5.5.2.10, "Customizing the Workspace Application"
- Section 5.5.2.11, "Using the Mobile Server Admin API"

5.5.2.1 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 5–3 provides examples of each database component type.

Table 5–3 Database Component Types

Component Type	Example
Static Component	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).
Dynamic Component	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.
Database Component	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 Win32 and WinCE platforms.

5.5.2.2 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.

5.5.2.3 Application Roles

It is common for applications to behave differently depending on the type of user executing the application. For example, an application may display 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 of `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.

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.

5.5.2.4 Developing Java Server Pages

Web-to-Go handles HTTP requests for Java Server Pages (JSP) using the Mobile Client Web Server, Mobile Server, and Mobile Client for Web-to-Go, as described in the following sections:

- Section 5.5.2.4.1, "Mobile Server or Mobile Development Kit Web Server"
- Section 5.5.2.4.2, "Mobile Client for Web-to-Go"

5.5.2.4.1 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.

5.5.2.4.2 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.

5.5.2.5 Developing Java Servlets for Web-to-Go

The following sections describe how to develop a Java Servlet for Web-to-Go:

- Section 5.5.2.5.1, "Using Mobile Development Kit for Web-to-Go for Development"
- Section 5.5.2.5.2, "Creating a Servlet"
- Section 5.5.2.5.3, "Obtaining Database Connectivity from a Servlet"
- Section 5.5.2.5.4, "Web-to-Go User Context for Authentication"

- Section 5.5.2.5.5, "Web-to-Go Servlet Uses Applet Package When Communicating With an Applet"
- Section 5.5.2.5.6, "Accessing the Mobile Server Repository from a Servlet"
- Section 5.5.2.5.7, "Debugging the Servlet Using WTGDEBUG.EXE"

5.5.2.5.1 Using Mobile Development Kit for Web-to-Go for Development Develop Web-to-Go Java servlets with the Mobile Development Kit, which simplifies the process of writing Mobile Server servlets. Before using the Mobile Development Kit for Web-to-Go, 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.

The Mobile Development Kit for Web-to-Go Web server is a scaled down version of the Mobile Server and has the following limitations.

- 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.
- Provides connection management only to Oracle Database Lite. It connects the user to the schema `SYSTEM` in the Oracle Database Lite named `webtogo`.

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.

5.5.2.5.2 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();
   }
}
```

5.5.2.5.3 Obtaining Database Connectivity from a Servlet 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.

```
HttpSession sess = request.getSession();
WTGUser user = (WTGUser)sess.getAttribute("x-mobileserver-user");
Connection conn = user.getConnection();
```

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

On the command-line on the client, you can connect to the Oracle Lite database locally outside the realm of the application with the mSQL command, as follows:

```
msql system/manager@jdbc:polite:webtogo
```

5.5.2.5.4 Web-to-Go User Context for Authentication 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");
   ...
}
```

5.5.2.5.5 Web-to-Go Servlet Uses Applet Package When Communicating With an Applet The `oracle.lite.web.applet` package provided by Web-to-Go 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*.

5.5.2.5.6 Accessing the Mobile Server Repository from a Servlet 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` method 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();
  ...
}
```

5.5.2.5.7 Debugging the Servlet 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.

5.5.2.6 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.

- Section 5.5.2.6.1, "Creating the Web-to-Go Applet"
- Section 5.5.2.6.2, "Creating the HTML Page for the Applet"

5.5.2.6.1 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.

5.5.2.6.2 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.

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`.

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();
    }
}
```

5.5.2.7 Developing Applets to use 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 5.5.2.6.1, "Creating the Web-to-Go Applet".

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;
}
```

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.

You can close the connection by calling the following statement.

```
dBConnection.close()
```

5.5.2.8 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.

5.5.2.8.1 Extend HttpServlet Class When creating a servlet, it 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();
   }
}
```

5.5.2.8.2 Communicate With the getResultObject Method 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;
}
```

5.5.2.8.3 Add SessionID With the setSessionID Method 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();
}
```

5.5.2.8.4 Display Static Documents With the showDocument Method 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);
}
```

5.5.2.9 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 JDeveloper, Borland's JBuilder, or Visual J++. The example in this section assumes you are using JDeveloper. However, most of the information provided is also relevant to other debuggers.

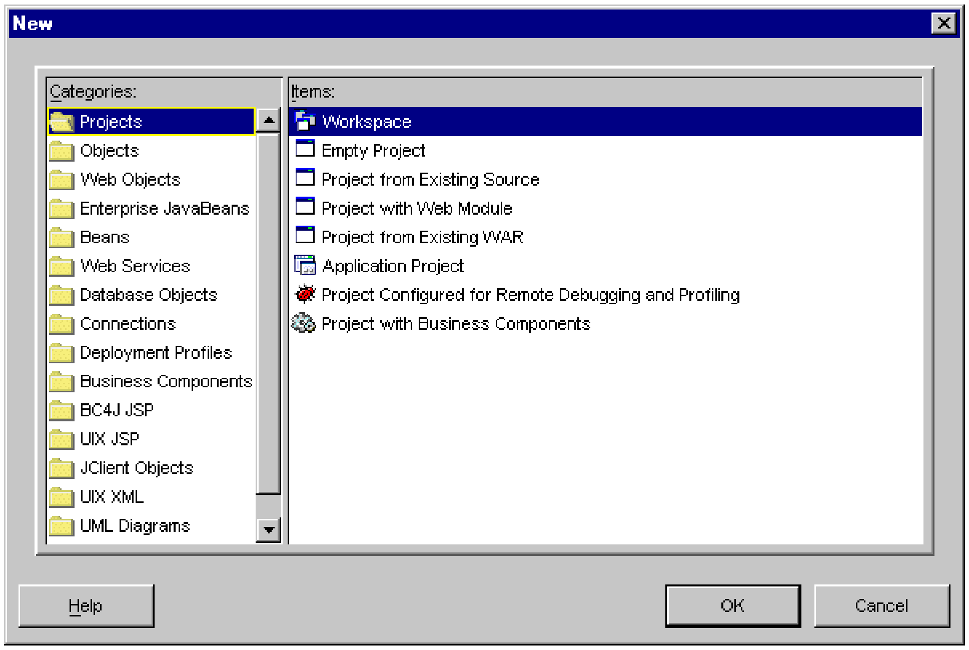
The following sections describe how to configure JDeveloper to run the Sample 1 application that is bundled with the Mobile Development Kit for Web-to-Go. For detailed information and full documentation on how to use JDeveloper, consult the online help in JDeveloper and the JDeveloper documentation.

- Section 5.5.2.9.1, "Creating a Debug Project"
- Section 5.5.2.9.2, "Creating a Library"
- Section 5.5.2.9.3, "Adding Files to the Project"
- Section 5.5.2.9.4, "Running and Debugging"
- Section 5.5.2.9.5, "Troubleshooting"

5.5.2.9.1 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 5–2, then select **Empty Project**.

Figure 5–2 Creating a New Project

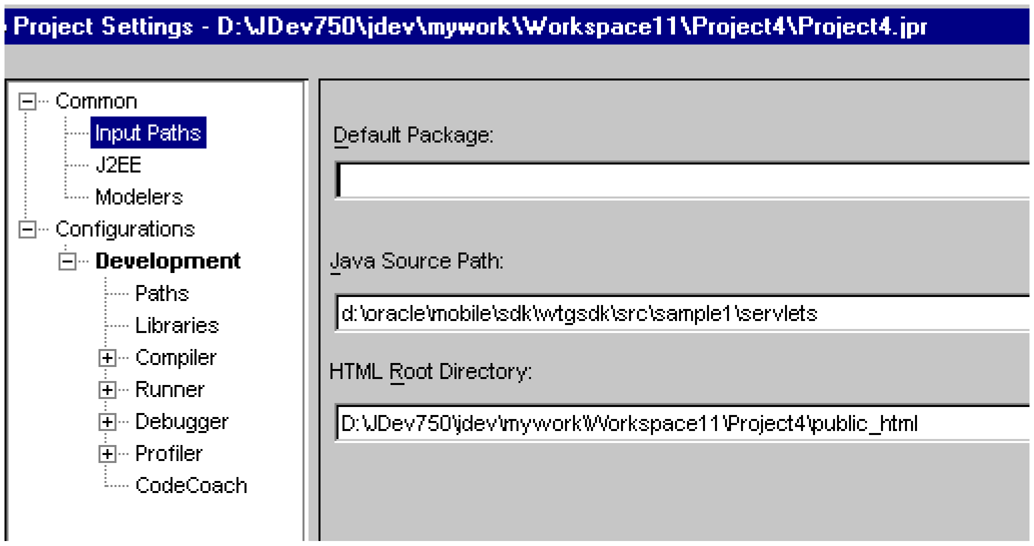


4. Set the Project Settings for your new project. Right click on **Project** to retrieve Project Settings. In the Project Settings dialog, expand Common in the left panel and select Input Paths. In the right panel, enter the following information in the Java Source Path field, as displayed in Figure 5–3.

   ```
   <ORACLE_HOME>\mobile\sdk\wtgsdk\src\sample1\servlets
   ```

   Leave the **Default Package** field blank. Do not change the default **HTML Root Directory**.

Figure 5–3 Project Settings - Input Paths



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`

5.5.2.9.2 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`

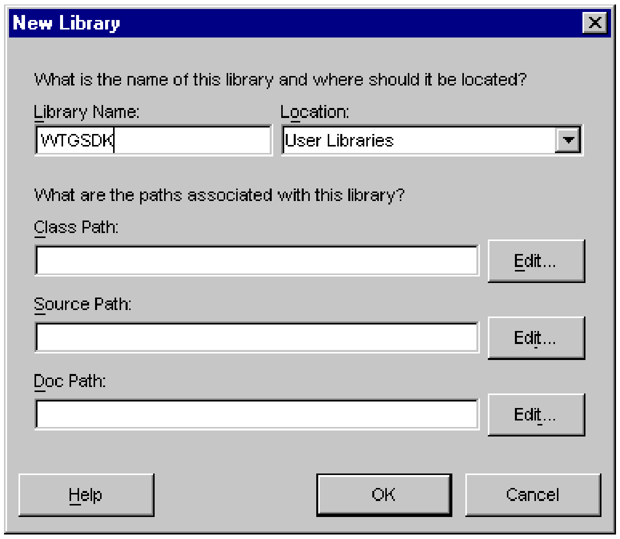
`<`*`ORACLE_HOME`*`>\mobile\classes\wtgpack.jar`

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 5–4. In the **Library Name** field, enter WTGSDK.

Figure 5–4 The New Library Dialog



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**.

5.5.2.9.3 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**.

5.5.2.9.4 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.

5.5.2.9.5 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.

5.5.2.10 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 5.5.2.10.1, "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 5.5.2.10.1, "Web-to-Go Parameters".

5.5.2.10.1 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 5–4 describes `webtogo.ora` parameter settings.

Table 5–4 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.

5.5.2.10.2 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 5–5 describes scripts available for JSP compilation.

Table 5–5 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.

5.5.2.11 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*.

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 MDW. For more information, see Chapter 6, "Using Mobile Database Workbench to Create Publications".

6 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.

6.1 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 6.1.1, "Set Access Privileges to SYSTEM Tables for Your Application Schema"
- Section 6.1.2, "Launch MDW"

6.1.1 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.

6.1.2 Launch MDW

To launch MDW, execute `oramdw`, which is located in *`$ORACLE_HOME`*`\Mobile\Sdk\bin`.

6.2 Create a Project

Create a new project with MDW. 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**.

- Mobile client database type: Select the type of the Mobile client, which can be either the SQLite Mobile client or Oracle Lite Mobile client.

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:

Username and Password

Specify the Mobile Server repository username and password (with administrator privilege). This is the same username 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, ANALYZE ANY,
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, DROP ANY
SYNONYM, DROP ANY TRIGGER, DROP ANY INDEX, DROP ANY TABLE,
INSERT ANY TABLE, SELECT ANY TABLE, SELECT ANY DICTIONARY,
SELECT_CATALOG_ROLE, UPDATE ANY TABLE
```

Specify the Connection

When you define the connection, you have two options:

- Simple connection definition. For a connection to a single back-end Oracle database, select the **Oracle JDBC Thin driver** and provide the host (or IP address), port and SID for the Oracle database that contains the Mobile repository.

  This creates a connect string which will be as follows:

  ```
  jdbc:oracle:thin:@<host>:<port>:<SID>
  ```

- RAC connection definition. If you are connecting to a RAC configuration, then select the **Oracle JDBC Thin driver—Advanced**. Selecting this option enables the Connect String field where you can enter the tnsnames connect string that defines all databases included in the RAC configuration. For example, the following is a tnsnames connect string definition that includes two Oracle databases in the RAC configuration:

  ```
  (DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=-ON)
   (ADDRESS=(PROTOCOL=TCP)(HOST=HOST1)(PORT=1555))
   (ADDRESS=(PROTOCOL=TCP)(HOST=HOST2)(PORT=1555)))
   (CONNECT_DATA=(SERVICE_NAME=ORCL.TW.ORACLE.COM)
   (FAILOVER_MODE=(TYPE=SELECT)(METHOD=BASIC)(RETRIES=180)(DELAY=5))))
  ```

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 the application information, as follows:

   a. Specify the application schema username and password, each of which are limited to 28 characters. 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.

   b. Use the Database Instance pull-down to select the database where the application is to be deployed. In this database, the application schema will be created. You can select the Main database, where the Mobile repository is stored, or any registered remote databases meant solely for application data.

      Note: For more information on using and registering remote databases for application data, see Section 3.2, "Register a Remote Oracle Database for Application Data".

      Once selected, the JDBC URL for the selected database is displayed in the Connect String field.

   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.

6.3 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 6.4, "Create a Publication Item".

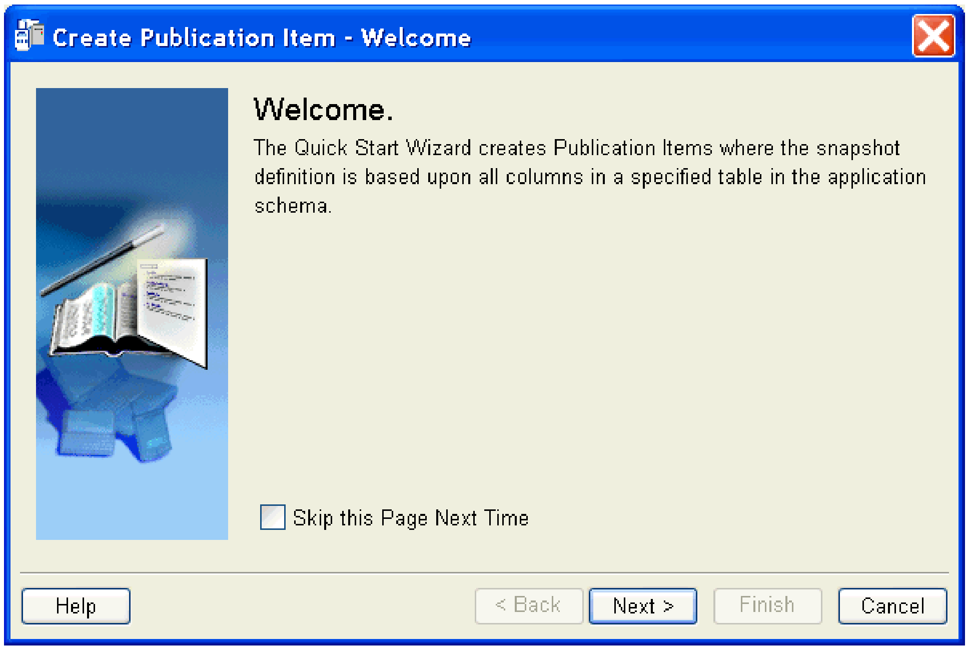
The publication item name defaults to the following: `<table_name>_PI<number>` where `<number>` is sequential between 1 and 9. For example, the first publication item created on table `EMP` would be named `EMP_PI1`. If, in a separate publication, you have already defined a publication item for `EMP_PI1`, 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 6–1 Welcome Screen for Quick Wizard to Create a Publication



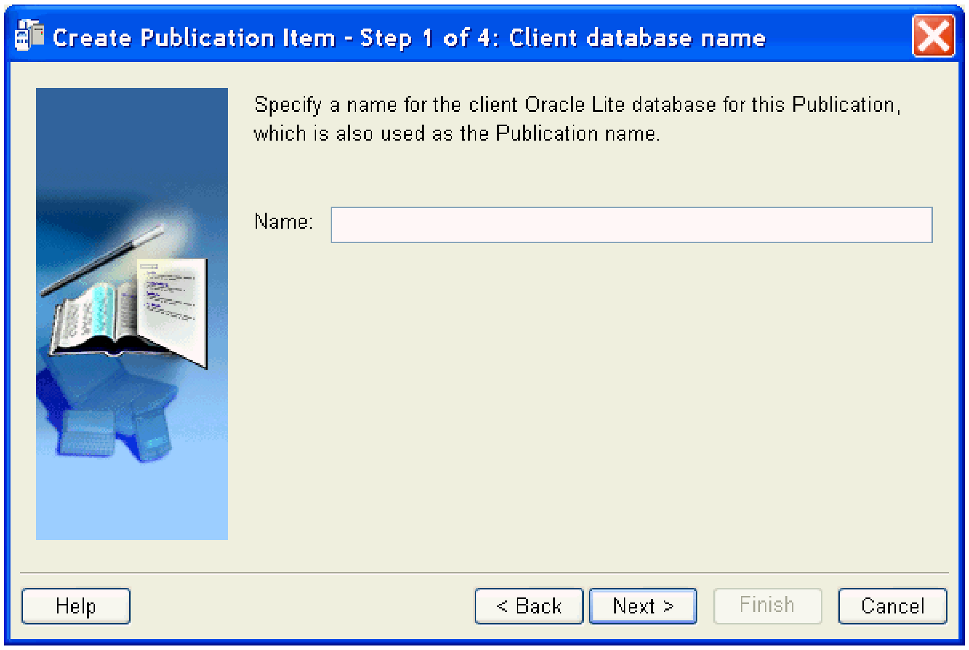
Welcome.

The Quick Start Wizard creates Publication Items where the snapshot definition is based upon all columns in a specified table in the application schema.

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 6–2 Define Client Oracle Lite Database Name



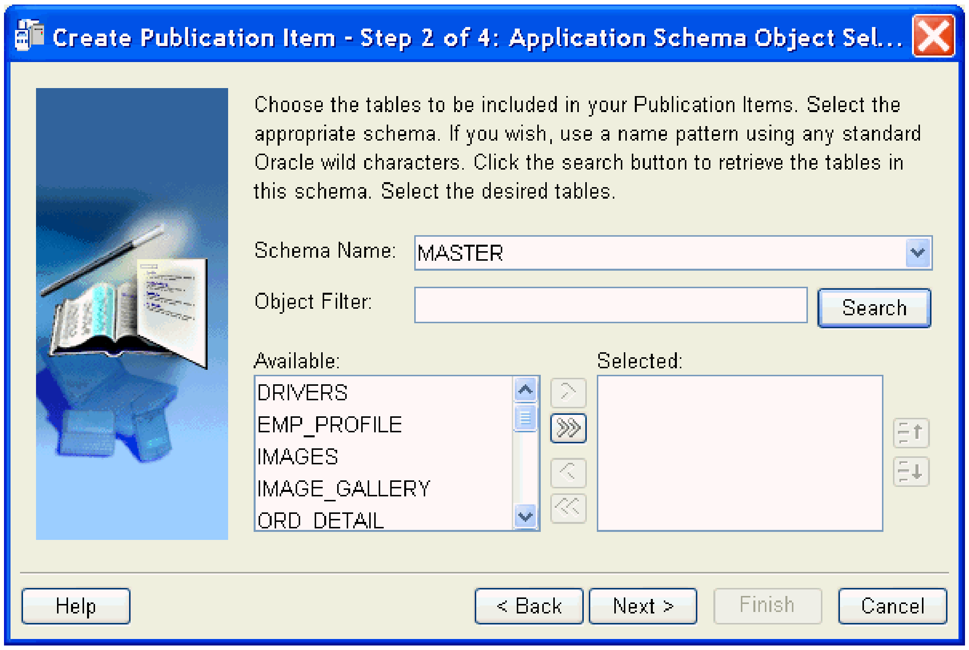
Create Publication Item - Step 1 of 4: Client database name

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 6–3 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.

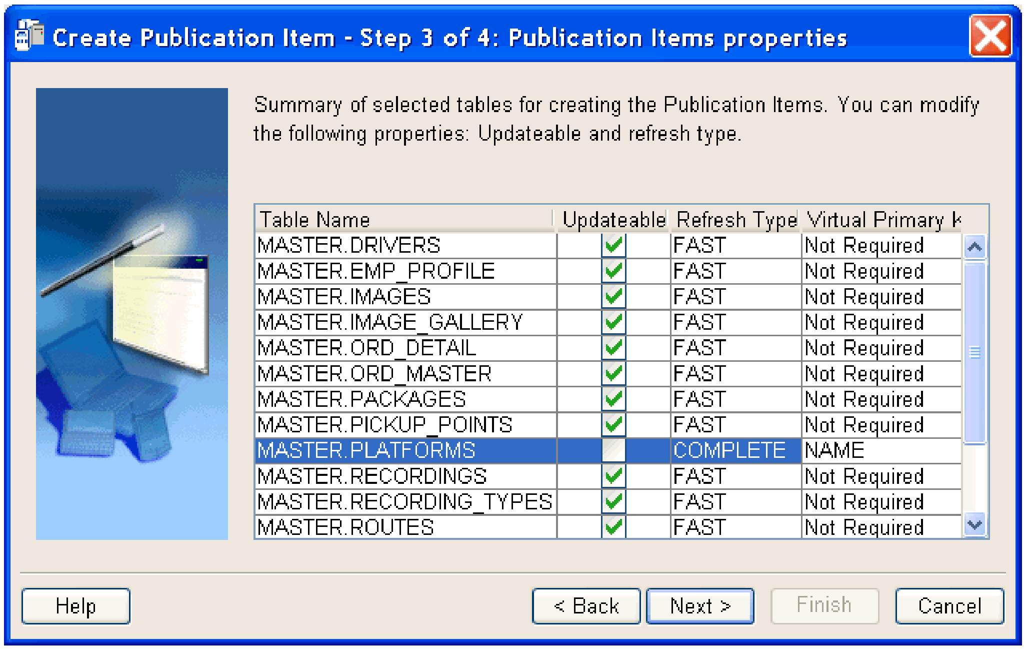
- Click **Search** to display all tables within this schema in the Available column. To search for a specific table or tables, enter the name or partial name with wild charaters in the Object Filter field and then click **Search**. You can use any of the standard Oracle wild card characters.
- Select 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 6–4 Modify the Table Properties for Synchronization


Create Publication Item - Step 3 of 4: Publication Items properties
Help
< Back
Next >
Finish
Cancel

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 2.3.1.1, "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 2.9, "Understanding Your Refresh Options".

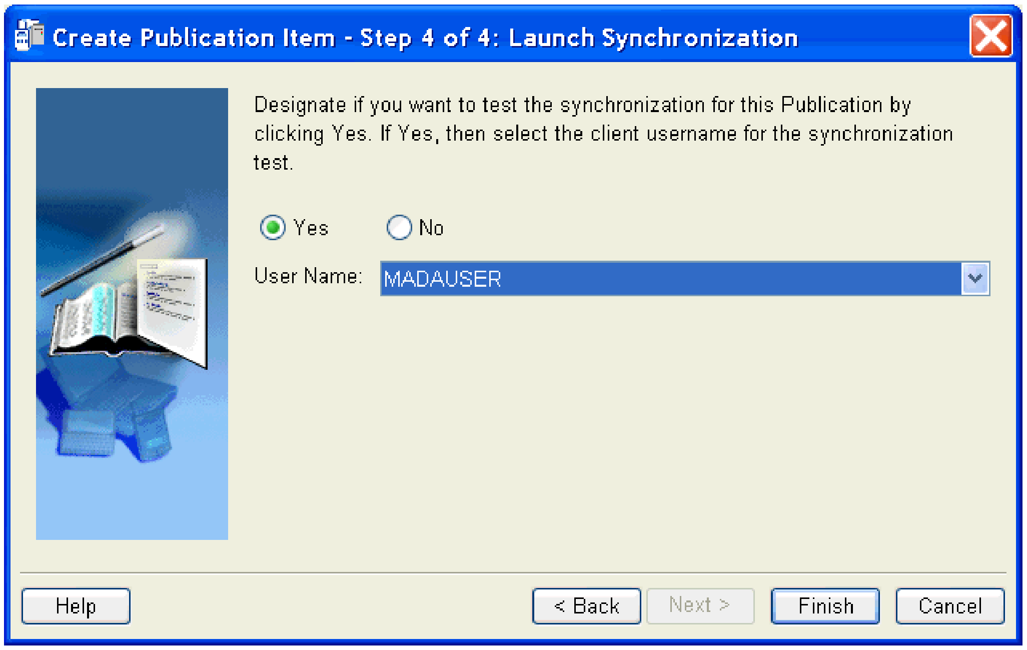
- **Virtual Primary Key**: This field displays the virtual primary key for the table. If you want to have the table be updatable or use the fast refresh type, then the table must have a virtual primary key. If the table does not have a primary key, but it does contain a field with UNIQUE constraints, 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 6–5 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

Help
< Back
Next >
Finish
Cancel

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.

6.4 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 6.9.2, "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 2.9, "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 2.13, "Customizing Synchronization 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. However, if you have multiple publication items where these contain tables with a master-detail relationship, then ALL of these publications items must be either enabled or disabled for automatic synchronization.

  Note: If, after you have published the application, you want to turn off automatic synchronization, you can only disable automatic synchronization as follows:

  1. Execute the API to disable the automatic synchronization. Use the `reCreatePublicationItem` method to disable automatic synchronization. For more information on modifying a publication item using the APIs, see Section 2.4.1.12, "Modifying a Publication Item".
  2. Execute a manual synchronization on the device.The manual synchronization restarts the automatic Sync Agent, which will then use the new rules The new settings will NOT be downloaded automatically during automatic synchronization.

  Step 7 shows you how to specify—with a SQL statement—which users receive the automatic synchronization. Section 6.5, "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 3.8, "Datatype Conversion Between the Oracle Server and the 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.

     The publication item query must select primary keys in the same order as they are defined in the base table.

   - 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. Create indexes and associate them with the publication item. These indexes will display in the Index Selection window. The index name, index type and the index columns are shown. If you want to add or remove an index in the publication item, use the following options:

Note: You can only modify an index by removing and creating it again.

- Add: Click **Add** to create a new index and associate it with the publication item. You will need to provide the index name, type and snapshot columns for the new index.
- Remove: Removes an existing index from the publication item by selecting the index in the Index Selection window and clicking **Remove**.

By default, Oracle Database Lite creates a primary key index for every publication item and is created on the primary key of the snapshot table. This index is named `<piName>_PK`. You cannot remove the primary key index from the publication item within this screen. If you want to remove the primary key index, use the `dropPublicationItemIndex` method. For more details, see Section 2.4.1.6, "Create Publication Item Indexes".

If a snapshot is based on a table without a primary key, then the primary key index will not be created and the Index Selection box is empty.

When the desired indexes have been added or removed for this publication item, click **Next** to advance to the next window.

6. 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 6.4.1, "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 customized 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 2.7.2, "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 6.4.2, "Create a Dependency Hint" for more information.
7. If you specified a view, then you may need to define parent table and primary key hints. See Section 6.4.3, "Specify Parent Table and Primary Key Hints" for directions on how to define these hints.
8. 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.

9. 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.

6.4.1 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**.

6.4.2 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 3.3.2, "Creating or Removing a Dependency Hint".

6.4.3 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.

6.5 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 2.2.3, "Define the Rules Under Which the Automatic Synchronization Starts" for directions on how to perform this programmatically.

If after defining these rules and publishing the application, you want to modify the rules, you can do so through MDW. However, you must perform a manual synchronization. The manual synchronization restarts the automatic Sync Agent, which will then use the new rules The new settings will NOT be downloaded automatically during automatic synchronization.

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

- Section 6.5.1, "Configure Publication-Level Automatic Synchronization Rules"
- Section 6.5.2, "Configure Platform-Level Automatic Synchronization Rules"

6.5.1 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 6.9.6, "Event Tab Configures Automatic Synchronization Rules for this Publication".

Table 6–1 describes the publication level data events that trigger automatic synchronization. The lowest value you can specify is 1.

Table 6–1 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.
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**.

6.5.2 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 6.4.1, "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 6.5.2.1, "Define System Event Rules for the Platform".
   - Conditions—See Section 6.5.2.2, "Define Automatic Synchronization Conditions for the Platform".

Note: You can only modify the network settings for the platform using Mobile Manager, see Section 6.4.1, "Specifying Platform Rules for Automatic Synchronization" in the *Oracle Database Lite Administration and Deployment Guide* for more information.

6.5.2.1 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.

6.5.2.2 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.

    Note: You can only use fast refresh with a high priority restricting predicate. If you use any other type of refresh, the high priority restricting predicate is ignored.

    See Section 1.2.10, "Priority-Based Replication" in the *Oracle Database Lite Troubleshooting and Tuning Guide* for more information.

  - 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.

6.6 Create a Sequence

Note: There is no support for sequences on a SQLite Mobile client.

A sequence is a database object, from which you can generate unique integers. You can use sequences to automatically generate primary key values. However, when you have multiple clients accessing a single server, you need a method to guarantee unique identifying numbers for new records from multiple clients. Oracle Database Lite provides a method for unique sequence 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.

For Oracle Database Lite, you can add a sequence to a publication; then, the sequence is created on all subscribed clients during the initial synchronization. On each client Oracle Lite database, the sequence can be used independently. However, since the sequences are used to generate primary key and unique key values for snapshot tables, it is important to ensure that different clients do not generate the same sequence values. If they did, then conflicts may occur when the clients synchronize their changes to the server tables.

The Sync Server guarantees uniqueness across all clients. During synchronization, the Sync Server assigns separate sequence ranges, known as a window, to each client when necessary. A client cannot increment a sequence beyond its current window. Once a client exhausts its window, the Sync Server assigns a new window on the next synchronization. All windows are unique and never reassigned.

Since the sequence windows are obtained from the Sync Server only during synchronization, there is a chance that the client could exhaust all available sequence numbers in its window in between synchronization events. To prevent this from happening, the administrator can configure clients to obtain a new window before the current one is exhausted by setting the threshold value. A threshold is less than the window size. If the range of values left in the window is less than the threshold size, then during the next synchronization, a new window will be assigned to the client. For example, you set the window to 200 and the threshold to 25. During a synchronization event, the Sync Server notices that the current sequence value is greater than or equal to 175, then it allocates the next window for the client.

The following describes how to use Oracle Database Lite sequences:

- Only clients use the sequence: If you have more than a single client, you want each client to use a specific range of unique sequence numbers, so that none of the records have duplicate sequence numbers.

  Specify the start value and the window size. When you define the size of the window, you provide the Sync Server the number of assigned identifiers for each client and the range of values never overlaps with those of other clients.

- Server and clients use the same sequence: If you want the server and one or more clients to share a sequence, then the server and the client use every other number—the identifiers are generated where the server uses all even numbers and the clients use odd numbers—or vice versa, if the start value of the sequence is an even number. Specify the start value, window size and select the "Generate Server-side Sequence" option that tells Oracle Database Lite to generate a server-side sequence. The increment value always defaults to 2 for this case, even if you specify another number. If you have more than one client, configure the window size to ensure that the client sequence numbers do not conflict.

6.6.1 Configuring Sequences in MDW

Within Oracle Database Lite, you configure how you want sequence numbers generated in the sequence definition. In MDW, create a sequence by clicking **File->New->Sequence**. When you are creating your publication, configure the following values to instruct how the sequence is generated for all clients and the server:

- Name: This sequence name must be a valid database 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: Provide the size of the window that the Sync Server assigns to the client. For each client, the Sync Server assigns the initial range of sequence values at the time of subscription. For example, if you set the window size to 100 and the Starts With value to 1, then the Sync Server assigns the client windows as follows:
  - Client A: sequence numbers 1 through 100
  - Client B: sequence numbers 101 through 200
  - Client C: sequence numbers 201 through 300

  If any client exhausts their window, they are assigned another 100, which is the defined window size in our example, during the next synchronization. If you also click the **Generate Server-Side Sequence** checkbox, then the sequence numbers used by the clients are the odd numbers in their range, such as 1, 3, 5, 7 and so on.
- Threshold: When the number of identifiers left in the window is less than the threshold, a new window of sequence numbers is assigned to the client on the next synchronization. For example, if you have a window size of 200 and threshold of 25, then when the current sequence number is equal to or greater than 175, the Sync Server assigns the next window of values to the client.

  Note: If the window size is 100 and the threshold is 25, then no matter what the increment is, the next window is assigned when the sequence numbers are equal to or greater than 75. It is based on the window size, not on the number of sequence values left for the client.

- 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.

When you check the Generate Server-Side Sequence checkbox, then no matter what value is specified for Increment, it will always be set to 2. A sequence of the supplied name is created automatically on the server in the `mobileadmin` schema to use all even numbers. Specify a 1 as the Starting Value, so that on the server side, the sequence uses even values starting with 2 and on the clients, the odd values are used. Thus, the server and client sequence values are unique.

If there are multiple clients, then to ensure that the clients use unique numbers, set up separate windows for each client. There is no window for the server, because the server uses all even numbers in the whole range of the sequence.

For example, the sequence number for the first client starts at 1 and increments by 2 for all of its sequence numbers. The first client still has a window size, which in this example is 100, but it starts with an odd number within that window and always increments by 2 to avoid any even numbers. Thus, client A has the window of 1 to 100, but the sequence numbers would be 1, 3, 5, and so on up to 99.

Oracle Database Lite creates and maintains the sequence based on the sequence definition in the publication. Once you create a sequence in the project, you can associate it with a publication. See Section 6.9.3, "Sequence Tab Associates Existing Sequences With the Publication" for details.

See the Section 2.4.1.8, "Creating Client-Side Sequences for the Downloaded Snapshot" for more information on sequences.

6.6.2 Configuration Scenarios for Sequence Generation

When setting up a sequence, you can configure one of the following three scenarios:

- Multiple Clients: In this case, always define Start Value and the Window Size parameters. When you define the size of the window, you provide the Sync Server the number of assigned identifiers for each client and the range of values never overlaps with those of other clients. Also, set the starting value for each client.
- Server and Clients Use Same Sequence: If you want the server and one or more clients to share a sequence, select the "Generate Server-Side Sequence" checkbox. If you have multiple clients, set the Start Value and the Window Size. The checkbox tells Oracle Database Lite to create a sequence on the server side, where the clients use the Start Value and the server uses Start Value +1. The identifiers are generated where the server or the client uses either all odd numbers or even numbers. The window ensures that the clients do not use the same sequence window.

6.6.3 Example of a Sequence

For this example, the sequence is defined as follows:

Table 6–2

Parameter	**Definition**
Name	`audiodb_seq`
Start Value	1
Increment	1
Window size	200
Threshold	25

The first client starts at 1 with an increment of 1. The full range of sequential values provided to client 1 is 200 and a new set of sequential numbers is assigned during synchronization by the Sync Server when the unused portion of the window is less than 25.

Oracle Database Lite creates the sequence locally on the Mobile client by executing the following SQL statement:

```
create sequence audiodb_seq start with 1 maxvalue 200 increment by 1;
```

On the second client, the Sync Server adjusts the numbers appropriately to accommodate what was created on client 1 and creates the sequence locally with the following SQL statement:

```
create sequence audiodb_seq start with 201 maxvalue 400 increment by 1;
```

During each synchronization, the Sync Server tracks the number of assigned windows to ensure that each client has a unique range. When the Sync Server assigns a new set of sequential numbers as identifiers for the client, it recreates the sequence, as follows:

```
drop sequence audiodb_seq;
create sequence audiodb_seq start with 401 maxvalue 600 increment by 1;
```

6.6.4 Example of a Client and Server Sharing a Sequence

You can define a sequence to provide unique sequence values by assigning all odd or even sequence numbers to either the client or the server. The value specified in the Start Value sets the starting value for the clients. If the server is sharing a sequence with a client, then the start value also determines the values for the server. If the starting value is odd, then the server will use all even numbers; if the starting value is even, then the server uses all odd numbers.

The following example demonstrates how to set up a sequence where the odd numbers are for the client and the even numbers for the server.

Enter the following sequence definitions for the client in MDW when defining the publication:

Table 6–3

Parameter	Definition
Name	`audiodb_seq`
Start Value	1
Increment	2
Window size	200
Threshold	25
Generate server-side sequence	Check on

The sequence on the server starts at 2 and uses all even numbers; within the publication, you specified that all clients use odd numbers starting at 1.

6.7 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 6.7.1, "Writing SQL Scripts"
- Section 6.7.2, "Test SQL Scripts"
- Section 6.7.3, "Load the Script Into the Project"

6.7.1 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 6.7.3, "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.

6.7.2 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 6.7.2.1, "Connect to the Database"
- Section 6.7.2.2, "Load and Execute SQL Scripts"

6.7.2.1 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**.

6.7.2.2 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**.

6.7.3 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 6.9.4, "Script Tab Associates Existing Scripts With the Publication" for more information.

6.8 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**.

Note: You cannot add resources for a SQLite Mobile client.

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 6.9.5, "Resource Tab Associates Existing Resources With the Publication" for more information.

6.9 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 6.9.1, "General Tab Configures Publication Name"
- Section 6.9.2, "Publication Item Tab Associates Publication Items With the Publication"
- Section 6.9.3, "Sequence Tab Associates Existing Sequences With the Publication"
- Section 6.9.4, "Script Tab Associates Existing Scripts With the Publication"
- Section 6.9.5, "Resource Tab Associates Existing Resources With the Publication"
- Section 6.9.6, "Event Tab Configures Automatic Synchronization Rules for this Publication"

6.9.1 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.

6.9.2 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**.

6.9.2.1 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 2.3.1.1, "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 2.11, "Resolving Conflicts 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 2.4.1.13, "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 1.2.7, "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.

See Section 1.2.10, "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 2.6, "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.

6.9.3 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.

6.9.4 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 6.7.1, "Writing SQL Scripts".

6.9.5 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.

6.9.6 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.

6.10 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 6.10.1, "Import Existing Publication from Repository"
- Section 6.10.2, "Import Existing Publication Item From the Repository"
- Section 6.10.3, "Import Existing Sequence From the Repository"

- Section 6.10.4, "Import Existing Resource From the Repository"
- Section 6.10.5, "Import an Existing Script From the Repository"

6.10.1 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.

6.10.2 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 6.9.2, "Publication Item Tab Associates Publication Items With the Publication" for more information.

6.10.3 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 6.9.3, "Sequence Tab Associates Existing Sequences With the Publication" for more information.

6.10.4 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**.

Note: You cannot add resources for a SQLite Mobile client.

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 6.9.5, "Resource Tab Associates Existing Resources With the Publication" for more information.

6.10.5 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 6.7.1, "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 6.9.4, "Script Tab Associates Existing Scripts With the Publication" for more information.

6.11 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.

6.12 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.

   Note: You can only use fast refresh with a high priority restricting predicate. If you use any other type of refresh, the high priority restricting predicate is ignored.

   See Section 1.2.10, "Priority-Based Replication" in the *Oracle Database Lite Troubleshooting and Tuning Guide* for more information.

   - 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 4.2.8, "Manage What Tables Are Synchronized With Selective Sync".

   Click **OK** to save the synchronization options or **Cancel** to return to the previous screen.

6.13 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.

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 6, "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 one of the following methods:

- SQL on the back-end Oracle database
- Consolidator APIs
- MDW
- Packaging Wizard

Note: If you create your publication using the Packaging Wizard, you cannot use remote databases for your application.

- 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 6, "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.

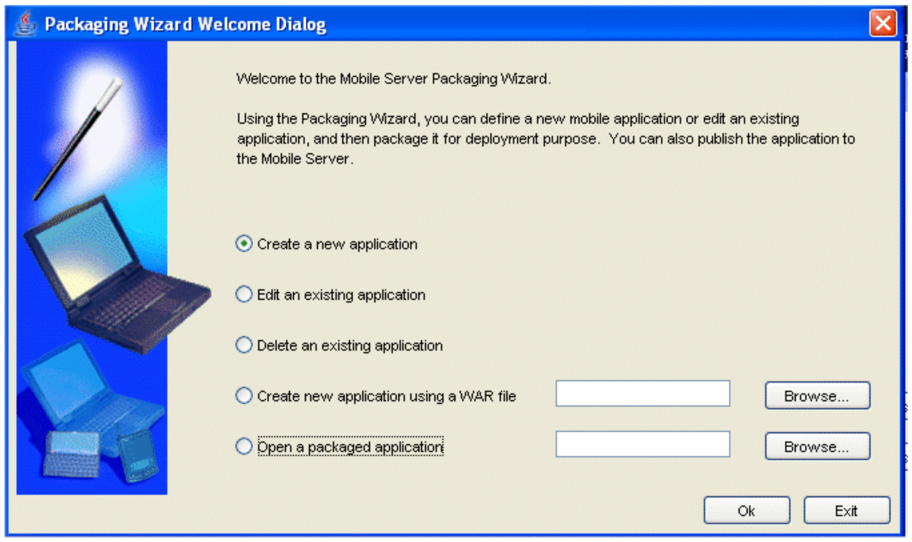
```
runwtgpack
```

Note: If you enable the Mobile Server to be SSL-Enabled, then you have to change the configuration on the host where the Packaging Wizard is located in order for it to successfully communicate with the Mobile Server.

In order for Packaging Wizard to be SSL-Enabled, set the `SSL` parameter to `TRUE` in the `webtogo.ora` file located on the host where the MDK is installed.

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



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.

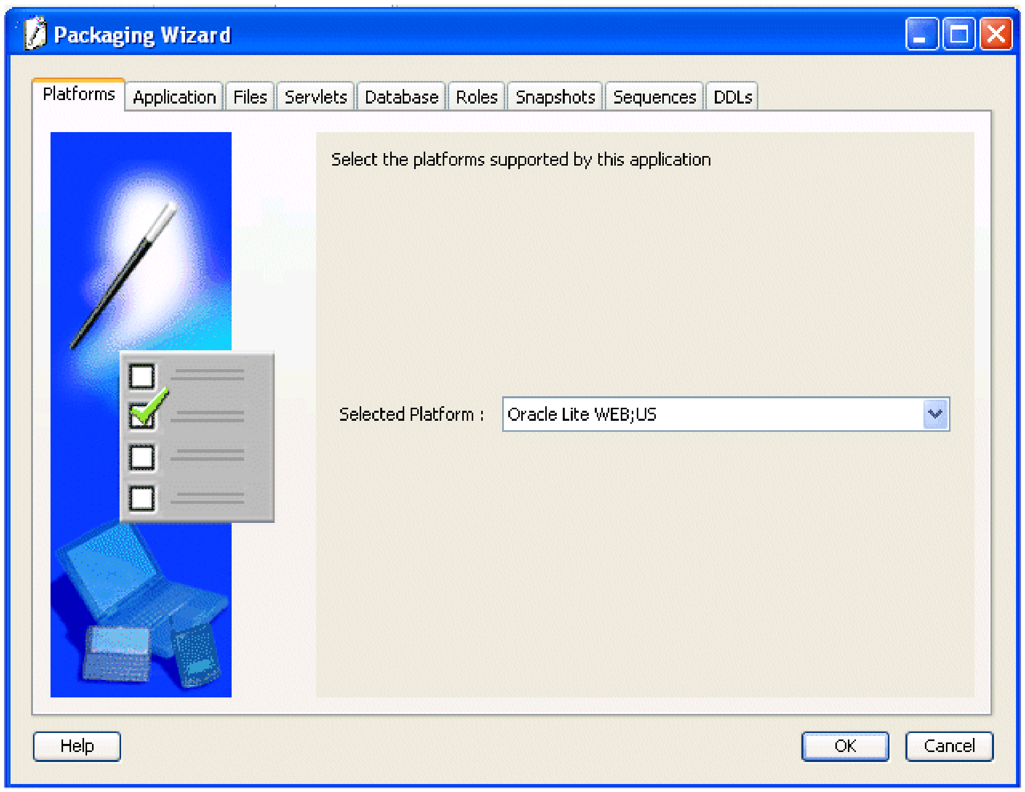
[✓ Create a new application]
Edit an existing application
Delete an existing application
Create new application using a WAR file
Open a packaged application

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



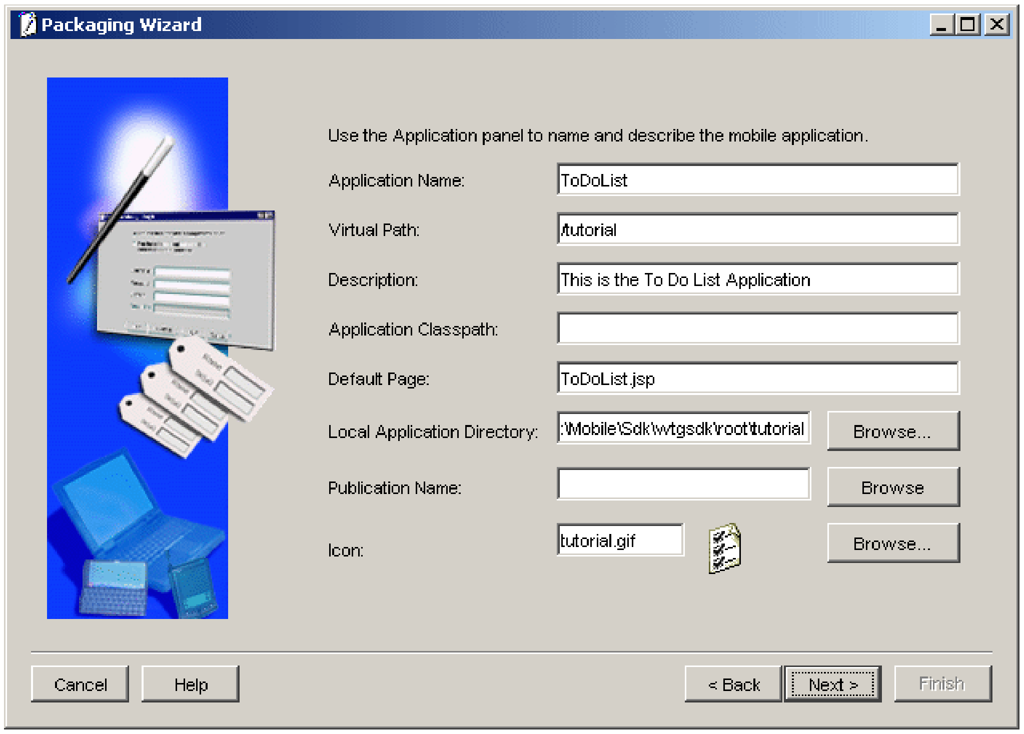
Select the platforms supported by this application

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



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

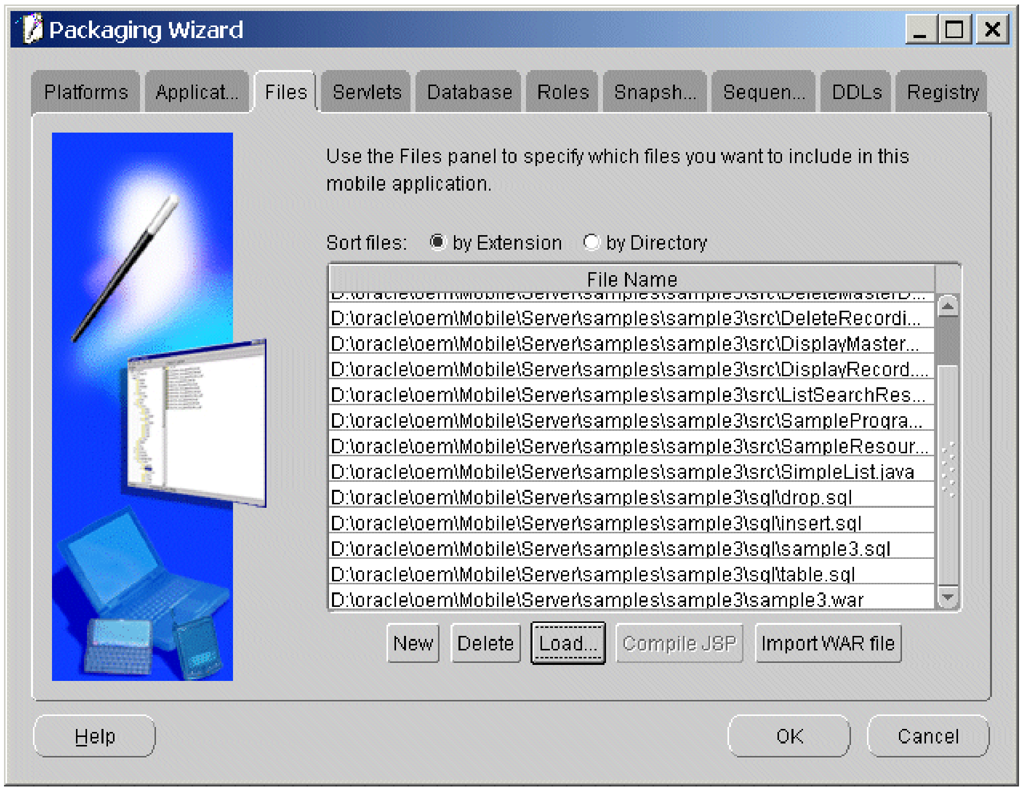
7.1.3 Listing Application Files

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\DeleteMasterD...
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.

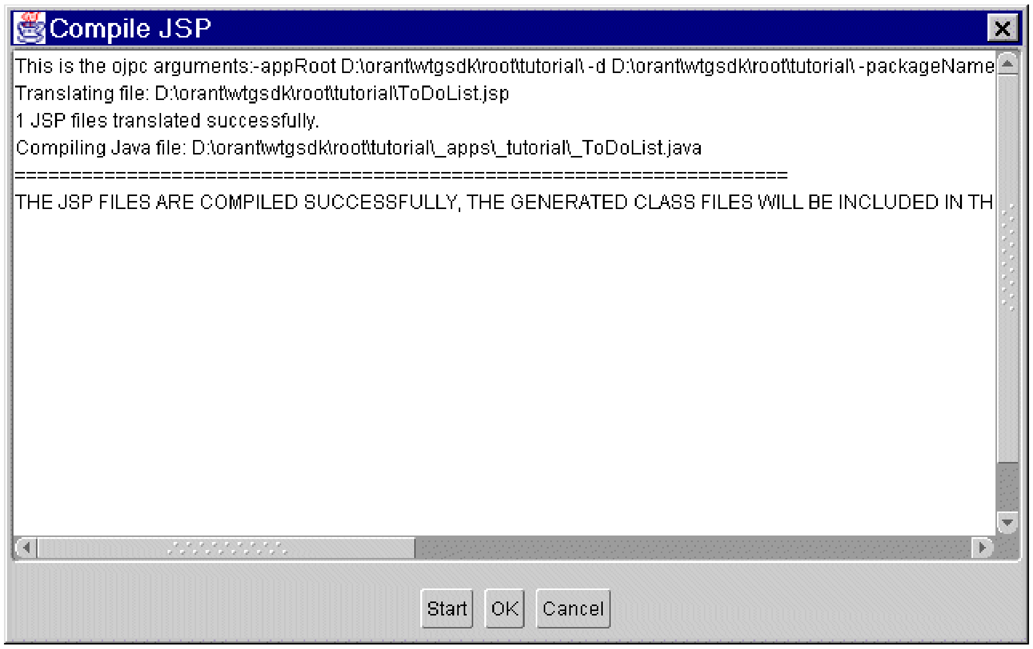
If you are importing a WAR file into an existing application, click the **Import WAR File** button on the 'Files' tab. Once you have specified the location of the WAR file, the 'Files' tab displays content of the WAR file.

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.

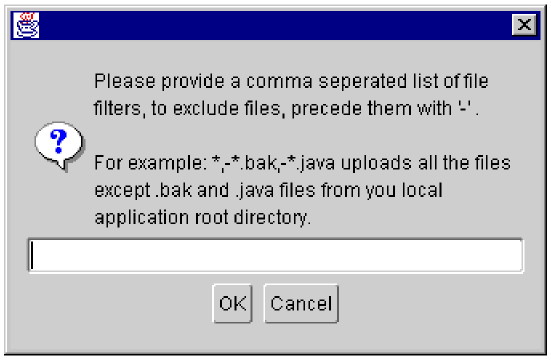
7.1.3.2 Filters

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



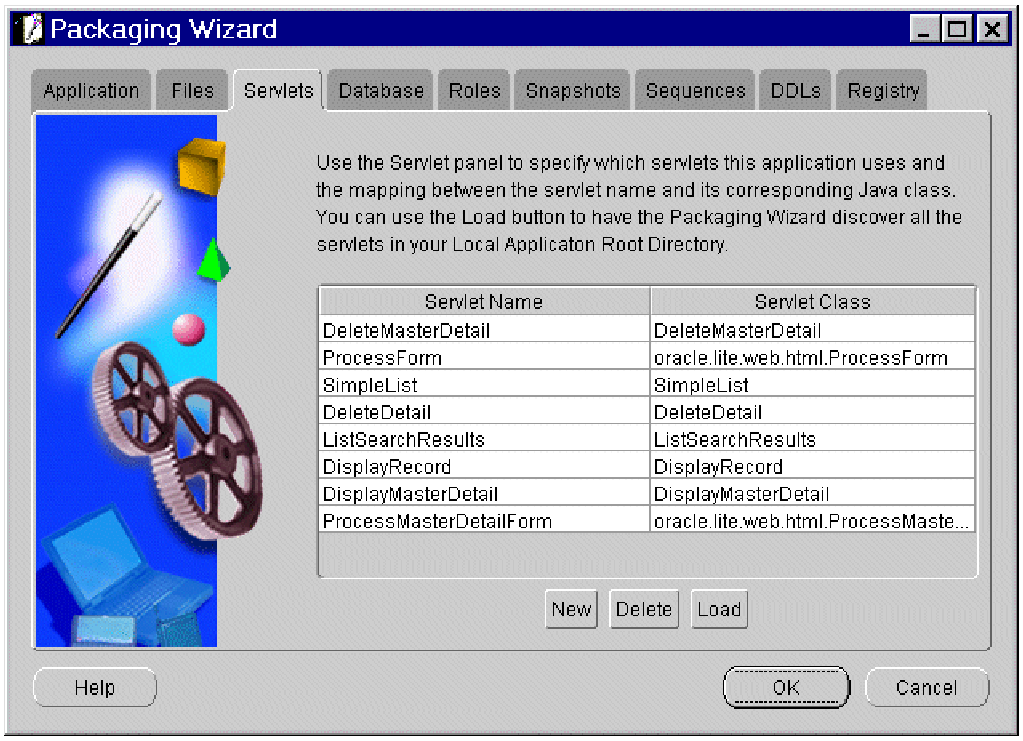
Please provide a comma seperated list of file filters, to exclude files, precede them with '-' .

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

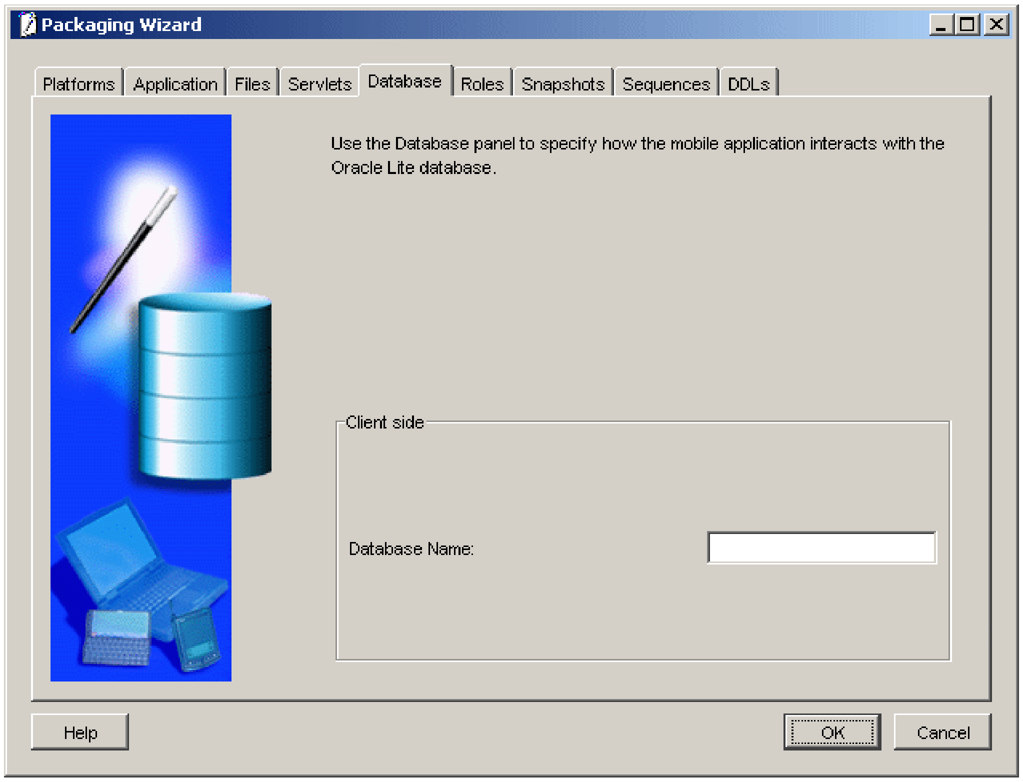
Using the 'Servlets' tab, you can add, remove, or load any servlets that are listed under the 'Servlets' tab. If you are creating a new application, the Packaging Wizard automatically lists all 'Servlets' based on files that are listed in the 'Files' tab. If you are editing an existing application, use 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:

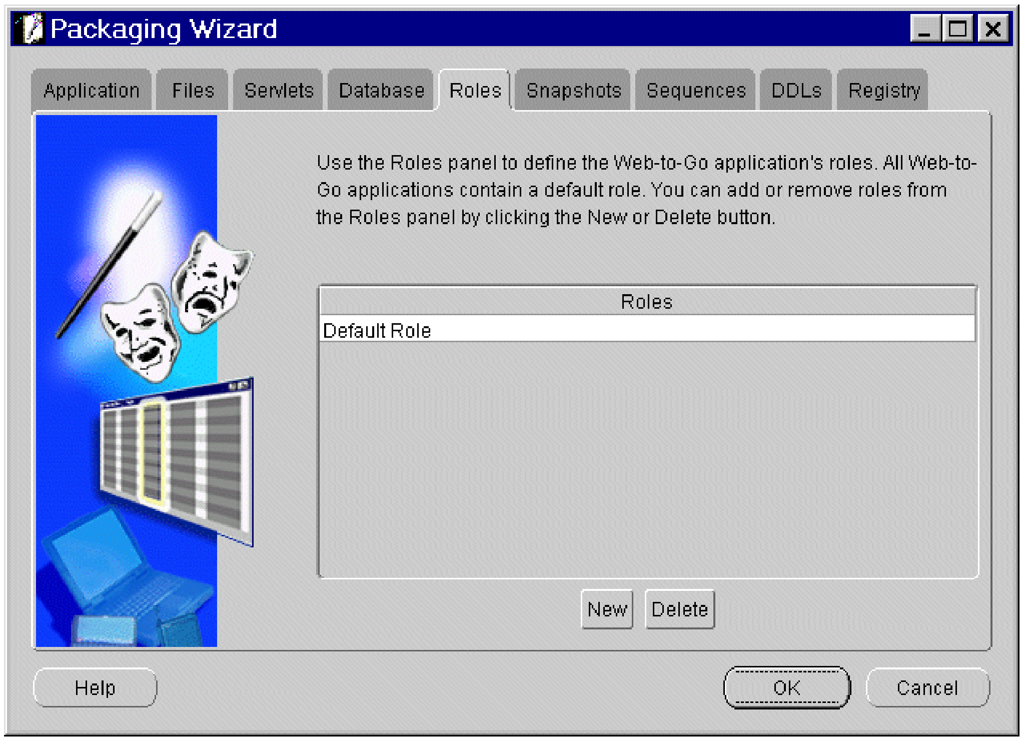
Enter the database name that you want to create on the client side. For example, a native Windows 32 application accesses the client database with this name. However, this is not required for Web applications.

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.

Roles
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.

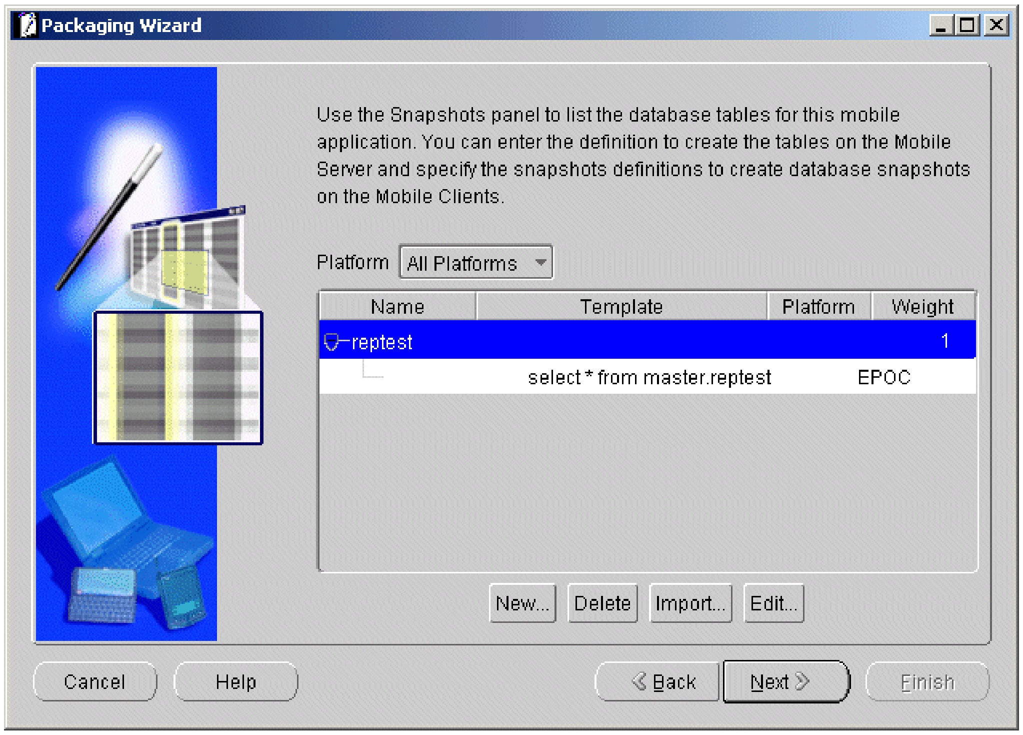
7.1.7 Defining Snapshots for Replication

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.

Note: You cannot create snapshots for a SQLite Mobile client.

Figure 7–10 displays the Snapshots tab.

Figure 7–10 Snapshots Tab



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

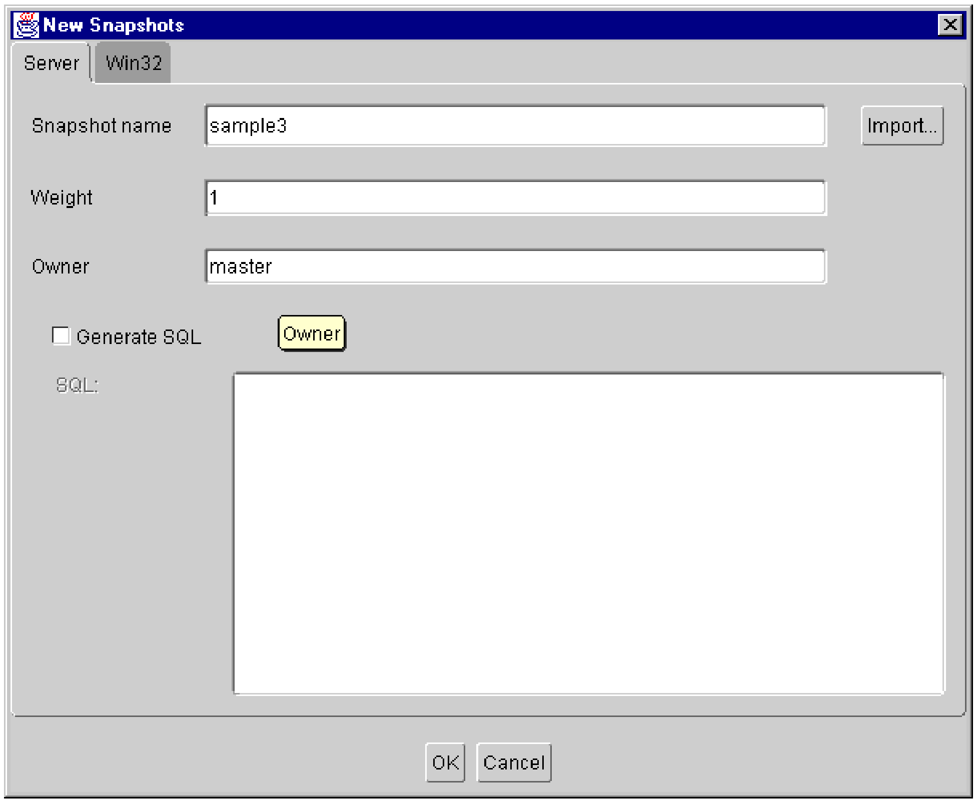
You can add or remove snapshots from the Snapshots tab using the 'New' or 'Delete' button. You can also import or edit snapshots using the 'Import' or 'Edit' button.

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



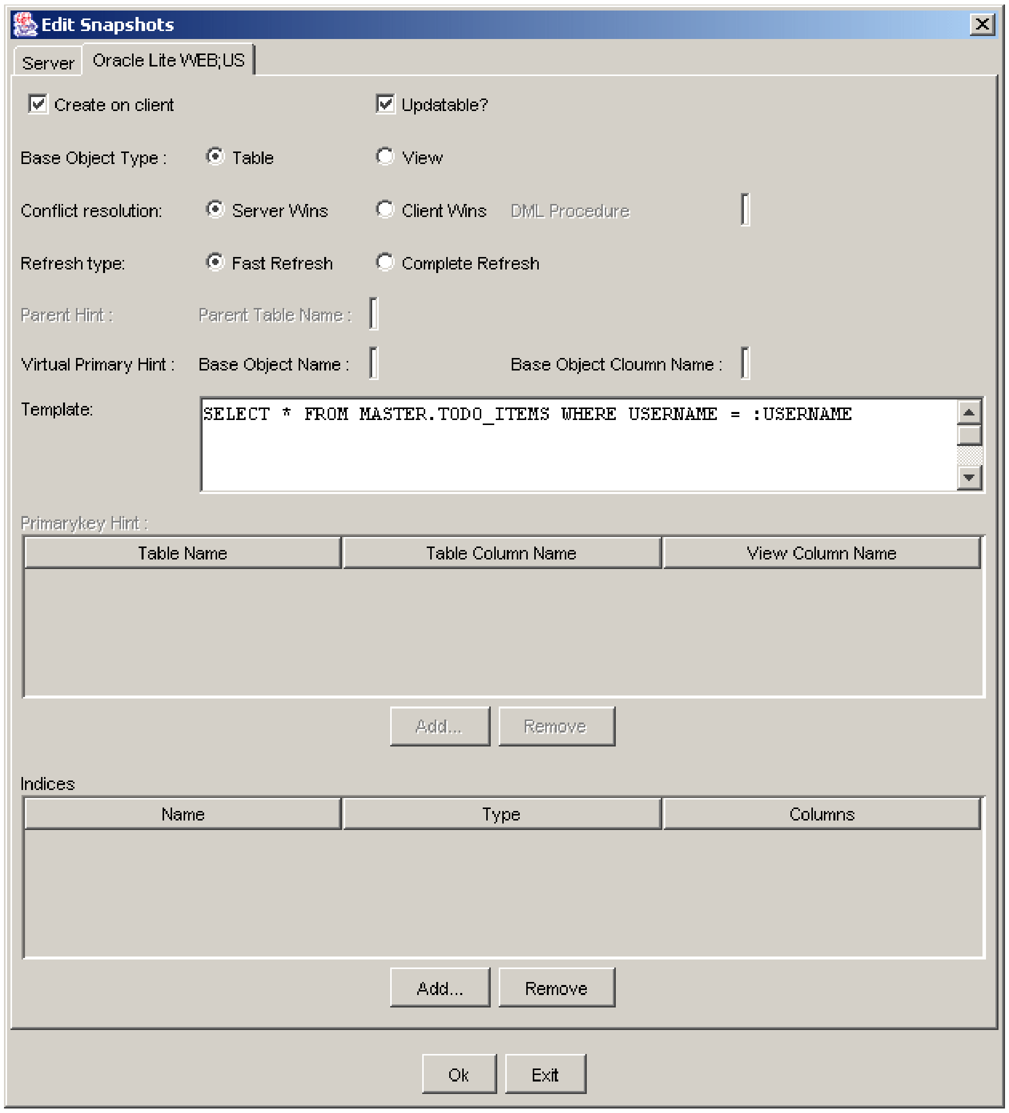
Snapshot name: sample3
Weight: 1
Owner: master
☐ Generate SQL

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



[✓] 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

Indices

Name	Type	Columns

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

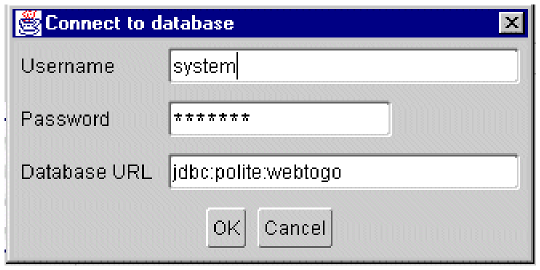
Field	Description
Create on Client	If selected, creates the snapshot on the client machine.
Updatable	If selected, creates an updatable snapshot of the specified table or view.
Base Object Type	Select **Table** to include a table as the base object type. or Select **View** to include a view as the base 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).

Where:

- Username and password: The Mobile Server repository administrator username and password.
- Database URL: You can specify the JDBC URL of an Oracle Lite database, an Oracle database or an Oracle RAC database, as follows:
  - If on a client, specify the Oracle Lite database with `jdbc:polite:webtogo`.
  - The URL for a back-end Oracle database has the following structures: `jdbc:oracle:thin:@<host>:<port>:<SID>` or `jdbc:oracle:thin:@<oracle_net_or_tnsnames_entry>`
  - The JDBC URL for an Oracle RAC database can have more than one address in it for multiple Oracle databases in the cluster and follows this URL structure:

```
jdbc:oracle:thin:@(DESCRIPTION=
 (ADDRESS_LIST=
   (ADDRESS=(PROTOCOL=TCP)(HOST=PRIMARY_NODE_HOSTNAME)(PORT=1521))
   (ADDRESS=(PROTOCOL=TCP)(HOST=SECONDARY_NODE_HOSTNAME)(PORT=1521))
 )
 (CONNECT_DATA=(SERVICE_NAME=DATABASE_SERVICENAME
)))
```

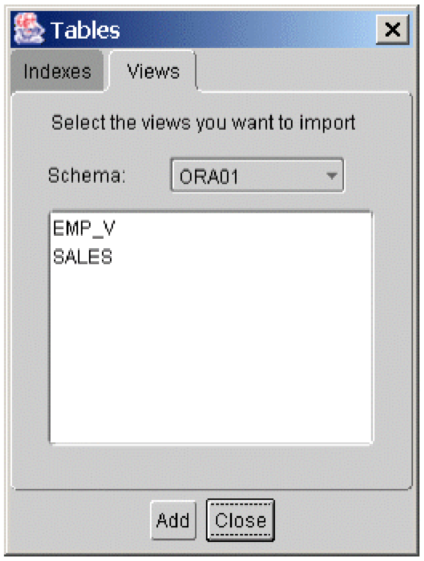
Note: For full details on specifying JDBC URLs, see Chapter 2, "Connecting to the Oracle Lite and Oracle Databases" in the *Oracle Database Lite Administration and Deployment Guide*.

The Tables window appears.

Note: See Chapter 2, "Connecting to the Oracle Lite and Oracle Databases" in the *Oracle Database Lite Administration and Deployment Guide* for directions on how to construct the JDBC URL for either the Oracle database or the client Oracle Lite database.

Figure 7–14 displays the Tables dialog.

Figure 7–14 Tables Dialog

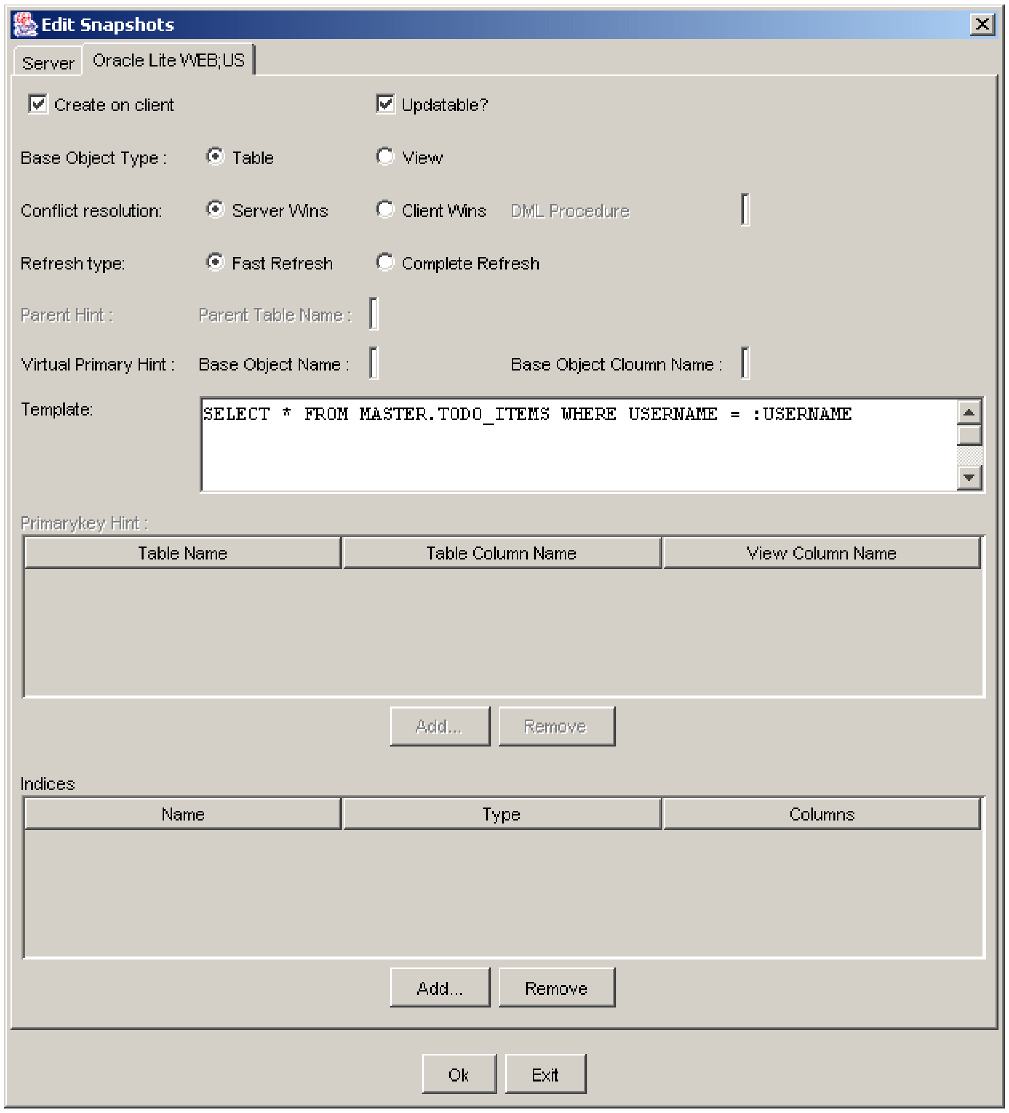


Click the Schema list and choose the required schema from the list displayed. The Tables dialog displays views associated with the chosen schema. Select the view that you need to import. Click Add and click Close.

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



[✓] 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

Indices

Name	Type	Columns

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

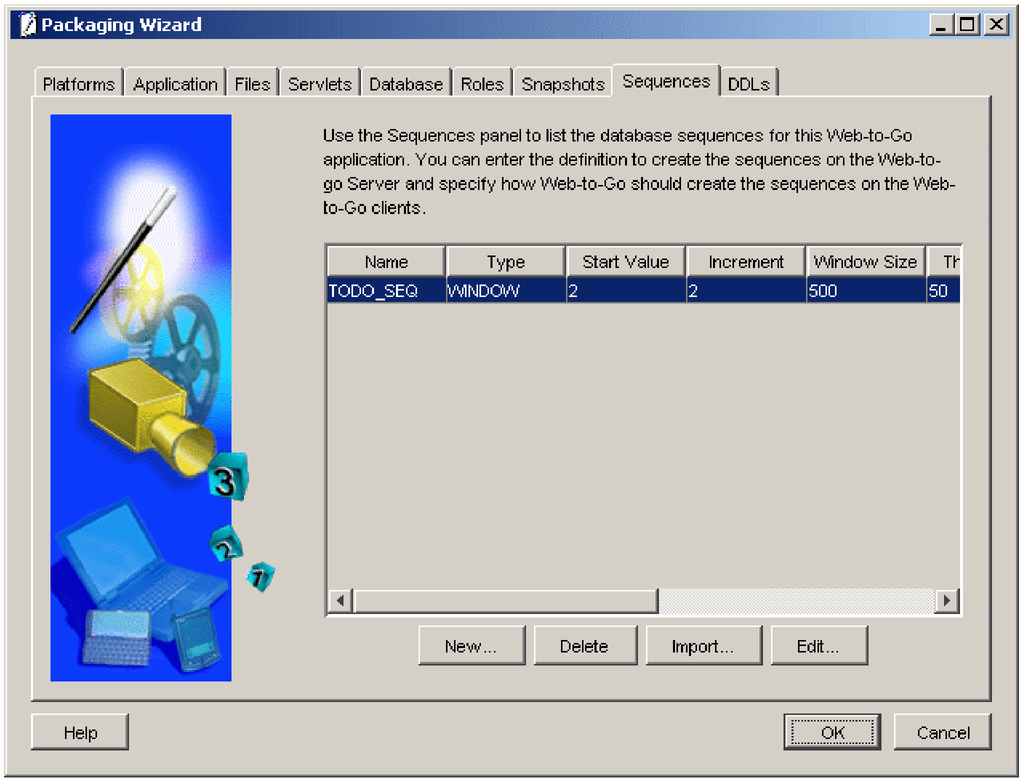
Feature	Description
Create on Client	When selected, the checkbox allows you to edit the snapshot on the Mobile Client for Web-to-Go.
Updatable	When selected, this check box creates an updatable snapshot of the named table.
Template	Displays the snapshot template for the named table. You can modify the 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 application, which uses sequences as unique primary key value. Sequences are important because they eliminate replication conflicts by preventing duplicate primary key values across disconnected applications. For full details of how to use and create sequences within Oracle Database Lite, see Section 6.6, "Create a Sequence".

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
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

Table 7–10 (Cont.) Sequences Dialog Description

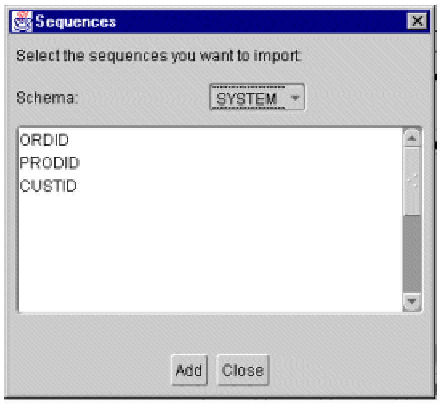
Field	Description	Required
Increment	The number by which the sequence increments on the Mobile Client for Web-to-Go, beginning at its start value.	Yes
Window Size	Defines the range of numbers in a window sequence.	Yes
Threshold	Defines the minimum range of required numbers in a window sequence. Web-to-Go 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:

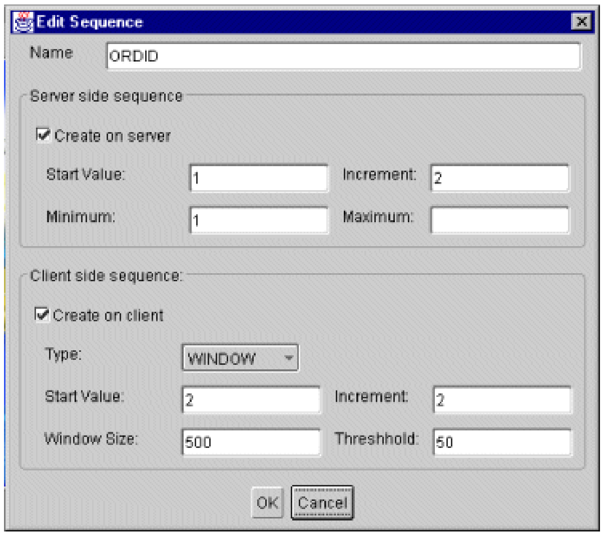
Schema: SYSTEM

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



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

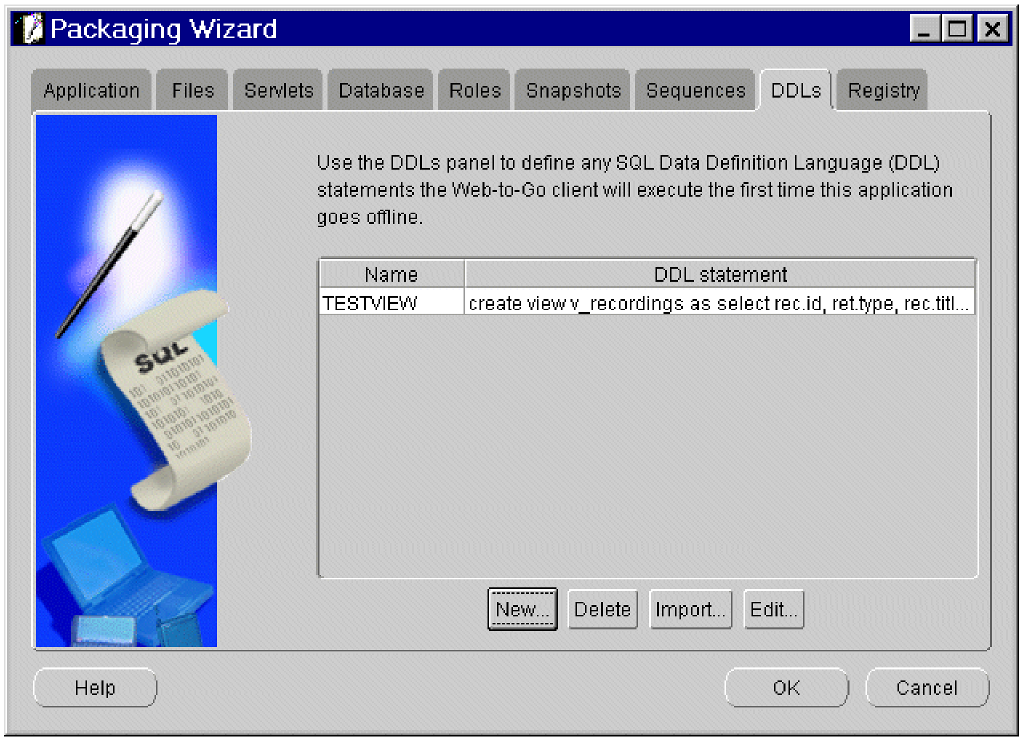
Feature	Description
Threshold	The minimum range of required numbers in a window sequence. Web-to-Go creates a new sequence when the existing one reaches this range and when the client disconnects from the back-end Oracle database. This information is not used by the leapfrog sequence.

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

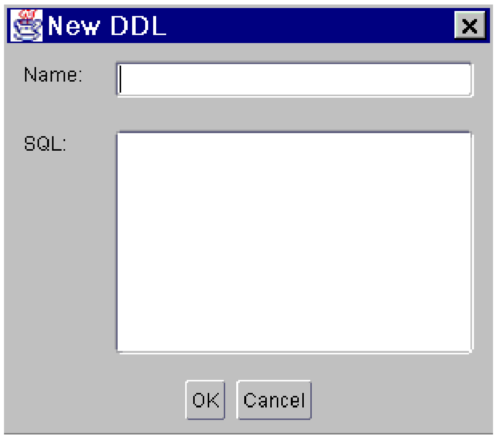
Field	Description
Name	The DDL name.
DDL Statement	Defines DDL statements with the Web-to-Go application. These DDL statements will be executed when the Web-to-Go application runs on the client.

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.

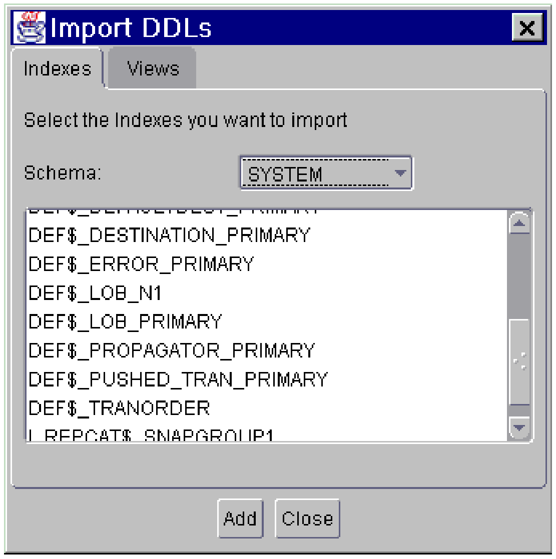
Figure 7–20 New DDL Dialog



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$_TRANORDER
I_REPCAT$_SNAPGROUP1
```

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

Create and Manage Jobs with APIs

The following sections describe how you can manage and create jobs with ConsolidatorManager APIs:

- Section 8.1, "Managing Scheduled Jobs Using ConsolidatorManager APIs"
- Section 8.2, "Start a Standalone Job Engine In Separate JVM"
- Section 8.3, "Using the ConsolidatorManager APIs to Create Jobs"

8.1 Managing Scheduled Jobs Using ConsolidatorManager APIs

Application developers can define, submit, and manage jobs programmatically based on a pre-determined time and interval. For example, jobs can be scheduled to run repeatedly for a specified duration on any specified day or days of the week or month. Administrators can schedule jobs to run repeatedly for a specified number of months, weeks or specified days of the month or week.

The Job Scheduler API schedules and executes jobs using a job engine. It is a generic component which enables apply and compose functions for MGP, device manager jobs, and custom jobs.

- Using the class `oracle.lite.sync.ConsolidatorManager`, application developers can register or de-register a job class, create, drop, enable or disable a job, search, and delete a job execution log.
- Use other supporting classes, such as `Job`, `Schedule`, `ExecutionResult` and `ExecutionLog` in the `oracle.lite.sync.job` package to manage your scheduled jobs.

For more information on these classes and their methods, refer to the *Oracle Database Lite API JavaDoc.*

8.2 Start a Standalone Job Engine In Separate JVM

If you want to execute a Standalone Job engine in a separate JVM from any of the Mobile Servers in the farm, then perform the following:

1. Retrieve a connection to the database with the Consolidator Manager `openConnection` method. Pass in the Mobile Manager administrator username, password and optionally, the JDBC URL to the back-end Oracle database.
2. Create a new Job engine with the `JobEngine` class and start it with the `startUp` method. The Standalone Job engine executes in a separate thread, which you can terminate from the main thread.
3. Define how long the thread is to sleep between execution of all jobs.

4. Terminate the Standalone Job engine when you have completed all activities.

Note: The following example demonstrates how to start up a Standalone Job engine in its own thread. It executes all of the jobs that have been scheduled either through the API or through the Mobile Manager Job Scheduler screens, because the Job Scheduler retrieves the scheduled job information from the repository.

```
JobEngine JobEngine = new JobEngine();
JobEngine.startUp();
if (JobEngine.runnerThreadException != null){
  System.out.println("runnerThreadException:");
  JobEngine.runnerThreadException.printStackTrace();
}

Thread.currentThread().sleep(60*1000);

if (JobEngine.runnerThreadException != null){
  System.out.println("runnerThreadException:");
  JobEngine.runnerThreadException.printStackTrace();
}
JobEngine.shutDown();
```

8.3 Using the ConsolidatorManager APIs to Create Jobs

Within the `oracle.lite.sync.ConsolidatorManager` class, there are several APIs, which are documented fully in the *Oracle Database Lite API JavaDoc*, that enable you to create, register, and schedule your job.

While these methods are described fully in the JavaDoc, the following demonstrates the order in which you would execute the methods:

1. Create your job class by implementing the `oracle.lite.job.Job` interface. Implement the Job interface methods, as follows:
   - `init` method—This method is invoked by the Job Scheduler when the job is loaded.
   - `execute` method—This method is invoked by the Job Scheduler when the job is scheduled to execute. Put a call into your application within this method. The Job Scheduler passes in the input parameter that was provided when the job is created—either with the `createJob` method or within the Mobile Manager Job Scheduler screen. When finished, the `execute` method returns an object of class type `ExecutionResult` containing whether the job was a success or failure.
   - `destroy` method—This method is invoked after the job completes.
2. After you have created your job class, register it with the `registerJobClass` method.
3. Create the job in the Job Scheduler by executing the `createJob` method. One of the input parameters is an object of class type `Schedule`, which defines when the job is executed. There are also other management methods that correspond to the Mobile Manager GUI, such as `dropJob`, `enableJob`, and `disableJob`.
4. If you want to retrieve any logs, execute the `getJobExecutionLogs` method, which retrieves objects of `ExecutionLog` class.

9

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. Supported functionality includes the client Oracle Lite database, access to the database through ODBC and JDBC and synchronization. However, at this time, no device management is supported.

This chapter helps to describe the supported features and how to use them on Symbian devices. You can read the rest of the Oracle Database Lite documentation for more details on these areas.

- Section 9.1, "Installing Oracle Database Lite on Symbian Devices"
- Section 9.2, "Developing Applications for Symbian Devices to Use Oracle Database Lite"
- Section 9.3, "Using CSQL, ODBC or JDBC to Access Oracle Database Lite"
- Section 9.4, "Invoking Synchronization from Applications on Symbian Devices"
- Section 9.5, "Use the Utility Tools on Symbian Devices"

9.1 Installing Oracle Database Lite on Symbian Devices

The following sections describe the pre-requisites and installation steps for Symbian devices:

- Section 9.1.1, "Supported Platforms and Environment"
- Section 9.1.2, "Prerequisites for Installation"
- Section 9.1.3, "Installing Oracle Database Lite"

9.1.1 Supported Platforms and Environment

Your development environment must include Oracle Database Lite 10*g* as the encompassing platform. For developing native applications with Oracle Database Lite 10*g* on a Symbian platform, see the following:

- Section 9.1.1.1, "Supported Devices for Symbian Platform"
- Section 9.1.1.2, "Symbian Operating System Support"
- Section 9.1.1.3, "Supported Development Environments for the Symbian Platform"

9.1.1.1 Supported Devices for Symbian Platform

The following devices are supported for the Symbian 7.x platform:

- Nokia 6620
- Nokia 9500
- Motorola M1000
- Sony Ericsson P910

The following devices are supported for the Symbian 8.x platform:

- Nokia 6630, which is also known in Japan as V702NK
- Nokia 6680

The following devices are supported for the Symbian 9.x platform:

- Nokia E61

9.1.1.2 Symbian Operating System Support

We support Symbian Operating System versions 7.x, 8.x, 9.x, UIQ 2.0, 2.1.

Table 9–1 Symbian Support

Platform	Programming Languages	Operating System	Hardware
Symbian OS on Nokia and Motorola	C, C++	Symbian OS versions 7.0, 8.0 and 9.0	ARM

9.1.1.3 Supported Development Environments for the Symbian Platform

The following are the supported development environments for Symbian 7 or 8:

- Microsoft Visual Studio 6.0
- Microsoft Visual Studio .Net

The supported development environment for Symbian 9 is Carbide.C++ Version 1.1 or Version 1.2.

9.1.2 Prerequisites for Installation

We assume that you have a basic Symbian OS development knowledge to develop your application.

Before installing Oracle Database Lite, perform the following:

- Installing the SDK:
  - You must install the SDK for the target device on the development machine. For example, if you are using a Motorola device, go to `www.motorola.com` and download the M1000 SDK on the development machine.
  - To build any application, install the S60 3rd Edition SDK for building and testing on the emulator. In addition, to build for the target device, install the CSL Toolchain (GCCE), which is a plug-in with the S60 3rd Edition SDK. The Toolchain has to be separately installed even though it has been downloaded with the SDK.
- Browsing, installation and uninstallation on the target device: Install the target device development suite, which is named as either "PC Suite" or "Desktop suite."
- Using command-line prompts on your device: Install `eshell.exe` on the device. In addition, we recommend that you purchase a hardware keyboard to connect to your phone to type in the `eshell.exe` window.

- Using JDBC: JDBC applications use the Multi-User (MU) component to connect to the Oracle Lite database. For Symbian 9, MU requires the Open-C library supplied by Nokia Open-C forum. Install Open-C sis files, such as `openc_glib.sis`, `openc_ssl.sis` and `pips_s60_wp.sis`, on the device.

9.1.3 Installing Oracle Database Lite

The following sections describe how to install Oracle Database Lite for the Symbian platform for the development environment or the Symbian device:

- Section 9.1.3.1, "Installing Oracle Database Lite for Symbian on the Development PC"
- Section 9.1.3.2, "Installing Oracle Database Lite on the Symbian Device"

9.1.3.1 Installing Oracle Database Lite for Symbian on the Development PC

The following sections describe how to install for each type of Symbian platform:

- Section 9.1.3.1.1, "For the Symbian 7 or 8 Development Environment"
- Section 9.1.3.1.2, "For Symbian 9 Development Environment"

9.1.3.1.1 For the Symbian 7 or 8 Development Environment Within the Oracle Universal Installer, perform the following to install the Symbian Development Kit on the developement PC:

1. Select **Custom**.
2. Select **Oracle Database Lite MDK for Symbian 7 & 8**.
3. Enter the directory where you installed the Symbian SDK, which is the same as the `EPOCROOT`.

   When you complete the installation, the following files are unzipped:

   - Header files are placed in the `epoc32\include\olite` directory.
   - ARMI (urel) `.lib` files are placed in the `epoc32\release\armi\urel` directory.
   - THUMB (urel) `.lib` files are place in the `epoc32\release\thumb\urel` directory.
   - WINS (udeb) binaries and `.lib` files are copied into the `epoc32\release\wins\udeb` directory.
   - Initial configuration files are copied into the `epoc32\wins\c\System\Data` directory.
   - Samples are copied into the `OliteEx` directory.

9.1.3.1.2 For Symbian 9 Development Environment Within the Oracle Universal Installer, perform the following to install the Symbian Development Kit on the developement PC:

1. Select **Custom**.
2. Select **Oracle Database Lite MDK for Symbian 9**.
3. Enter the directory where you installed the Symbian SDK, which is the same as the `EPOCROOT`.

When you complete the installation, the following files are unzipped:

- Header files are placed in the `epoc32\include\olite` directory.
- Initial configuration files are copied into the `epoc32\winscw\c\System\Data` directory.
- Samples are copied into the `OliteEx` directory.
- The ARMV5 (`lib`) `.dso` files are placed in the `epoc32\release\winscw\udeb` directory.
- The WINSCW (`udeb`) binaries and `.lib` files are placed in the `epoc32\release\winscw\udeb` directory.

9.1.3.2 Installing Oracle Database Lite on the Symbian Device

When you build your application, you need to include the header files and link against the libraries.

Once your application is built and tested, install the application and Oracle Database Lite on the Symbian platform as follows:

1. For most devices, copy and install the correct SIS `olite_core.sis` file to the device using PC Suite, Desktop Suite, or an external memory card. This installs all of the Oracle Database Lite files, including the DLLs and executables.

   The following lists the correct SIS file for each platform:

   - For the Symbian 7 or 8 platform, install the following SIS files:

     a. Use the `olite_core.sis` file or for the Sony Ericsson P910 device, the `olite_core_uiq2.sis` file.

     b. If using JDBC, then also install the `olite_server.sis` or for the Sony Ericsson P910 device, the `olite_server_uiq2.sis` file

   - For the Symbian 9 platform, install the `olite_core.sis` file.

   This installs the Oracle Database Lite binaries into the target drive. The default location is the `!:\System\Libs\` directory.

   Note: You may chose a different target directory during installation.

   If the configuration files do not already exist on the device, then the following files are copied into the `C:\System\Data\` directory: `polite.ini`, `odbc.ini`, and `olite40.msb`.

2. For Symbian 7 or 8, you have the option to install the Oracle Database Lite 10*g* Utility Tools. The Symbian 9 installation automatically installs all tools with the base install.

   Note: These utility tools are command line based programs; thus, you need to install and use the `eshell.exe` program to execute them.

   a. For most devices, copy the `olite_tools.sis` file to the target device using PC Suite, Desktop Suite, or a memory card. If you are using a Sony Ericsson P910, then copy the `olite_tools_uiq2x.sis` file.

   b. Install either the `olite_tools.sis` file or if using the Sony Ericsson P910 device, the `olite_tools_uiq2x.sis` file. This copies the following files into the target directory (which by default is the `!:\System\Programs\` directory): `CREATEDB.EXE`, `REMOVEDB.EXE`, `ENCRYPDB.EXE`, `DECRYPDB.EXE`, and `ODBINFO.EXE`.

Note: You may chose a different target directory during installation.

3. Install your application.

9.2 Developing Applications for Symbian Devices to Use Oracle Database Lite

Symbian applications that need a standard interface and work with multiple database engines can use either the JDBC interface, the ODBC interface or some other interface built on top of ODBC, as follows:

- Your application can use ODBC to access the database directly.
- To use JDBC, then the application must access the database through the multi-user service for the JDBC connection. Therefore, if you plan to use JDBC, you must also install and configure for the multi-user service.

When you are developing applications for the Symbian environment, you can use the following:

- For your development language, you can use C, C++, or Java APIs.
- Symbian applications that need a standard interface and work with multiple database engines can use either the JDBC interface, the ODBC interface or some other interface built on top of ODBC.

  Your application can use ODBC to access the database directly; however, if you want to use JDBC, then you must access the database using the multi-user service for the JDBC connection.

9.3 Using CSQL, ODBC or JDBC to Access Oracle Database Lite

The following sections describe how to use ODBC or JDBC to access the Oracle Lite database on the Symbian device:

- Section 9.3.1, "Using CSQL to Connect to the Database on Symbian"
- Section 9.3.2, "Using ODBC to Connect to the Database on Symbian"
- Section 9.3.3, "Using JDBC to Connect to the Database on Symbian"

9.3.1 Using CSQL to Connect to the Database on Symbian

CSQL is a sample application provided in the Oracle Database Lite installation for Symbian. It is an ODBC-based application with which the user can perform SQL operations on the Oracle Lite database. The application can be built for the target device as well as the emulator.

You can configure how CSQL accesses the Oracle Lite database in the `polite.ini` and `odbc.ini` configuration files.

9.3.2 Using ODBC to Connect to the Database on Symbian

When you are developing ODBC based applications, perform the following:

Note: For an example, see the CSQL example for Symbian in the `<EPOCROOT>\OliteEx\CoreDB\CSQL` directory.

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 uses STDLIB resources. You need to call `CloseSTDLIB()` after all database operations to free up resources.
5. Character data stored in Oracle Database Lite 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 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>`.

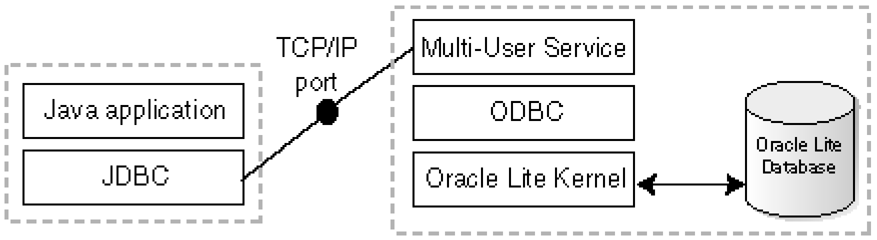
   b. Invoke the `CloseSTDLIB()` method after each `SQLFreeEnv()` call.

9.3.3 Using JDBC to Connect to the Database on Symbian

On the Symbian platform, only one JDBC connection is supported. You can use a JDBC driver for J2ME CLDC—in a limited capacity—for Java applications to connect and update the database. The CLDC compliant JDBC driver for the Oracle Lite is located in the `olitejdbccldc.jar` file. The API documentation is available in the Oracle Lite SDK for Symbian. Otherwise, you can see Section 7.8.1.2, "JDBC Driver for JDBC CLDC" in the *Oracle Database Lite Client Guide* for more details.

When you use JDBC for connecting to the Oracle Lite database, you have to use the Multi-User listener. Symbian does not support JNI to map the Java request into native code. Therefore, as shown in Figure 9–1, JDBC accesses the Oracle Lite database in client/server mode using a TCP/IP connection. The default TCP/IP port for the Multi-User listener is port 1160 .

Figure 9–1 Java Application on Symbian Accessing Database



Within the main Oracle Database Lite documentation, there is a discussion on how to configure and start the Multi-User service. However, for the Symbian platform, as soon as the SIS file is installed, the Multi-User service automatically starts and receives all incoming requests on port 1160.

9.4 Invoking 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 9.4.2.1, "Prepare Your Application for Synchronization"
- Section 9.4.2.2, "How to Use the Synchronization API for Symbian Devices"

9.4.1 Using MSync UI to Invoke Synchronization

MSync is a sample application provided in the Oracle Lite installation. This can be used to synchronize the Oracle Lite database on the Symbian device with the Mobile Server. Msync is included in the binary for Symbian 9 devices.

Before you can use MSync on the Symbian 9 device, you must configure the following parameters in the `msync.ini` configuration file:

- `USER=`<*username*>
- `PASS=`<*password*>
- `URL=`<*url or IP address*>

The `msync.ini` file is located on the same drive where you installed Oracle Database Lite.

For example, the following parameters show that the Mobile Server IP address is `192.168.1.2`, the application user is `S11U1`, and the password is `abcd`.

```
USER=S11U1
PASS=abcd
URL=192.168.1.2
```

9.4.2 Invoking Synchronization through Programmatic APIs

The following sections describe how to invoke synchronization from within the application:

- Section 9.4.2.1, "Prepare Your Application for Synchronization"

- Section 9.4.2.2, "How to Use the Synchronization API for Symbian Devices"

9.4.2.1 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.

9.4.2.2 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.

9.5 Use the Utility Tools on Symbian Devices

The utility tools that are available for Symbian are as follows: `csql`, `msync`, `createdb`, `removedb`, `encryptdb`, `decryptdb`, and `odbinfo`.

Note: Before you can use any utility tools, ensure that they are installed on the device, as described in Section 9.1.3, "Installing Oracle Database Lite". For Symbian 9, all tools are automatically installed.

To be able to use the utility tools, see the appropriate section based upon the Symbian platform that you are using:

- Section 9.5.1, "Using Utility Tools on Symbian 7 and 8"
- Section 9.5.2, "Using Utility Tools on Symbian 9"

9.5.1 Using Utility Tools on Symbian 7 and 8

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.

9.5.2 Using Utility Tools on Symbian 9

To use the database utility tools on the emulator or device for Symbian 9, perform the following:

1. Click the installation folder on the emulator or device and then select the utility icon.
2. Follow the UI menu and provide the appropriate arguments for the tools.

10

Customizing Oracle Database Lite Security

Managing the provided security within Oracle Database Lite is described in Chapter 12, "Configuring Security in Oracle Database Lite" in the *Oracle Database Lite Administration and Deployment Guide*. This chapter describes how to customize authentication to provide your own mechanisms to be used within Oracle Database Lite.

The following section details security issues for Oracle Database Lite:

- Section 10.1, "Providing Your Own Authentication Mechanism for Authenticating Users for the Mobile Server"

10.1 Providing Your Own Authentication Mechanism for Authenticating Users for the Mobile Server

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.

The following sections describe how to implement and use an external authentication method for Oracle Database Lite:

- Section 10.1.1, "Implementing Your External Authenticator"
- Section 10.1.2, "Registering External Authenticator"
- Section 10.1.3, "User Initialization Scripts"

10.1.1 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: Samples `SampleAuthenticator.java` and `OIDAuthenticator.java` demonstrate how to implement an external authenticator. These samples can be found in the *<ORACLE_HOME>*`\Mobile\Server\demos\devmgr\java\` directory.

Implement the following methods in your external authenticator. The Mobile Server invokes each of these methods as appropriately.

- Section 10.1.1.1, "Initialization for the External Authenticator"
- Section 10.1.1.2, "Destruction of the External Authenticator"
- Section 10.1.1.3, "The Authentication Method for the External Authenticator"
- Section 10.1.1.4, "The User Instantiation Method for the External Authenticator"
- Section 10.1.1.5, "Retrieve the User Name or the User Global Unique ID"
- Section 10.1.1.6, "Log Off User"
- Section 10.1.1.7, "Change User Password"

10.1.1.1 Initialization 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)
```

10.1.1.2 Destruction of 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
```

10.1.1.3 The Authentication Method for the External Authenticator

Authenticate a user and return a session handle with the `authenticate` method. 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
```

You can pass error and warning information, as follows:

- Failure: Pass along any error information, such as why the authentication failed. Use the `AuthException` class, available in the package `oracle.lite.provider.auth`, to pass along failure information.
- Warning: Pass along any warnings, such as the situation when the user's password is about to expire. Use the `ExtAuthResult` class, available in the package `oracle.lite.provider.auth`, to pass along warning information.

Refer to the *Oracle Database Lite API Specification* for more details on these exception classes.

10.1.1.4 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 10.1.3, "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
```

10.1.1.5 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
```

10.1.1.6 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
```

10.1.1.7 Change User Password

```
Method: void changePassword (Object sid, String pwd) throws SecurityException
Parameter: Session handle returned by the authenticate method and new password
string
```

10.1.2 Registering External Authenticator

The EXTERNAL_AUTHENTICATION parameter in the WEBTOGO.ORA file facilitates the authentication of existing external users with the specified external authenticator class. To register your external authenticator class, 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.

In addition, you must configure the EXTERNALUSER parameter in the WSH.INI script, which notifies the server that the user being created is external and does not require a password in the WSH.INI script. Instead, the new user will be authenticated by the external authenticator specified in the WEBTOGO.ORA file. For more information on EXTERNALUSER parameter, see Appendix C, "Write Scripts for the

Mobile Server with the WSH Tool" in the *Oracle Database Lite Administration and Deployment Guide*.

10.1.3 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
   ```

11

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 Section 5.5, "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 11.1, "Develop the Application"
- Section 11.2, "Create Publication for Application"
- Section 11.3, "Package the Application Using the Packaging Wizard"
- Section 11.4, "Administer the Application"
- Section 11.5, "Execute the Application on the Mobile Client for Web-to-Go"

11.1 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 11–1 shows the components for the "To Do List" application:

Table 11–1 "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 11.1.1, "Create Database Objects in the Oracle Server"
- Section 11.1.2, "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.

11.1.1 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 11–2 shows the To Do Items table columns.

Table 11–2 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.

11.1.1.1 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;
```

11.1.1.2 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.

11.1.2 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
      ```

11.2 Create Publication for Application

As described fully in Chapter 6, "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 6, "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 11.2.1, "Create a Project"
- Section 11.2.2, "Create Publication Items"

- Section 11.2.3, "Create Publication"

11.2.1 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 6.2, "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**.

11.2.2 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 6.4, "Create a Publication Item".

The following sections describe how to create the publication item, the sequence, and an optional script for this publication:

- Section 11.2.2.1, "Create Publication Item"
- Section 11.2.2.2, "Create Sequence"
- Section 11.2.2.3, "Create Script"

11.2.2.1 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**.

11.2.2.2 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.

11.2.2.3 Create Script

Optionally, if you want members to have access to this tutorial on the Mobile client, then create the scripts for the To Do List demo, as follows:

1. Start the script wizard by selecting **File->New->Script**.
2. Enter the name as todo_script
3. Click **Browse** and select todo_scripts.sql from the following location:

   ```
   <ORACLE_HOME>\Mobile\Sdk\wtgsdk\src\tutorial
   ```

4. Click **OK**.

11.2.3 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. If you added the script described in Section 11.2.2.3, "Create Script", then in the Script tab, click **Add** to add the `todo_script`. Click **OK**.
5. 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
6. Save the publication by selecting **File->Save** and exit MDW.
7. If a window displays with the text: "Apply changes to Repository window," then click **Select All**. Click **OK** to apply the selected changes to the Repository.

11.3 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:

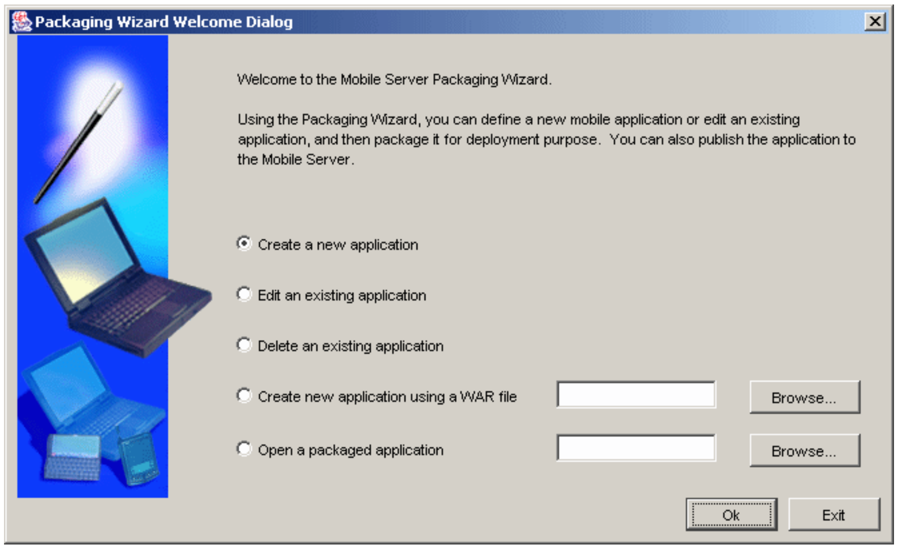
1. Start the Packaging Wizard, as follows:

   ```
   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 11–1.

   Note: Deleting an existing application merely deletes the application from the XML file and does not remove the application from the Mobile Server.

Figure 11–1 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

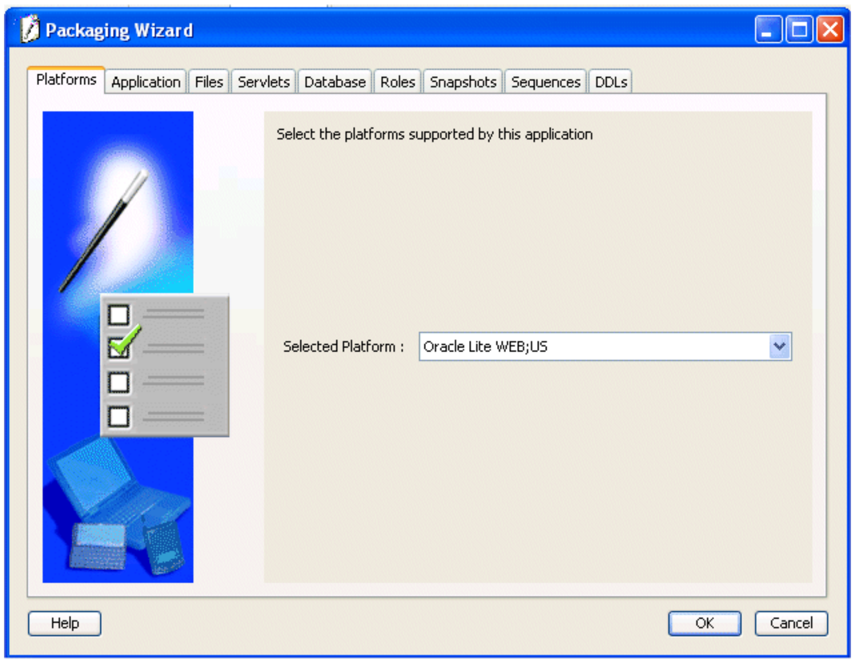
( ) Delete an existing application

( ) 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 11–2 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 11–2 Selecting a Platform



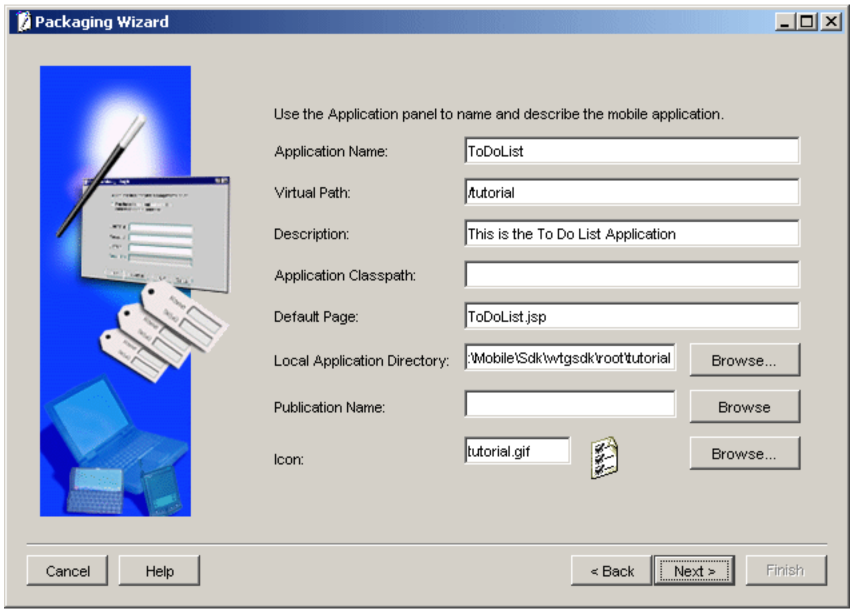
Packaging Wizard

Select the platforms supported by this application

Selected Platform : Oracle Lite WEB;US

4. As Figure 11–3 displays, the Application panel appears. Use the Application panel to modify "To Do List" application settings. As Table 11–3 describes, enter the specified values in the corresponding fields.

Figure 11–3 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

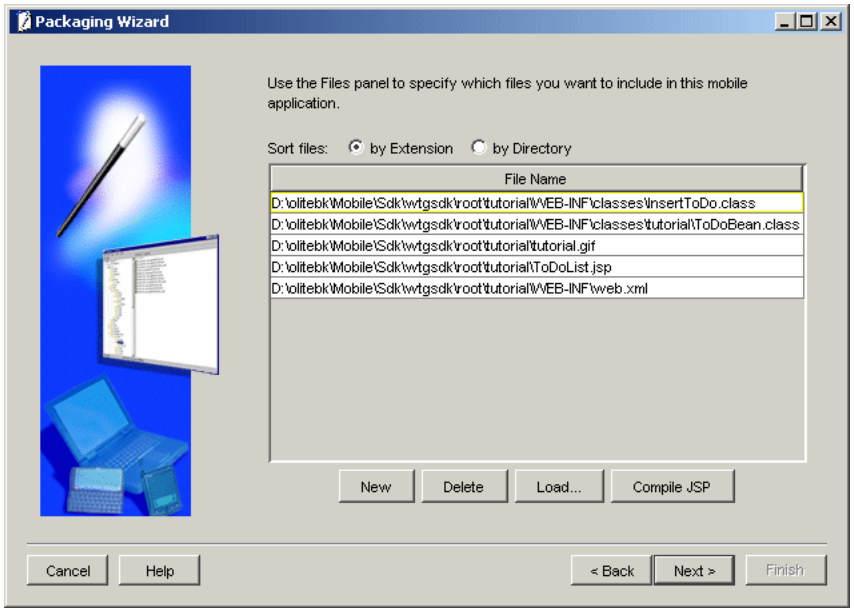
Table 11–3 The "To Do List" Application Values

Field	Value
Application Name	ToDoList
Virtual Path	`/tutorial`
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 11–4 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 11–4 Uploading Application Files



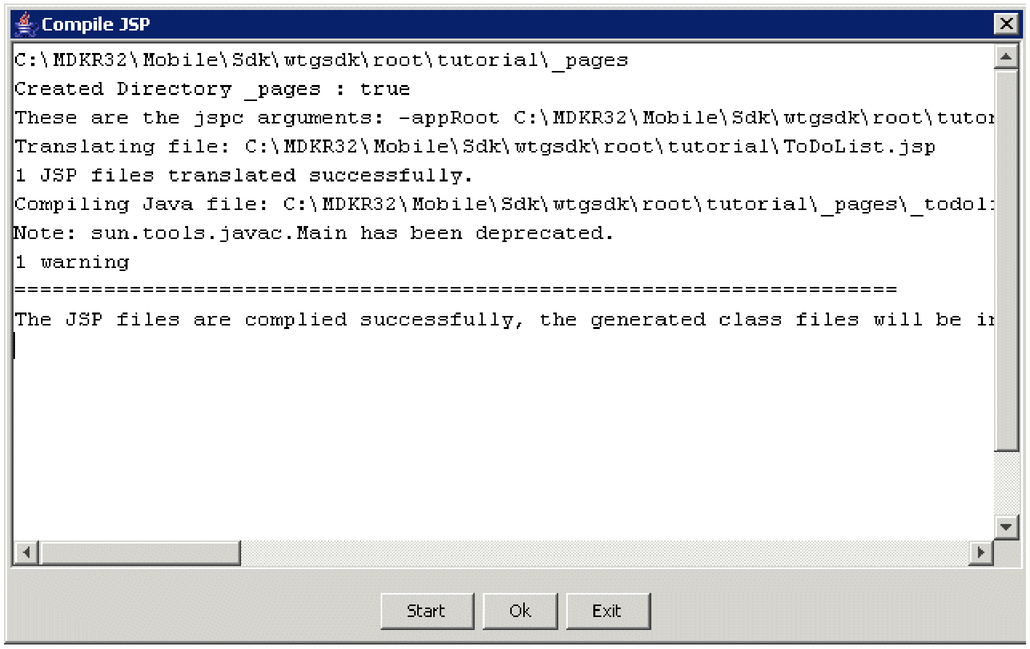
Use the Files panel to specify which files you want to include in this mobile application.

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 11–5 displays, the following confirmation page appears. Click **OK**.

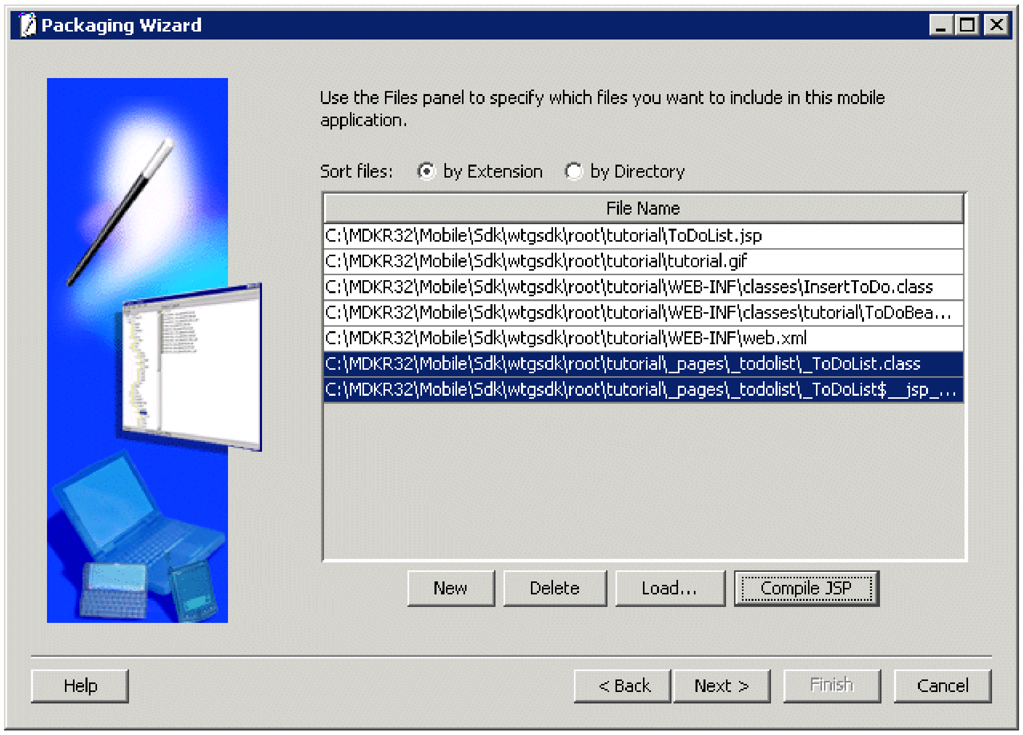
Figure 11–5 JSP Compilation Completion Message



```
C:\MDKR32\Mobile\Sdk\wtgsdk\root\tutorial\_pages
Created Directory _pages : true
These are the jspc arguments: -appRoot C:\MDKR32\Mobile\Sdk\wtgsdk\root\tutor
Translating file: C:\MDKR32\Mobile\Sdk\wtgsdk\root\tutorial\ToDoList.jsp
1 JSP files translated successfully.
Compiling Java file: C:\MDKR32\Mobile\Sdk\wtgsdk\root\tutorial\_pages\_todol
Note: sun.tools.javac.Main has been deprecated.
1 warning
=====================================================================
The JSP files are complied successfully, the generated class files will be i
```

7. The generated files are automatically added to the list of application files.

Figure 11–6 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:\MDKR32\Mobile\Sdk\wtgsdk\root\tutorial\ToDoList.jsp
C:\MDKR32\Mobile\Sdk\wtgsdk\root\tutorial\tutorial.gif
C:\MDKR32\Mobile\Sdk\wtgsdk\root\tutorial\WEB-INF\classes\InsertToDo.class
C:\MDKR32\Mobile\Sdk\wtgsdk\root\tutorial\WEB-INF\classes\tutorial\ToDoBea...
C:\MDKR32\Mobile\Sdk\wtgsdk\root\tutorial\WEB-INF\web.xml
C:\MDKR32\Mobile\Sdk\wtgsdk\root\tutorial_pages_todolist_ToDoList.class
C:\MDKR32\Mobile\Sdk\wtgsdk\root\tutorial_pages_todolist_ToDoList$__jsp_...

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.

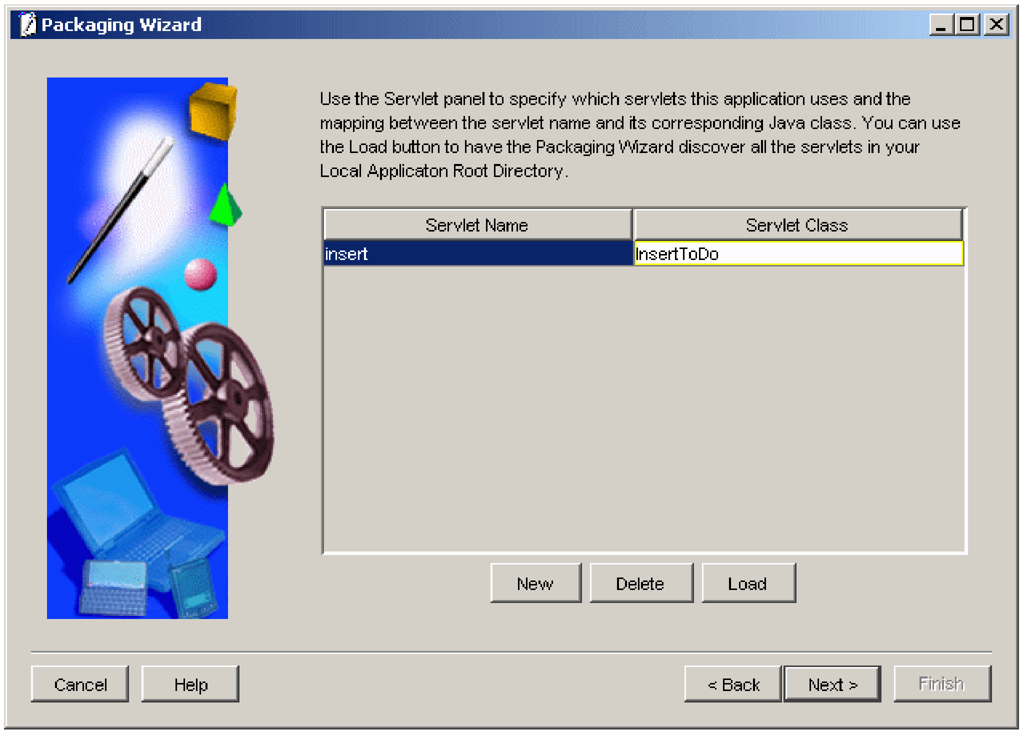
As Figure 11–7 displays, you can view the "To Do List" application servlet in the Servlets panel. Since the "To Do List" application contains only one servlet, the Servlets panel displays a single line.

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 11–7 Registering Servlets

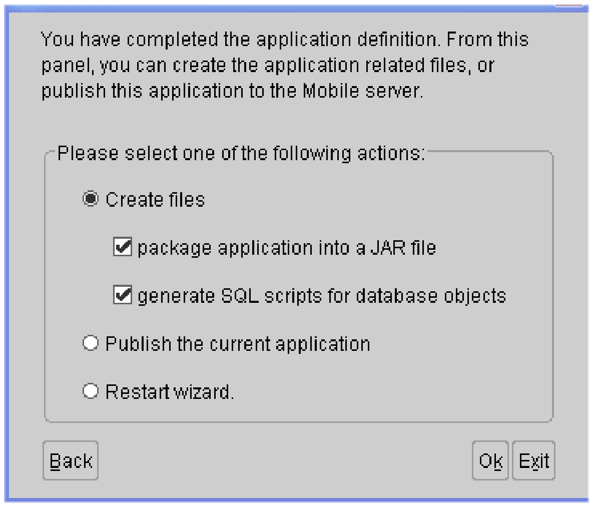


Use the Servlet panel to specify which servlets this application uses and the mapping between the servlet name and its corresponding Java class. You can use the Load button to have the Packaging Wizard discover all the servlets in your Local Applicaton Root Directory.

Servlet Name	Servlet Class
insert	InsertToDo

9. Click **Next** till you arrive at the Application Definition Completed Dialog as shown in Figure 11–8.

Figure 11–8 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.

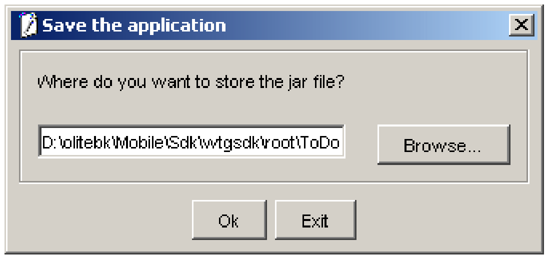
Using the Application Definition Completed panel, you can package the "To Do List" application into a JAR file. The Application Definition Completed Dialog remains open for you to initiate application packaging.

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 11–9 displays. The default location is given below.

```
<ORACLE_HOME>\Mobile\Sdk\wtgsdk\root\ToDoList.jar
```

Figure 11–9 Save the Application Dialog



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 11–4 describes, enter the specified values.

Note: The Mobile Server must be up for successful publishing.

Table 11–4 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.

11.4 Administer the Application

This section describes how to administer the application that you created and deployed through the following tasks.

- Section 11.4.1, "Start the Mobile Server and the Mobile Manager"

- Section 11.4.2, "Using the Mobile Manager to Create a New User"
- Section 11.4.3, "Setting Application Properties"
- Section 11.4.4, "Granting User Access to the Application"
- Section 11.4.5, "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*.

11.4.1 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
   ```

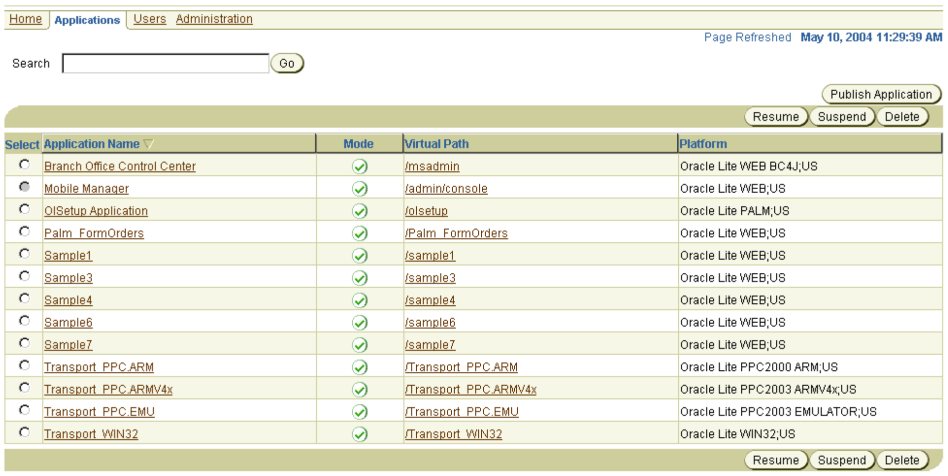
2. To start the Mobile Server for the first time and subsequent occasions, execute the `runmobileserver` command.
3. Start your Web browser and connect to the Mobile Server by enter the following URL:

   ```
   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 11–10 displays, the Applications page appears. Locate the To Do List application, which should be there since you published it.

Figure 11–10 Applications Page



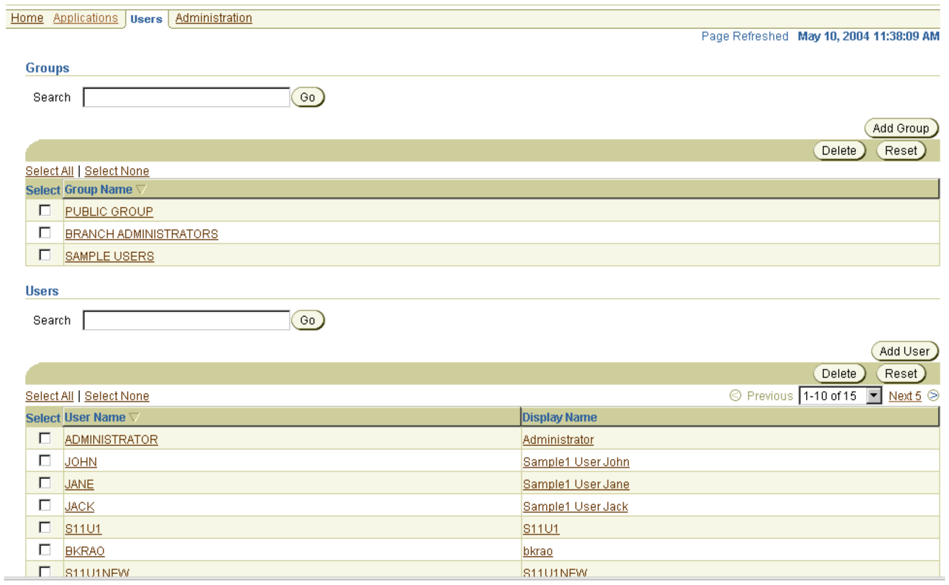
Select	Application Name ▽	Mode	Virtual Path	Platform
○	Branch Office Control Center	✓	/msadmin	Oracle Lite WEB BC4J;US
●	Mobile Manager	✓	/admin/console	Oracle Lite WEB;US
○	OlSetup Application	✓	/olsetup	Oracle Lite PALM;US
○	Palm_FormOrders	✓	/Palm_FormOrders	Oracle Lite WEB;US
○	Sample1	✓	/sample1	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

11.4.2 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 11–11 displays, the Users page appears.

Figure 11–11 Users Page



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

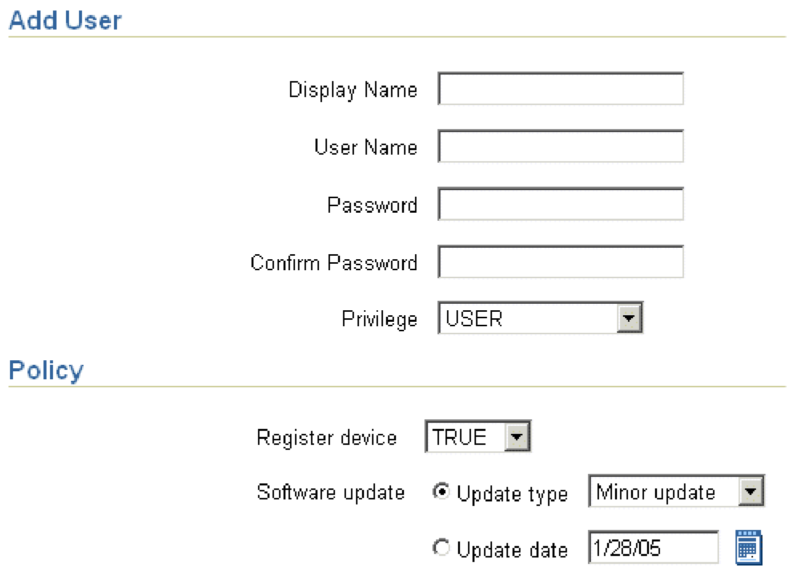
Delete Reset

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 11–12 displays, the Add User page appears.

Figure 11–12 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 11–5, enter the following information in the Add User page and click **Save**.

Table 11–5 Add User Page Description

Field	Value
Display Name	`tutorial`
User Name	`tutorial`
Password	`tutorial`
Password Confirm	`tutorial`
Privilege	User
Register Device	True
Software Update	Select all updates

11.4.3 Setting Application Properties

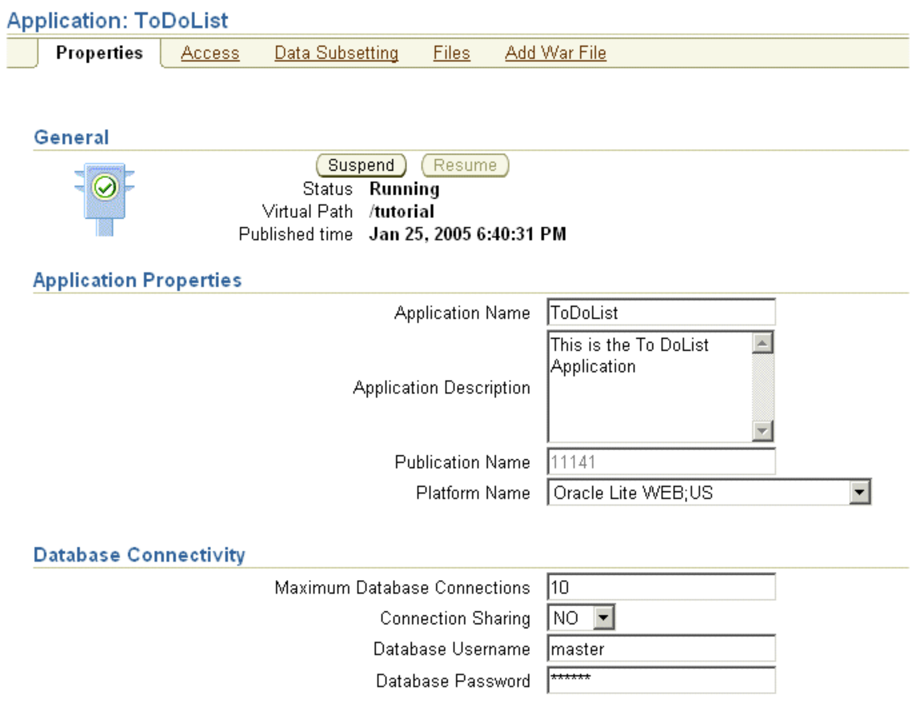
To set the "To Do List" application properties, perform the following steps.

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 11–13 displays, the Application Properties page lists application properties and database connectivity details.

Figure 11–13 Application Properties Page



Application: ToDoList

General

Status	Running
Virtual Path	/tutorial
Published time	Jan 25, 2005 6:40:31 PM

Application Properties

Application Name	ToDoList
Application Description	This is the To DoList Application
Publication Name	11141
Platform Name	Oracle Lite WEB;US

Database Connectivity

Maximum Database Connections	10
Connection Sharing	NO
Database Username	master
Database Password	******

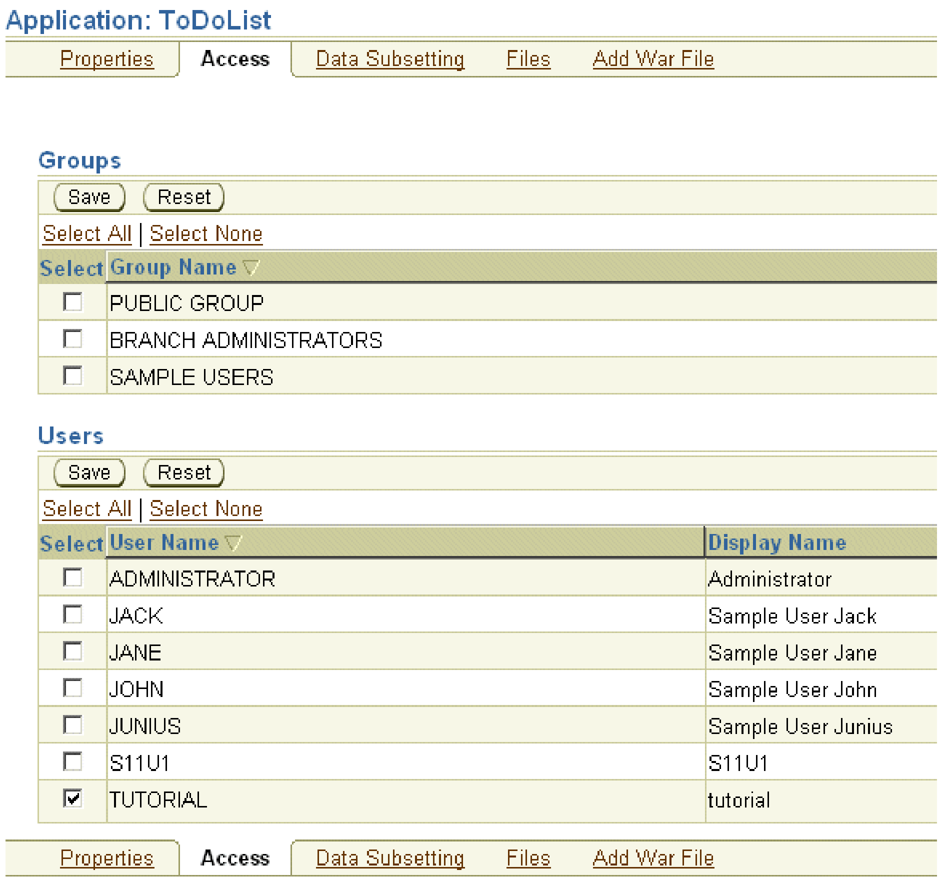
4. In the **Database Password** field type `master`. This is the default password for the Web-to-Go demo schema. Click **Apply**. The Mobile Manager displays a confirmation message.

11.4.4 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 11–14 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 11–14 Access Page


Application: ToDoList

Groups

Select	Group Name
☐	PUBLIC GROUP
☐	BRANCH ADMINISTRATORS
☐	SAMPLE USERS

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.

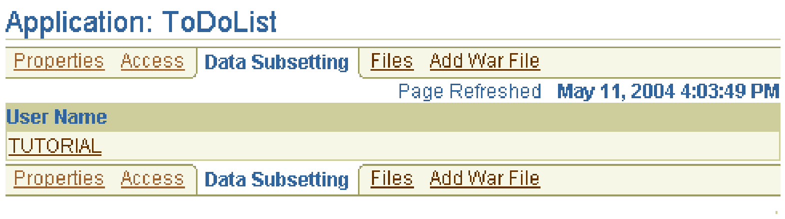
11.4.5 Defining Snapshot Template Values for the User

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 11–15 displays, the Data Subsetting page appears.

Figure 11–15 Data Subsetting Page


Application: ToDoList

Properties Access Data Subsetting Files Add War File

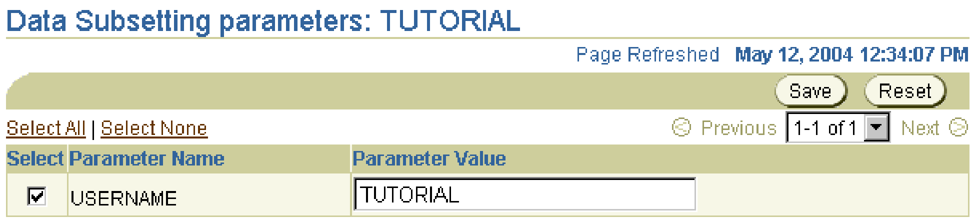
Page Refreshed May 11, 2004 4:03:49 PM

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 11–16 displays, the Data Subsetting Parameters page appears.

Figure 11–16 Data Subsetting Parameters Page


Data Subsetting parameters: TUTORIAL

Page Refreshed May 12, 2004 12:34:07 PM

Save Reset

Select All | Select None — Previous 1-1 of 1 Next

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*.

11.5 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 11.5.1, "Install the Mobile Client for Web-to-Go"
- Section 11.5.2, "Log into the Mobile Client for Web-to-Go"
- Section 11.5.3, "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.

11.5.1 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.

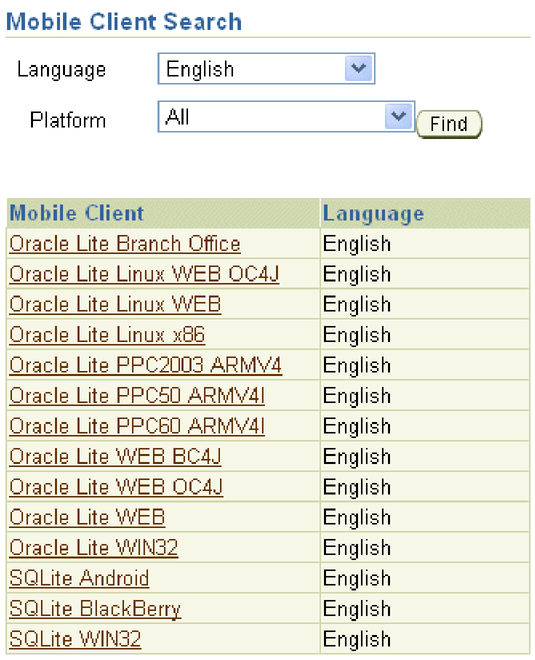
To install the Mobile Client for Web-to-Go, perform the following actions.

1. Start your Web browser and connect to the Mobile Server by entering the following URL.

```
http://<mobile_server>/webtogo/setup
```

2. As Figure 11–17 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 11–17 Mobile Client Setup Page



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 PPC2003 ARMV4	English
Oracle Lite PPC50 ARMV4I	English
Oracle Lite PPC60 ARMV4I	English
Oracle Lite WEB BC4J	English
Oracle Lite WEB OC4J	English
Oracle Lite WEB	English
Oracle Lite WIN32	English
SQLite Android	English
SQLite BlackBerry	English
SQLite 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.

---

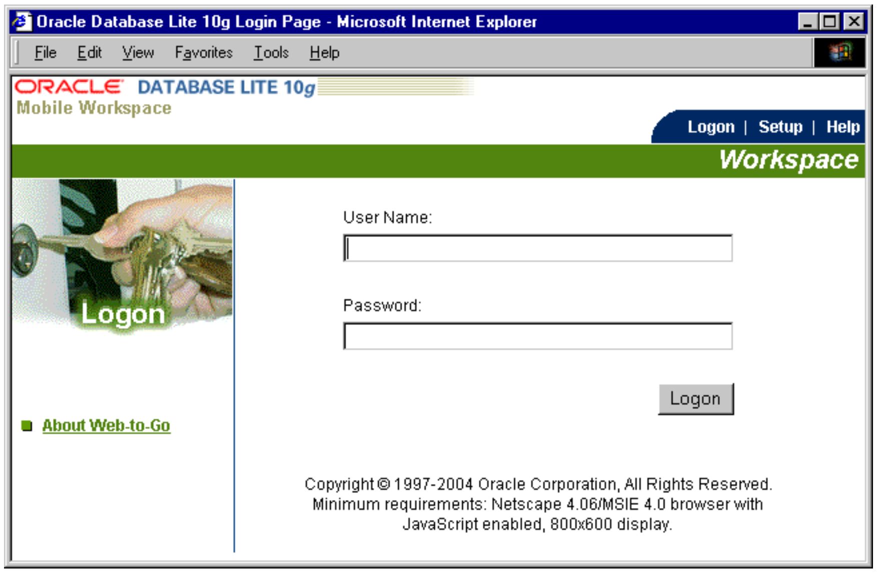
3. If you are using Netscape, choose a location to save the setup program and click **OK**. In Windows Explorer, double-click `setup.exe` to run the setup program.

   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 11–18 displays.

Figure 11–18 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.

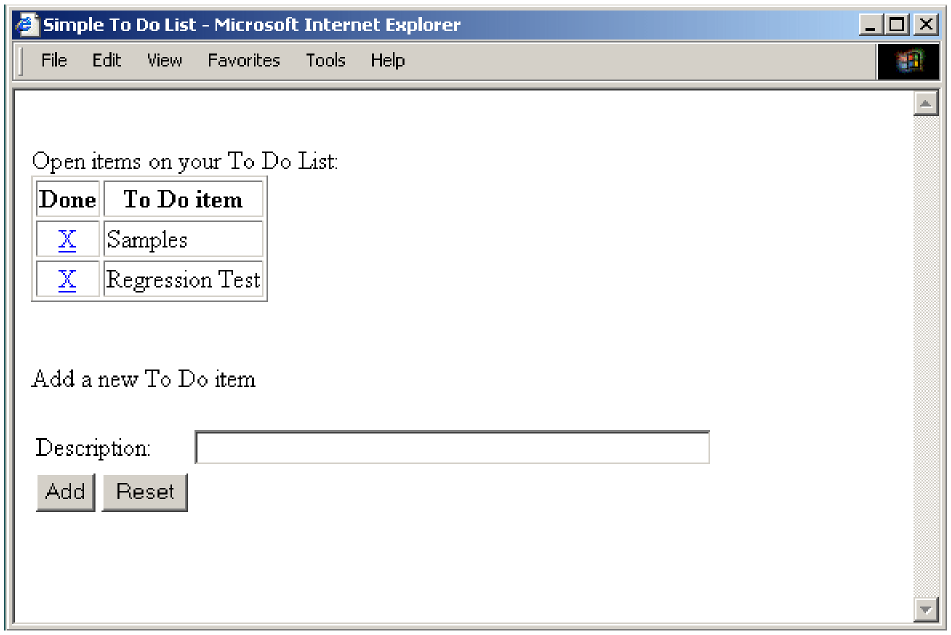
11.5.2 Log into the Mobile Client for Web-to-Go

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 11–19 displays, Web-to-Go launches the "To Do List" application in your browser.

Figure 11–19 The "To Do List" Application


Simple To Do List - Microsoft Internet Explorer

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**.

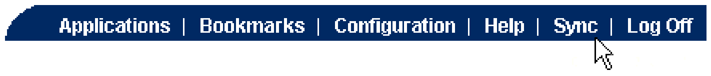
8. Exit the application by closing the browser window. This action returns you to the workspace.

11.5.3 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 11–20 displays, click the Sync tab in the upper right corner of the workspace.

Figure 11–20 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.

12 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:

12.1 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.

12.1.1 Before You Start

Ensure that the computer you are using for your development meets the requirements specified in this section. Table 12–1 lists configuration and installation requirements for the development computer.

Table 12–1 Development Computer Requirements

Requirement	Description
Windows User Login	The Windows 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.

12.2 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 12.2.1, "Creating a Database Connection to Oracle Database"
- Section 12.2.2, "Specify The Connection To The Oracle Lite Database"

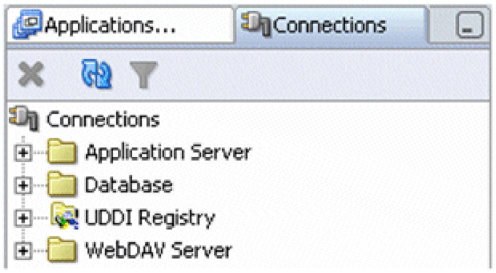
12.2.1 Creating a Database Connection to Oracle Database

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.

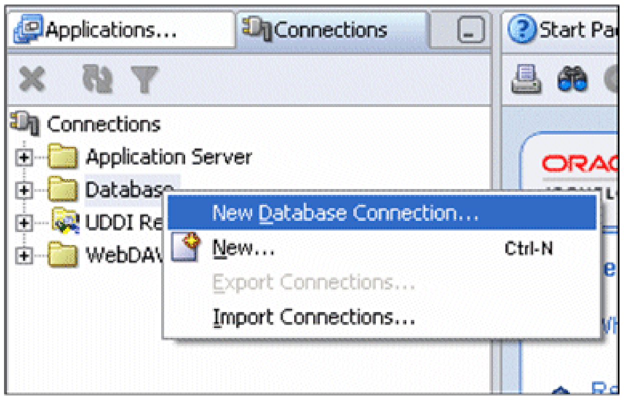
Note: If the Connections tab is not showing, choose **View -> Connection Navigator** from the JDeveloper main menu.

Figure 12–1 JDeveloper Connection tab on the Connection Navigator



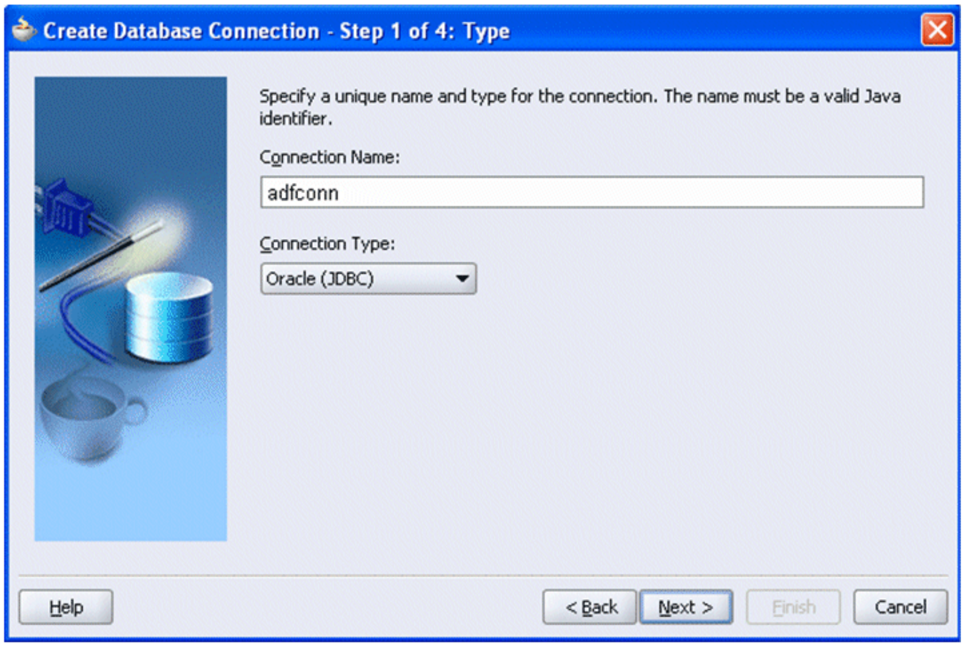
4. Right-click the `Database` folder and select **New Database Connection**. This starts the Create Database Connection Wizard.

Figure 12–2 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 12–3 Connection Wizard - Step 1 of 4: Type Panel



Create Database Connection - Step 1 of 4: Type

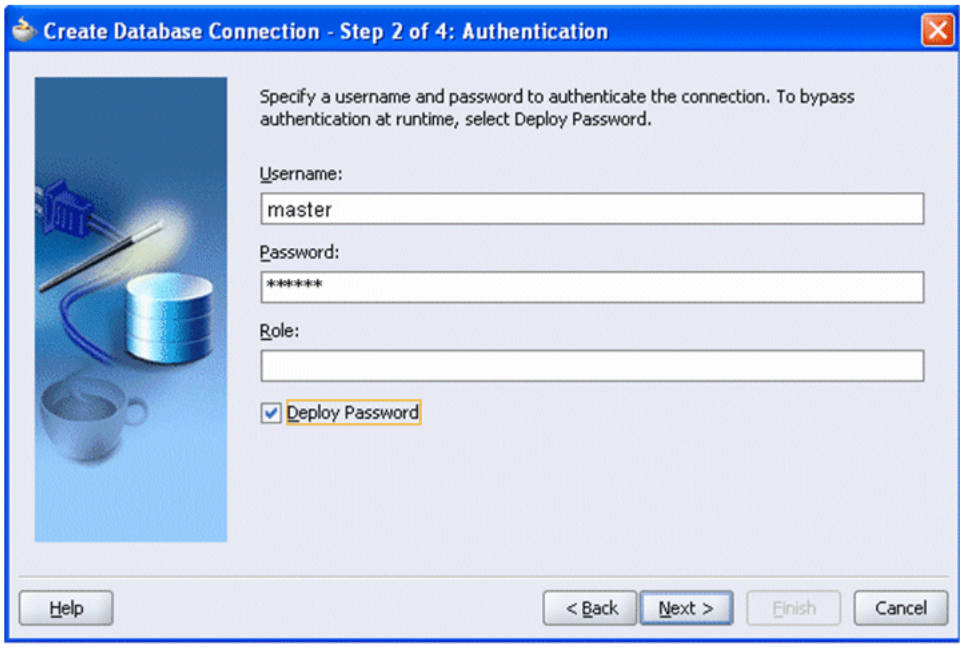
Specify a unique name and type for the connection. The name must be a valid Java identifier.

Connection Name: adfconn

Connection Type: Oracle (JDBC)

c. On the Authentication page, enter `master/master` for the username/password fields. Select **Deploy password**.

Figure 12–4 Connection Wizard - Step 2 of 4: Authentication Panel



Create Database Connection - Step 2 of 4: Authentication

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.

12.2.2 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.

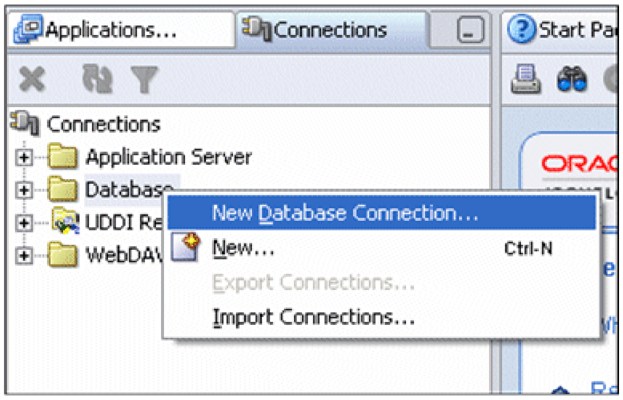
2. Select the Connections tab on the Application Navigator.

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 12–5 JDeveloper Connection tab on the Connection Navigator, New Database Connection



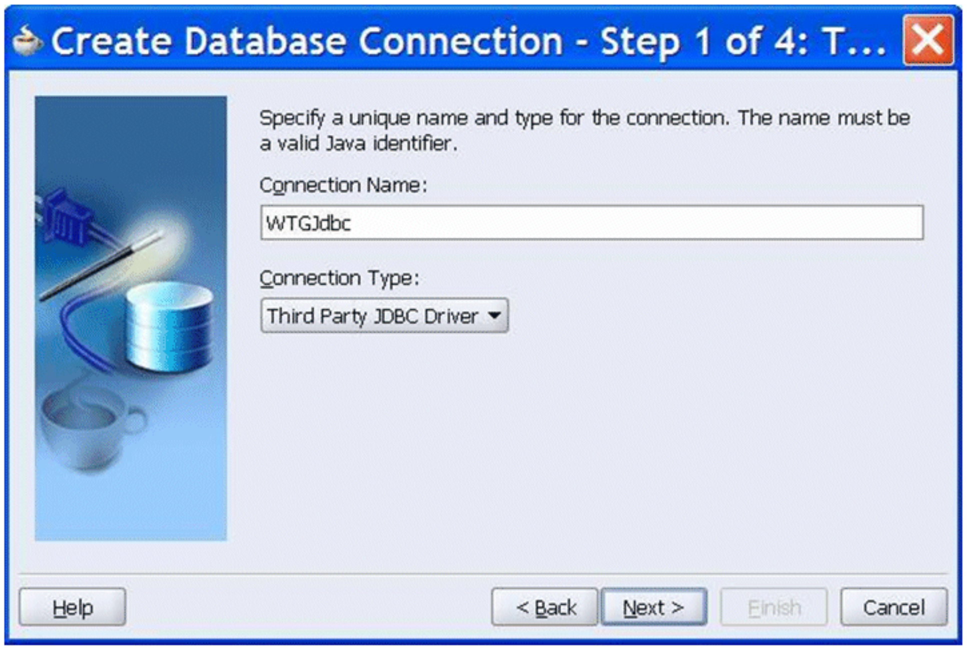
The Create Database Connection Wizard starts. Perform the following in creating a new database connection using this wizard:

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 12–6 Connection Wizard - Step 1 of 4: Type Panel


Create Database Connection - Step 1 of 4: T...

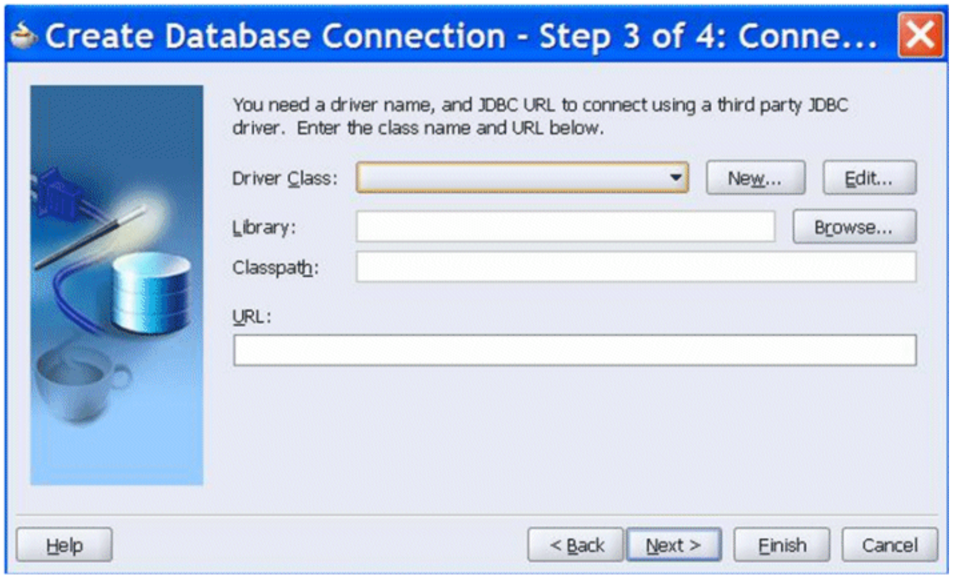
Specify a unique name and type for the connection. The name must be a valid Java identifier.

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 12–7 Connection Wizard - Step 3 of 4: Connection Panel


Create Database Connection - Step 3 of 4: Conne...

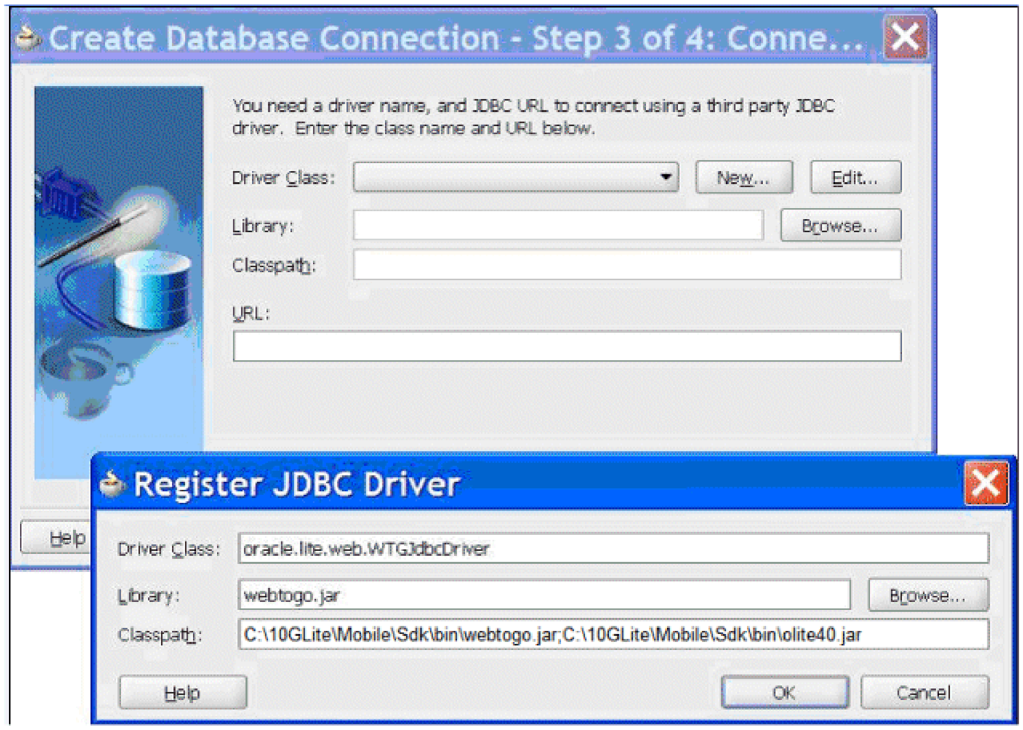
You need a driver name, and JDBC URL to connect using a third party JDBC driver. Enter the class name and URL below.

Driver Class:
Library:
Classpath:
URL:

6. The Register JDBC Driver dialog appears, as Figure 12–8 displays. Perform the following:

   a. Enter `oracle.lite.web.WTGJdbcDriver` as the Driver Class. Choose `webtogo` from the Library list and click **OK**.

Figure 12–8 JDBC Driver Dialog



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:

Library:

Classpath:

URL:

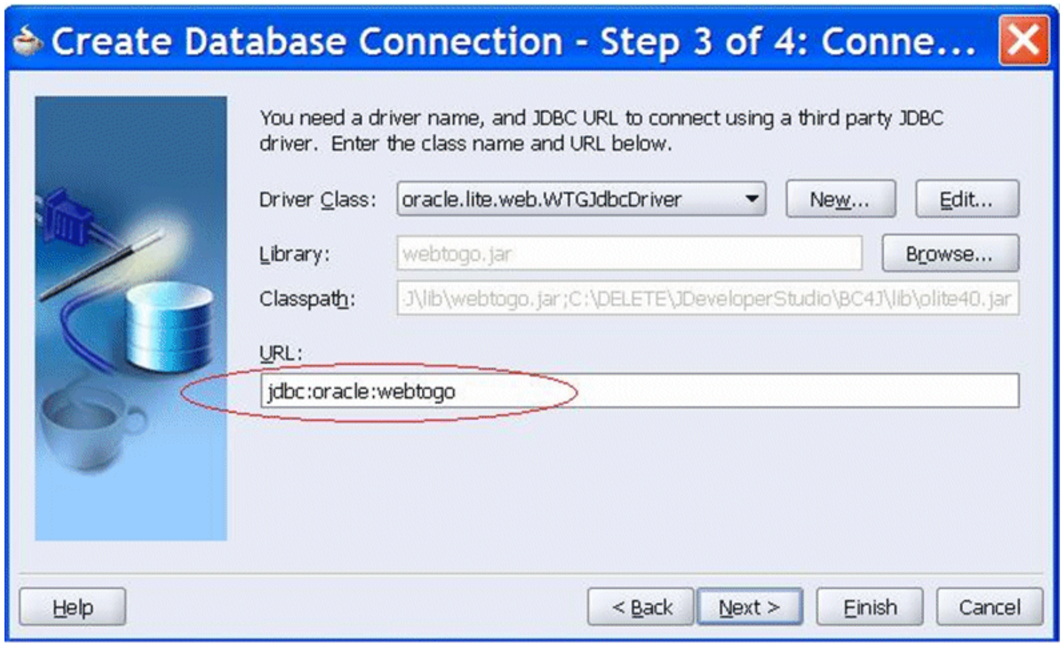
Driver Class: oracle.lite.web.WTGJdbcDriver

Library: webtogo.jar

Classpath: C:\10GLite\Mobile\Sdk\bin\webtogo.jar;C:\10GLite\Mobile\Sdk\bin\olite40.jar

b. Enter the following `jdbc:oracle:webtogo` URL and click **Next**.

Figure 12–9 Enter URL for Database Connection



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

Library: webtogo.jar

Classpath: J\lib\webtogo.jar;C:\DELETE\JDeveloperStudio\BC4J\lib\olite40.jar

URL: jdbc:oracle:webtogo

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.

12.3 Develop the ADF/BC4J Application

The following sections describe the steps to develop the ADF/BC4J application for Oracle Database Lite:

- Section 12.3.1, "Build the Data Model with ADF Business Components"
- Section 12.3.2, "Customize the Business Components Views"
- Section 12.3.3, "Create a Master-Detail JavaServer Faces Page"
- Section 12.3.4, "Run the JSF Page"
- Section 12.3.5, "Configure the ADF/BC4J Application for the Oracle Database Lite Environment"

12.3.1 Build 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 12.3.1.1, "Create a New Application and Projects"
- Section 12.3.1.2, "Create Business Components"

12.3.1.1 Create a New Application and Projects

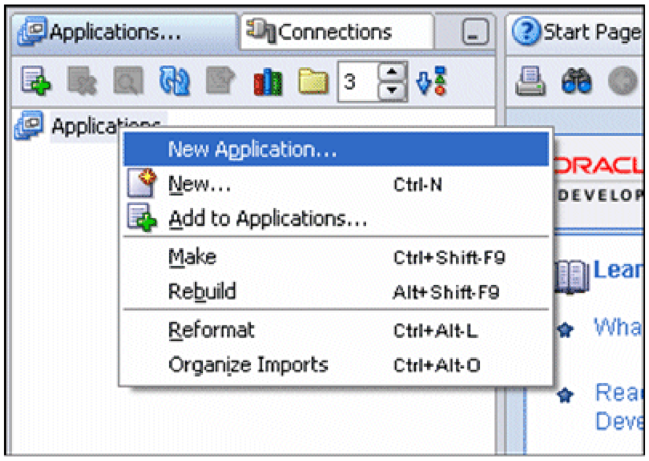
In JDeveloper, you work with projects contained in an application. The application is the highest point in the control structure.

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.

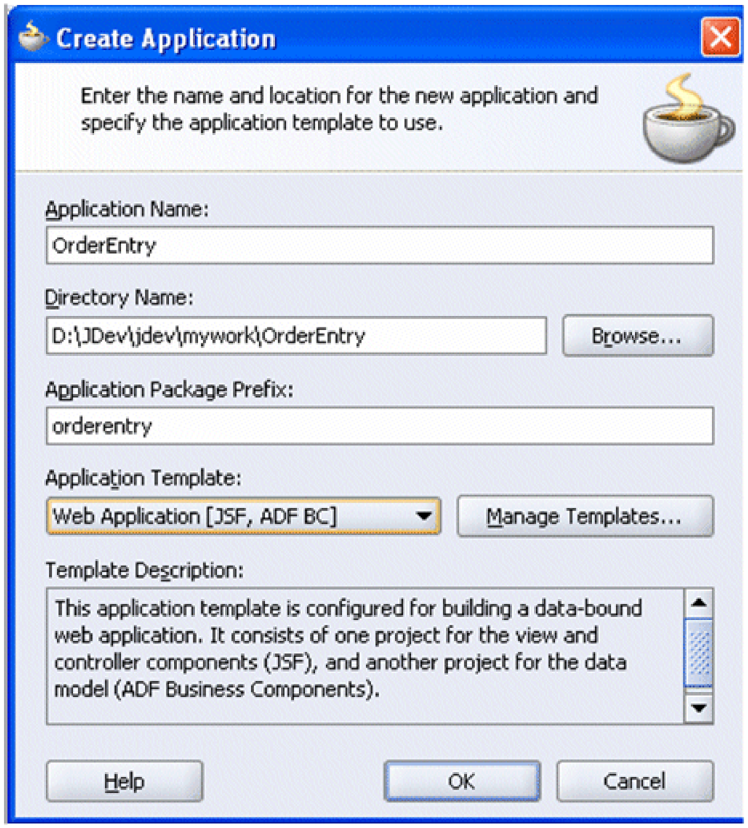
Figure 12–10 New Application



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 12–11 Create Application


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

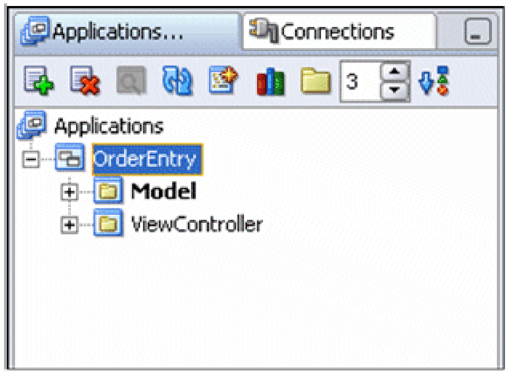
Application Template: Web Application [JSF, ADF BC]

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 12–12 OrderEntry in JDeveloper



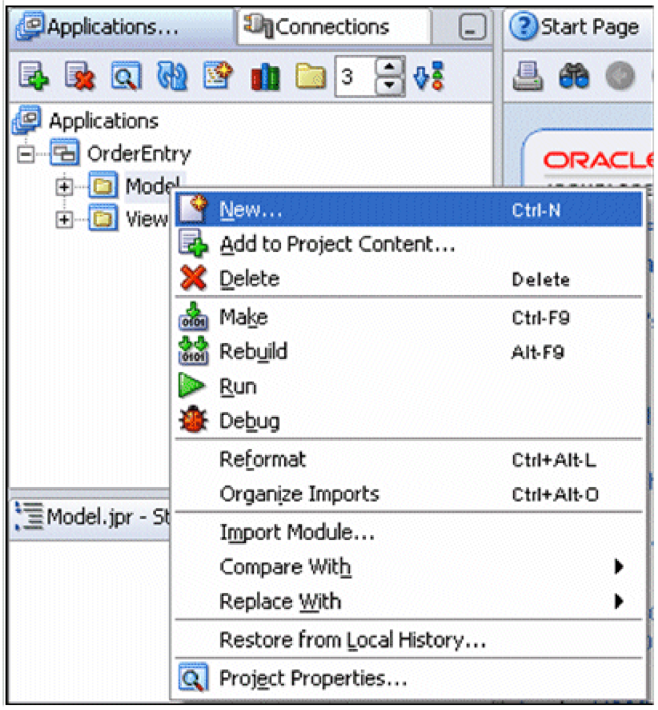
You now have an application and projects to contain and manage your application.

12.3.1.2 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.

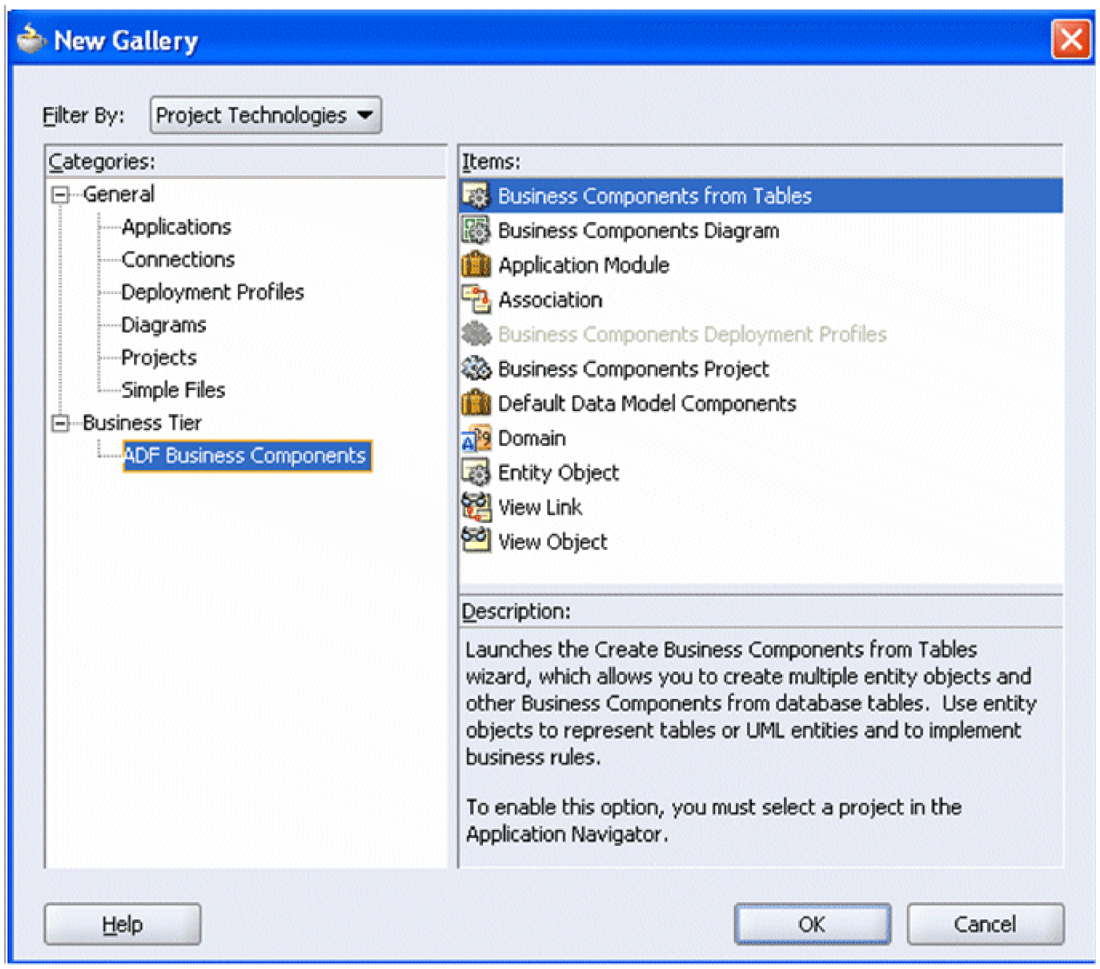
Figure 12–13 New Object in Model Project



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 12–14 Select ADF Business Components from Tables



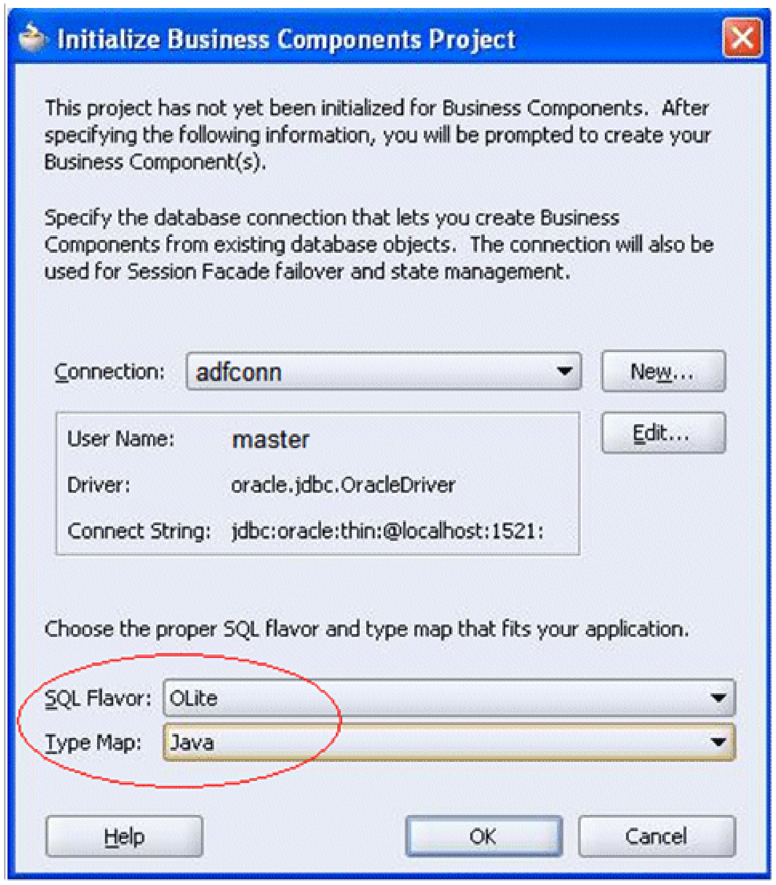
Launches the Create Business Components from Tables wizard, which allows you to create multiple entity objects and other Business Components from database tables. Use entity objects to represent tables or UML entities and to implement business rules.

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 12–15 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

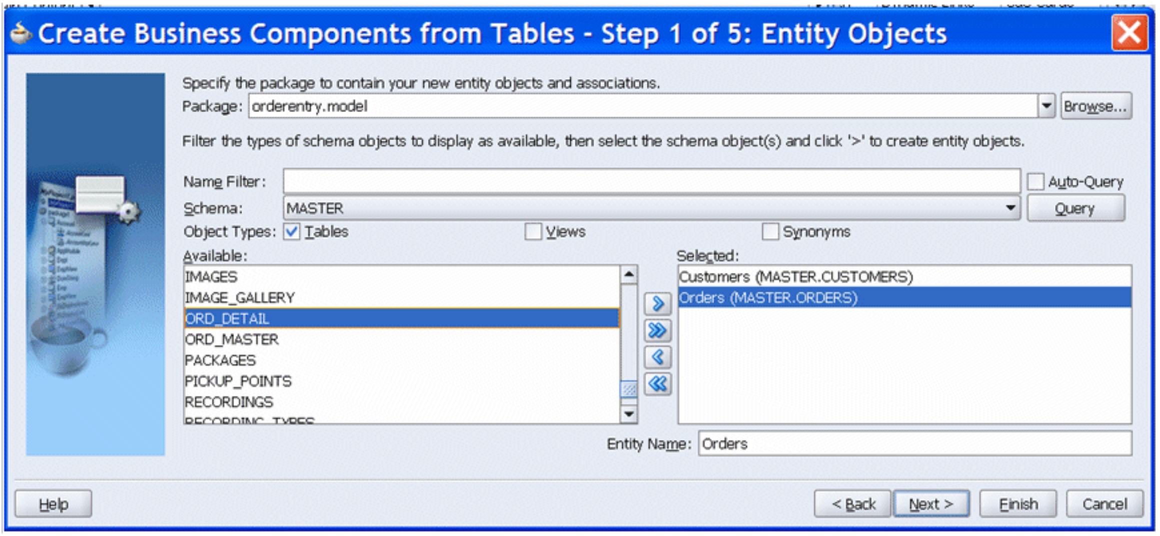
Type Map: Java

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 12–16 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:

Schema: MASTER

Object Types: [✓ Tables] Views Synonyms

Available:

IMAGES
IMAGE_GALLERY
ORD_DETAIL
ORD_MASTER
PACKAGES
PICKUP_POINTS
RECORDINGS
RECORDING_TYPES

Selected:

Customers (MASTER.CUSTOMERS)
Orders (MASTER.ORDERS)

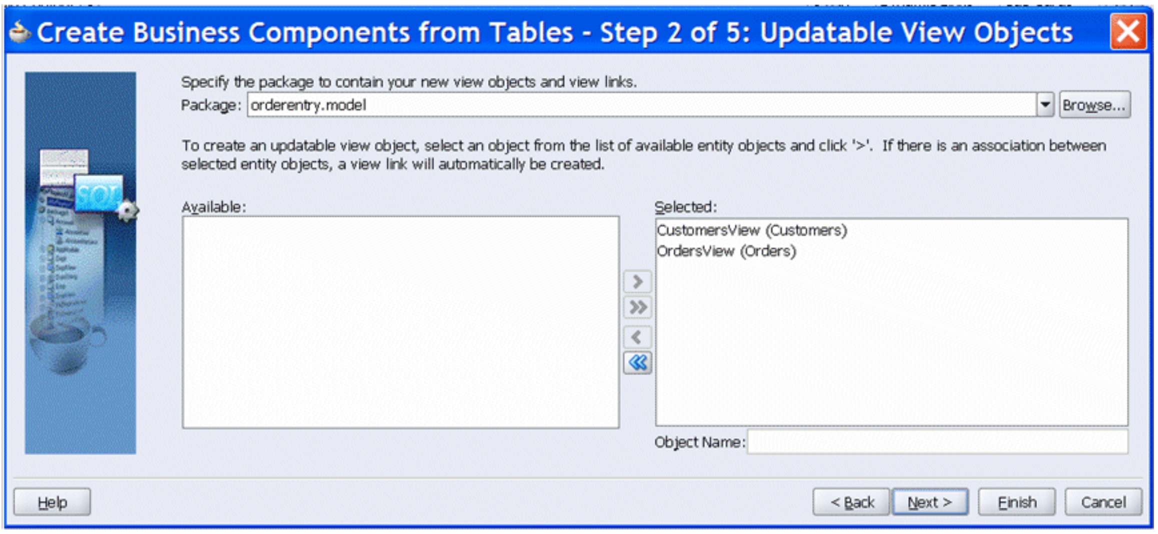
Entity Name: Orders

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 12–17 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:

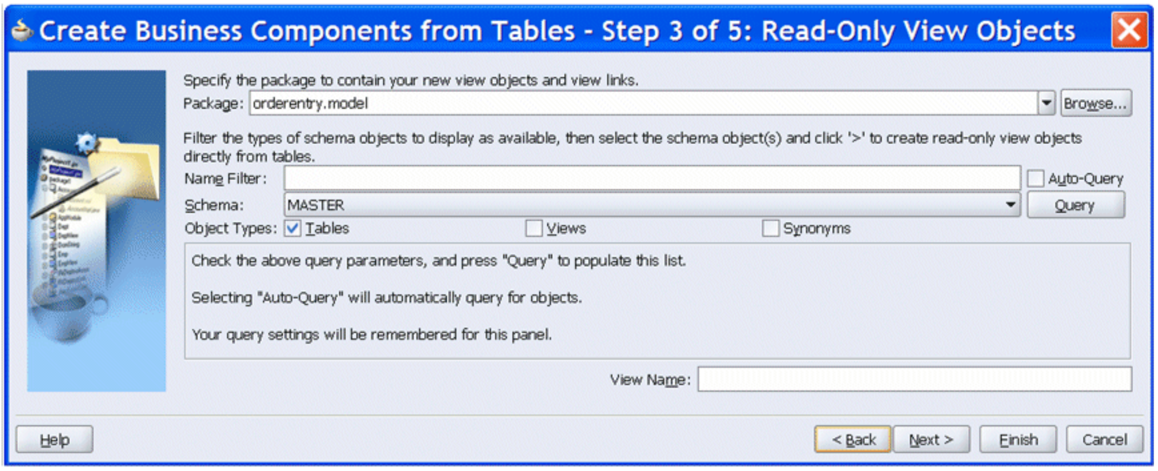
Selected:
CustomersView (Customers)
OrdersView (Orders)

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 12–18 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.

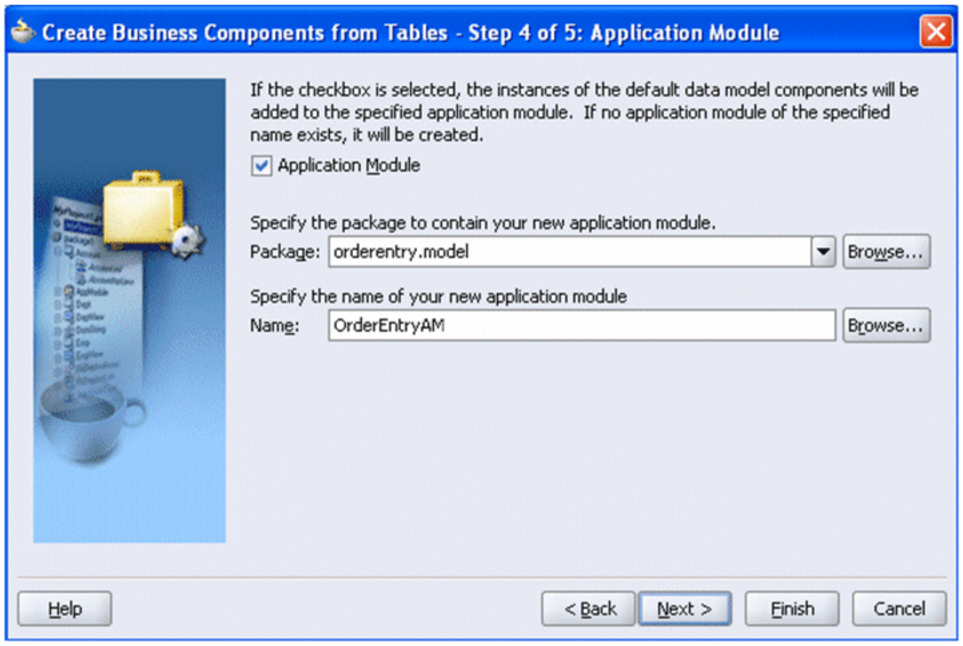
Your query settings will be remembered for this panel.

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 12–19 Application Module


Create Business Components from Tables - Step 4 of 5: Application Module
Help
< Back
Next >
Finish
Cancel
Browse...
Browse...

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

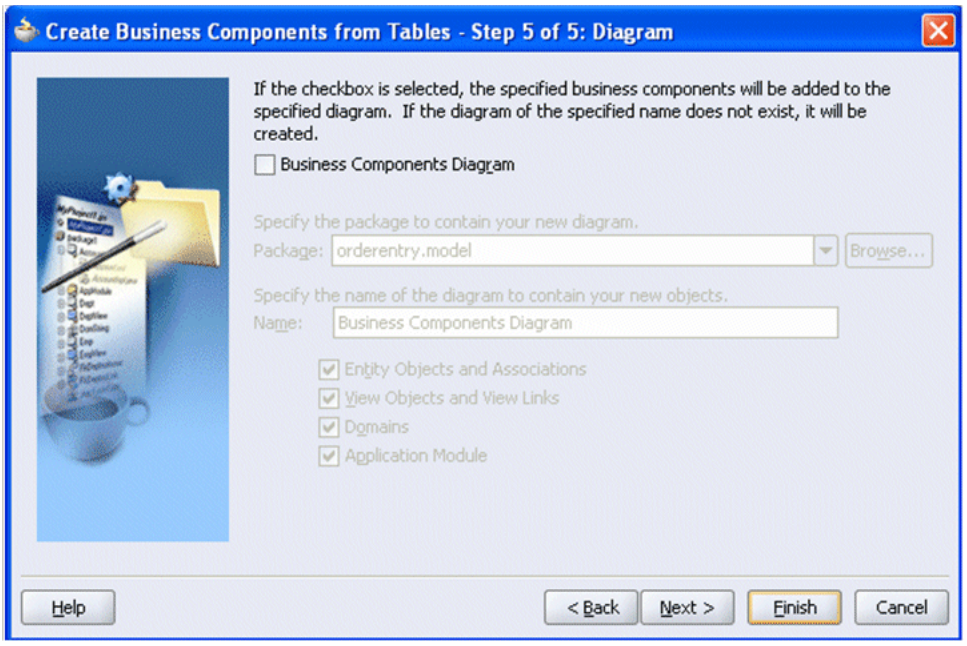
Name: OrderEntryAM

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 12–20 Diagram


Create Business Components from Tables - Step 5 of 5: Diagram
Help
< Back
Next >
Finish
Cancel
Browse...

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

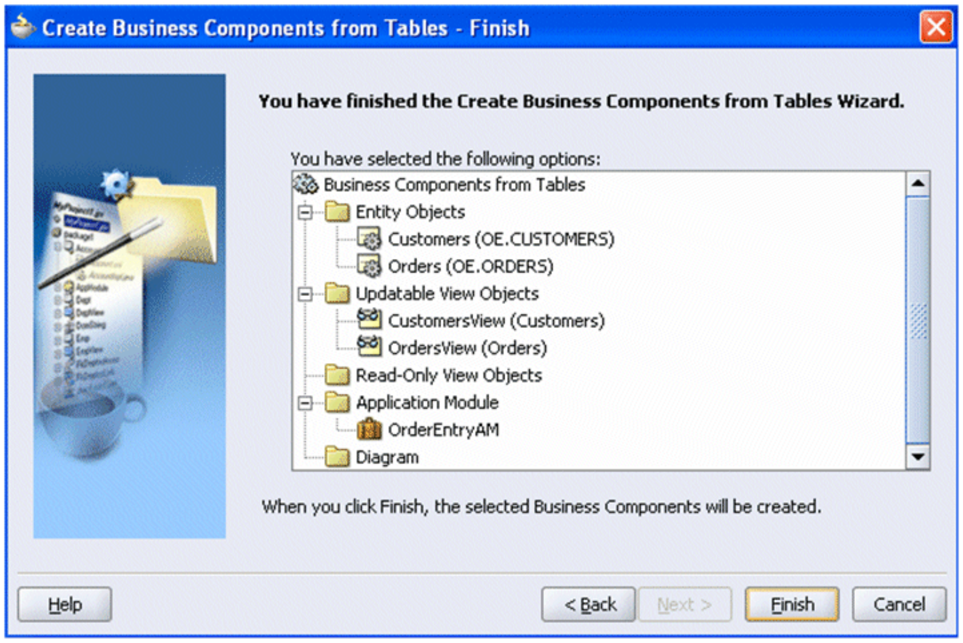
[✓] Domains

[✓] 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 12–21 Finish



Create Business Components from Tables - Finish

You have finished the Create Business Components from Tables Wizard.

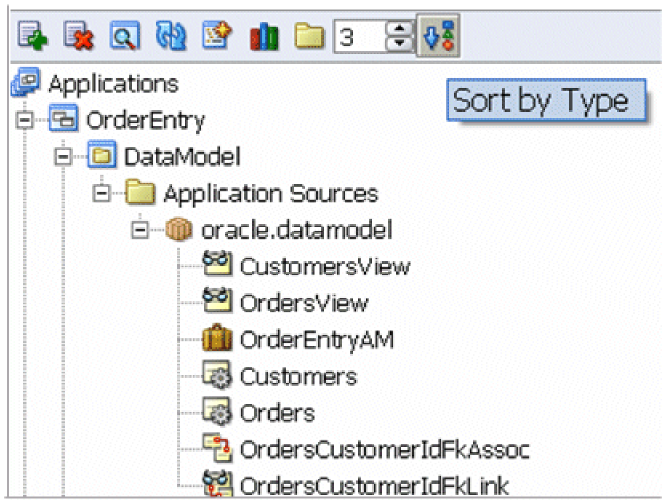
You have selected the following options:

- 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 12–22 Sort by Type



- Applications
  - OrderEntry
    - DataModel
      - Application Sources
        - oracle.datamodel
          - CustomersView
          - OrdersView
          - OrderEntryAM
          - Customers
          - Orders
          - OrdersCustomerIdFkAssoc
          - OrdersCustomerIdFkLink

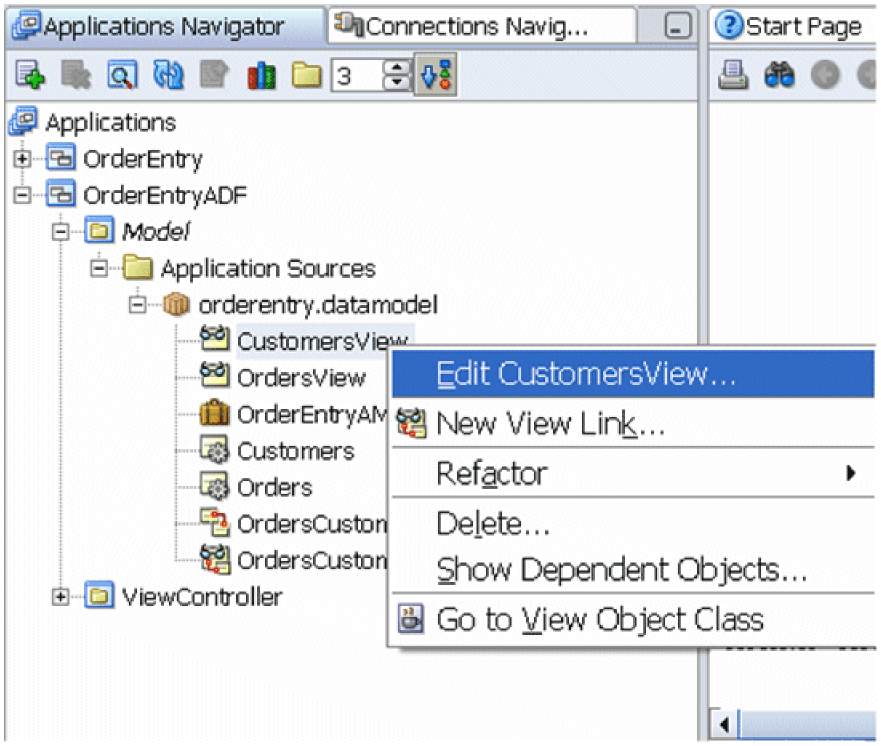
12.3.2 Customize the Business Components Views

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.

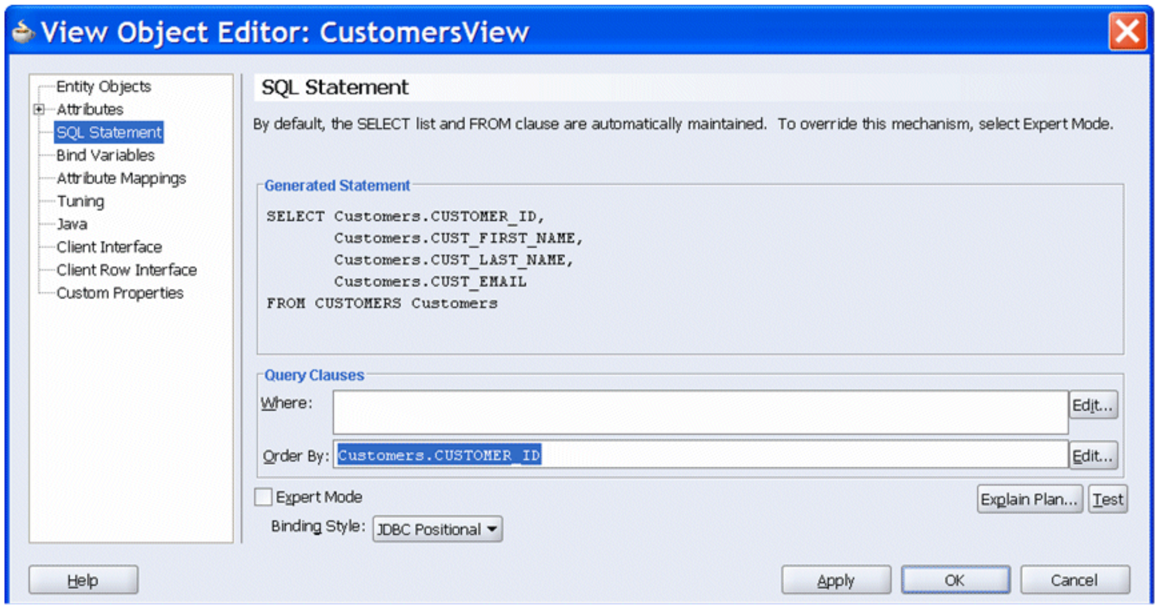
1. In the Applications Navigator, right-click the CustomersView node and select **Edit CustomersView** from the context menu.

Figure 12–23 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 12–24 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.

12.3.3 Create a Master-Detail JavaServer Faces Page

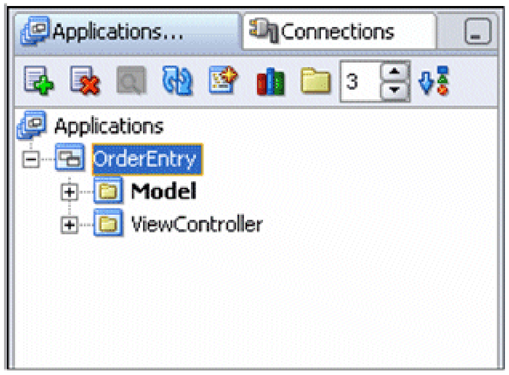
Conforming to the JSF standards, ADF Faces lets you concentrate on the application and layout rather than markup language and tags. Due to the integration of ADF Faces and ADF Business Components, you can easily change the default field labels for the user interface from within ADF Business Components.

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.

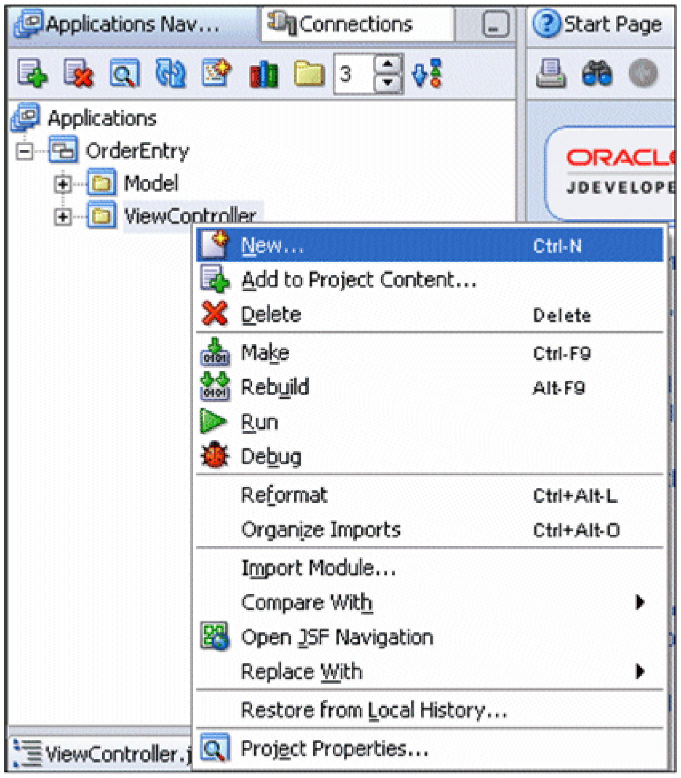
   Collapse the Model node so that the Applications Navigator appear as follows:

Figure 12–25 OrderEntry in JDeveloper



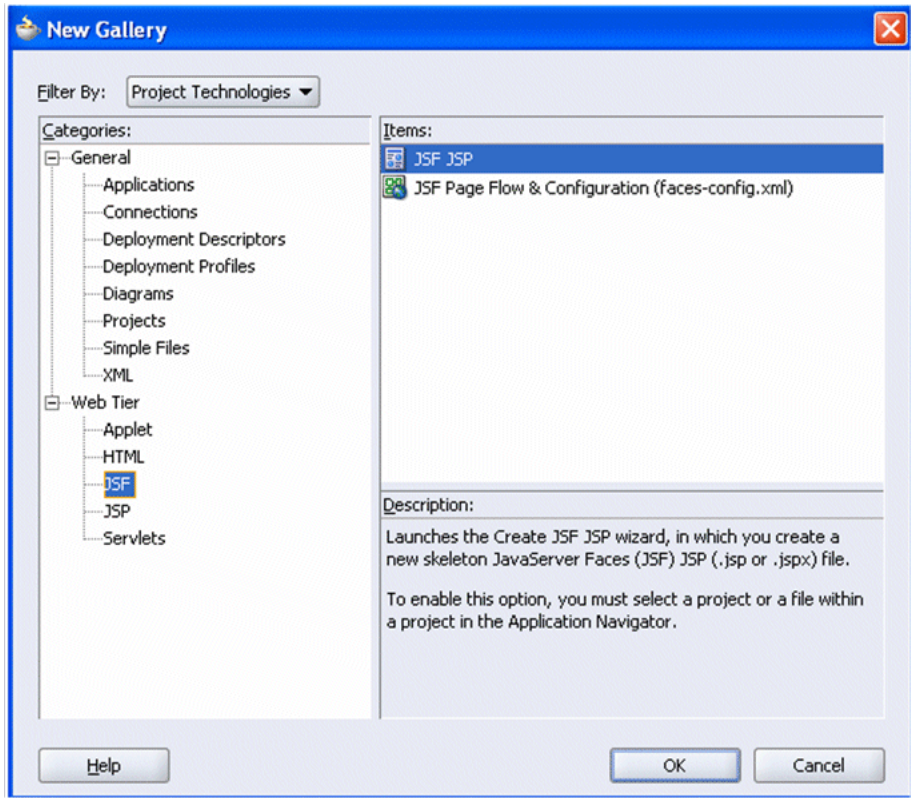
2. Create a new JSF by right-clicking ViewController in the Applications Navigator and selecting **New** from the context menu.

Figure 12–26 New Object Under ViewController



3. Select **JSF JSP** from the JSF Category.

Figure 12–27 New JSF JSP



Launches the Create JSF JSP wizard, in which you create a new skeleton JavaServer Faces (JSF) JSP (.jsp or .jspx) file.

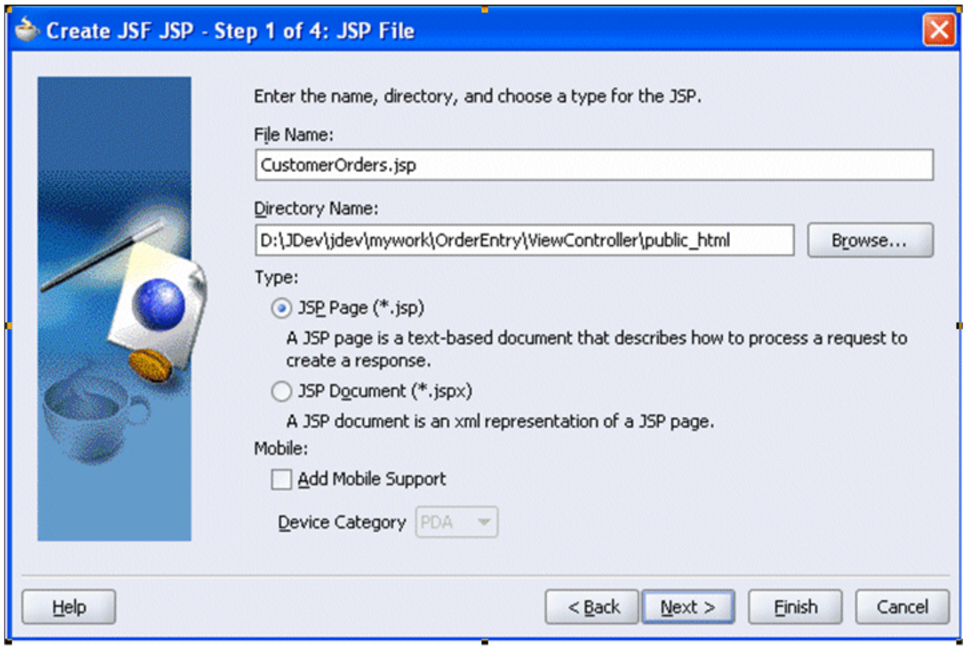
To enable this option, you must select a project or a file within a project in the Application Navigator.

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 12–28 Step 1 of Creating JSP



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.

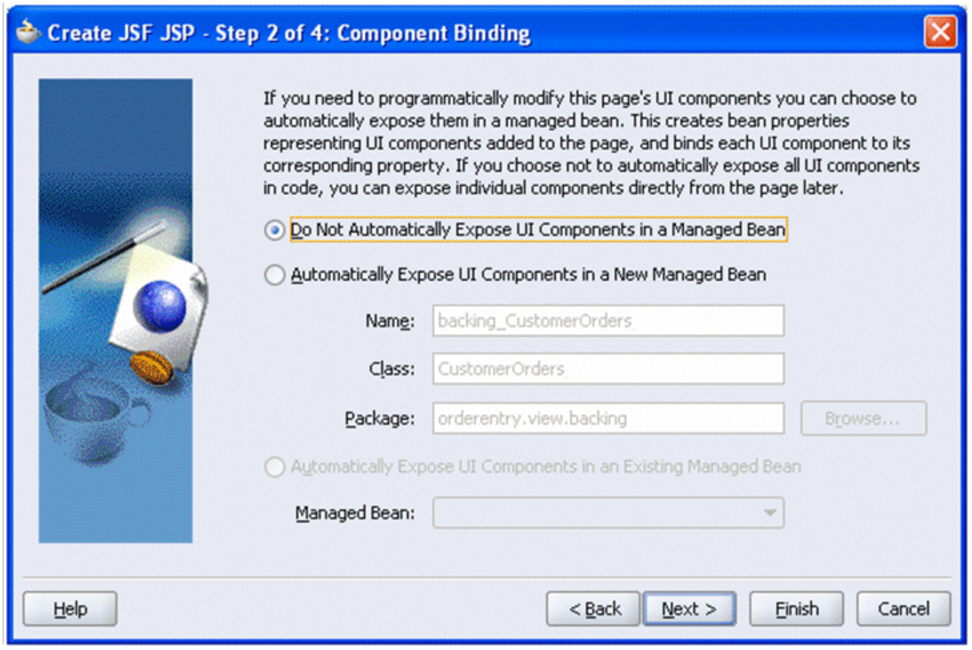
Mobile:

[ ] 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 12–29 Step 2 of Creating JSP



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:

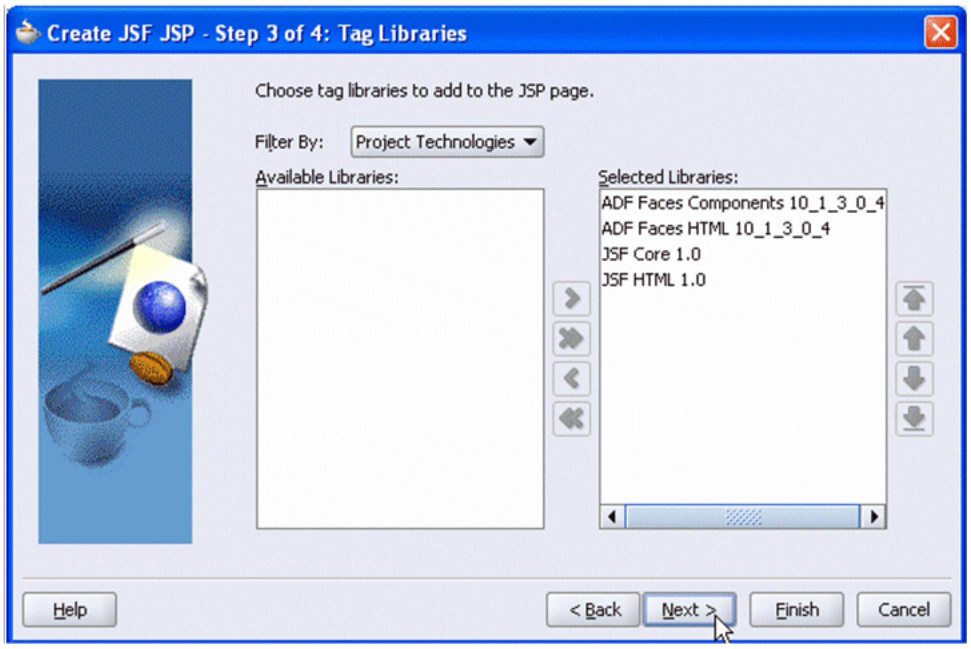
d. Select **libraries** in the Available Libraries window, and use the Add button to move them into the Selected Libraries section, as needed. Make sure the following libraries appear in Selected Libraries:

– JSF Core 1.0

– JSF HTML 1.0

- ADF Faces Components
- ADF Faces HTML

Figure 12–30 Step 3 of Creating JSP



Choose tag libraries to add to the JSP page.

Filter By: Project Technologies

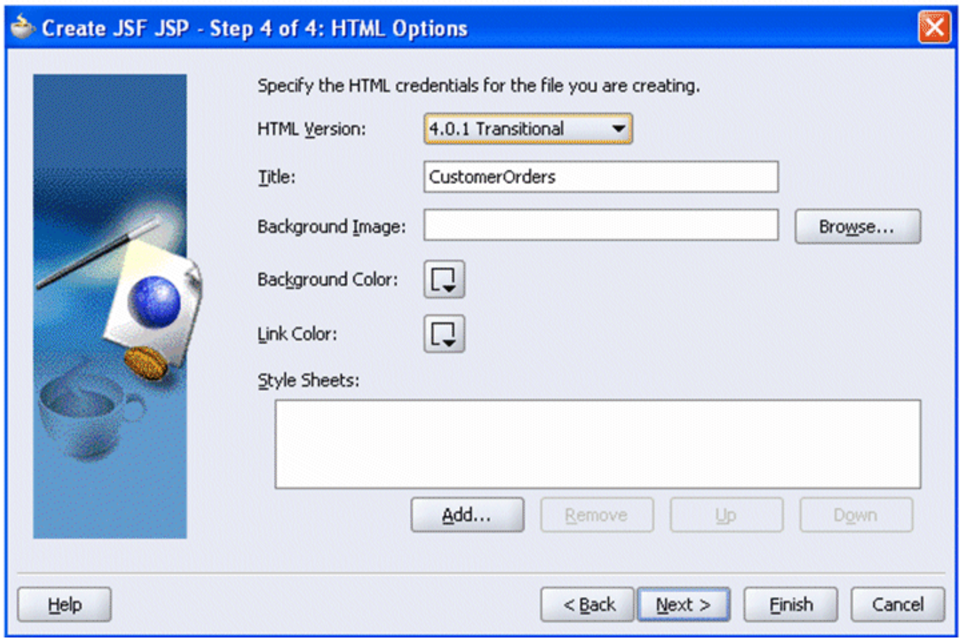
Available Libraries:

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 12–31 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:

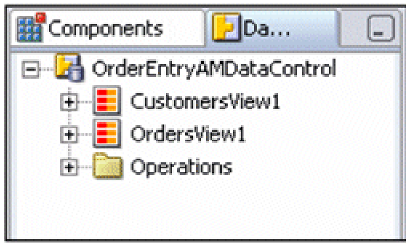
Style Sheets:

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.

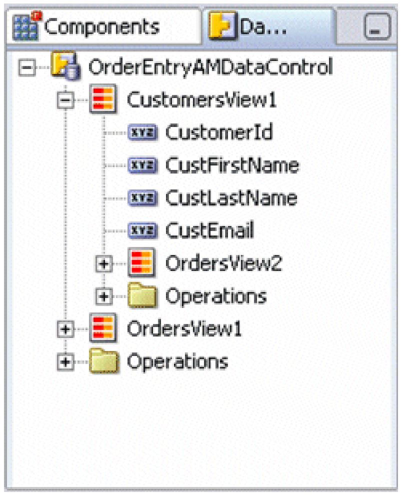
1. Expand OrderEntryDataAMControl in the Data Control palette.

Figure 12–32 Data Control Palette



2. Expand `CustomersView1`.

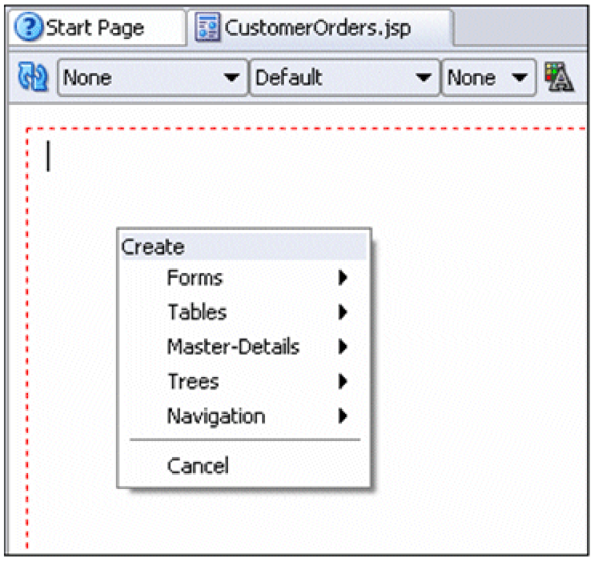
Figure 12–33 Expand CustomersView1



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.

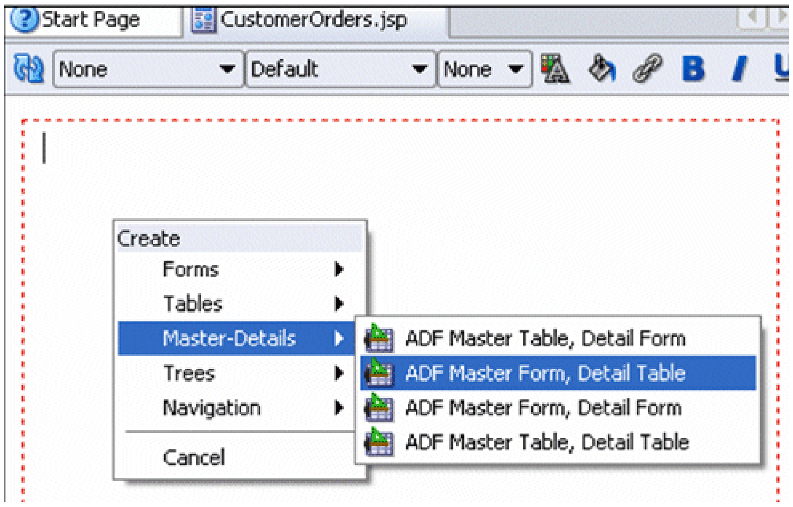
3. Drag `OrdersView2` to the visual editor. JDeveloper opens a context menu with the available options for that data control.

Figure 12–34 Visual Editor



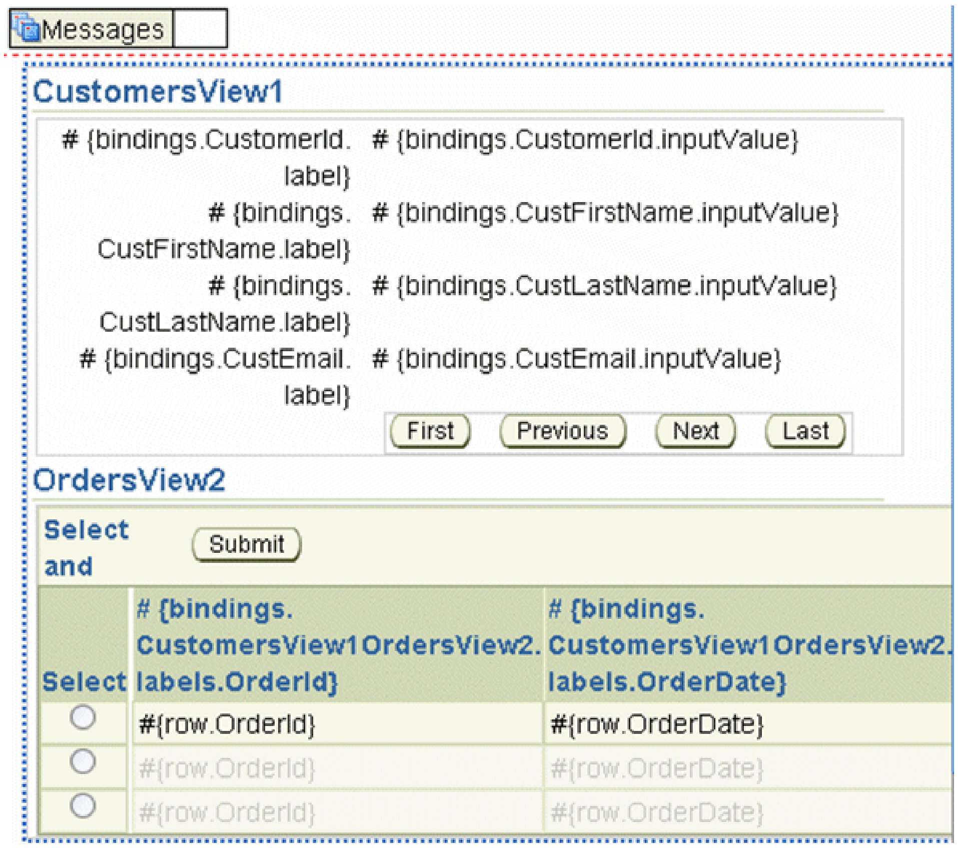
4. Place your cursor over the Master-Details option, and then select **ADF Master Form, Detail Table**.

Figure 12–35 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 12–36 Add ADF Master-Detail to JSP



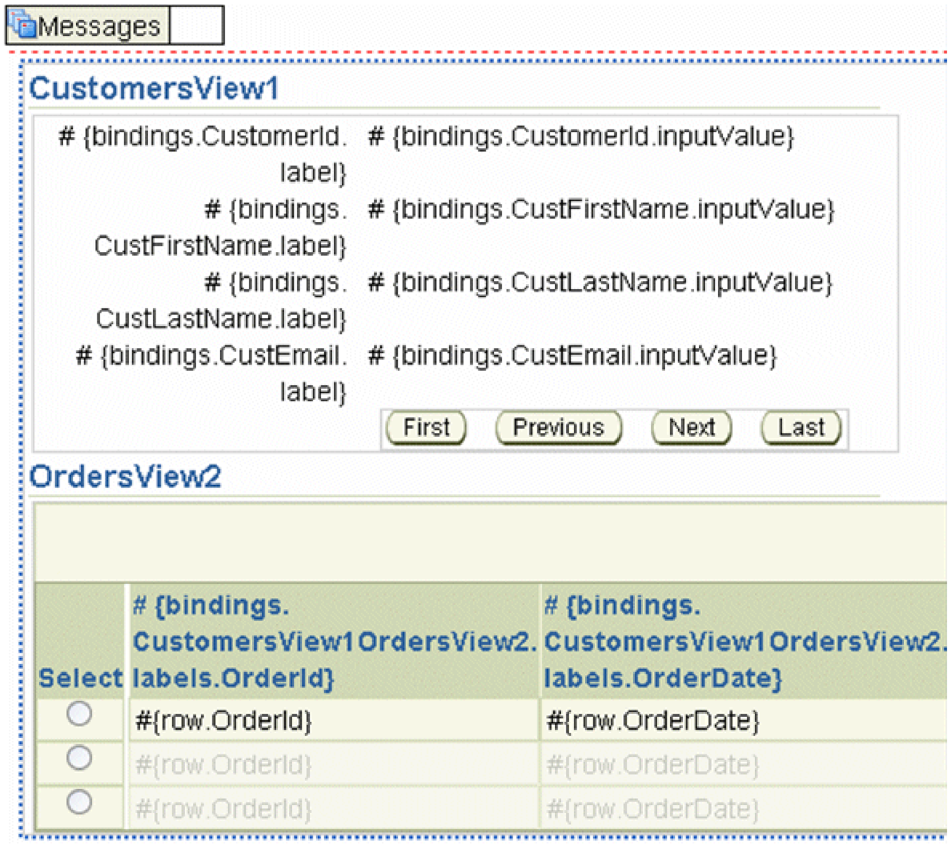
# {bindings.CustomerId.label}	# {bindings.CustomerId.inputValue}
# {bindings.CustFirstName.label}	# {bindings.CustFirstName.inputValue}
# {bindings.CustLastName.label}	# {bindings.CustLastName.inputValue}
# {bindings.CustEmail.label}	# {bindings.CustEmail.inputValue}

Select and

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 12–37 Submit Components



#{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}

You now have a complete JSF that is databound to your ADF BC business services.

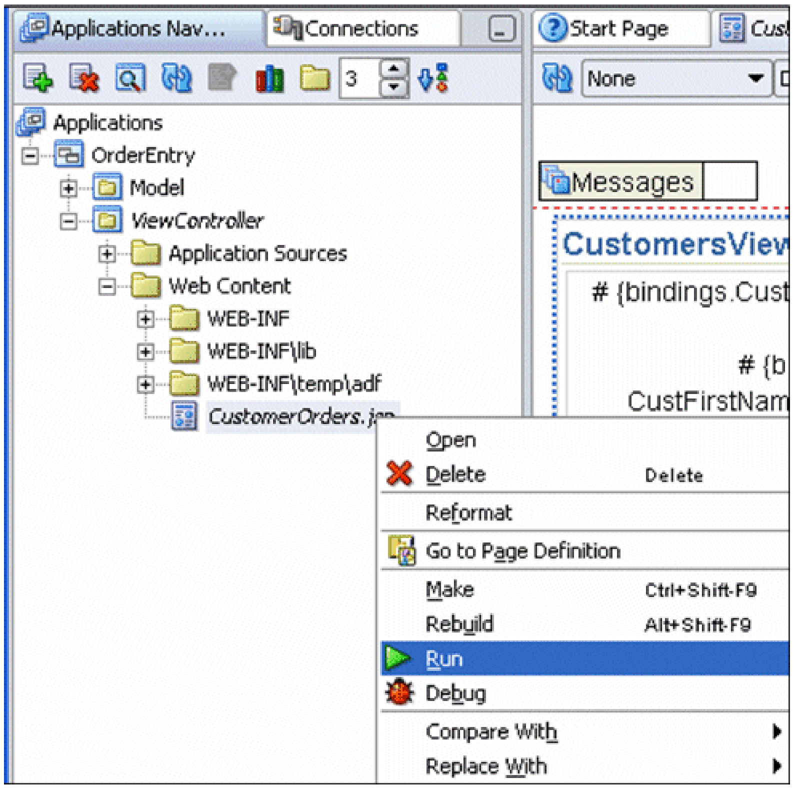
12.3.4 Run the JSF Page

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.

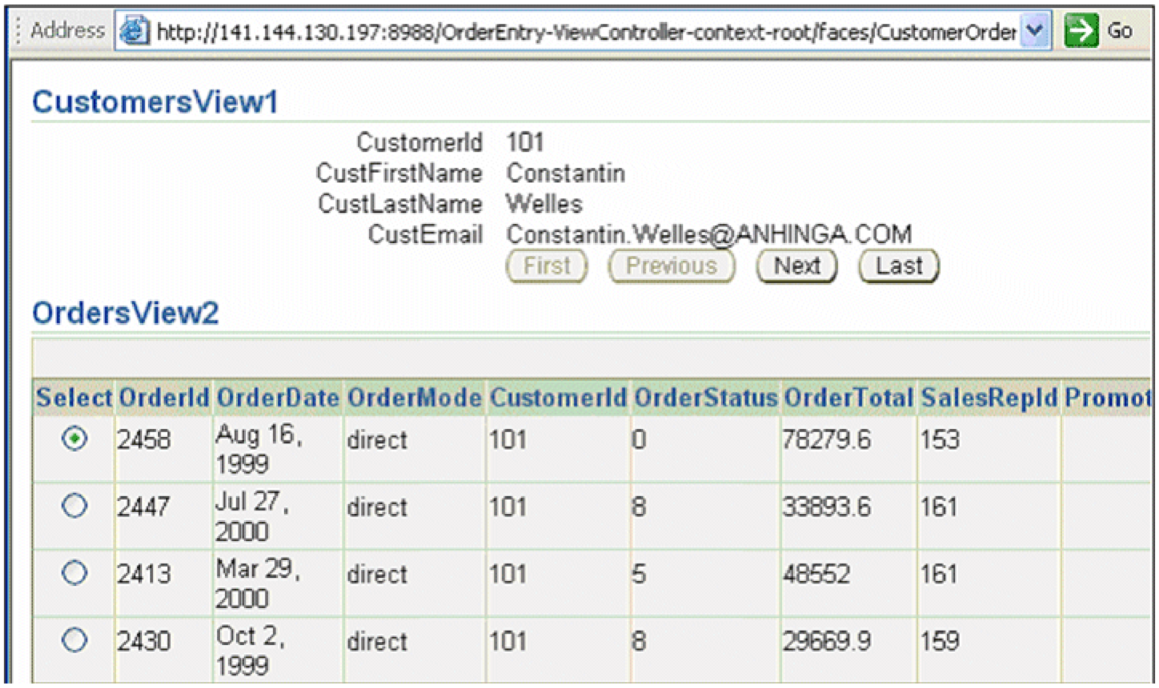
1. To test the page, right-click `CustomerOrders.jsp` in the Applications Navigator and select **Run** from the context menu. Alternatively, you can right-click inside the visual editor and select **Run** from that context menu.

Figure 12–38 Test the Page



2. After execution, the results page should be as follows:

Figure 12–39 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**.

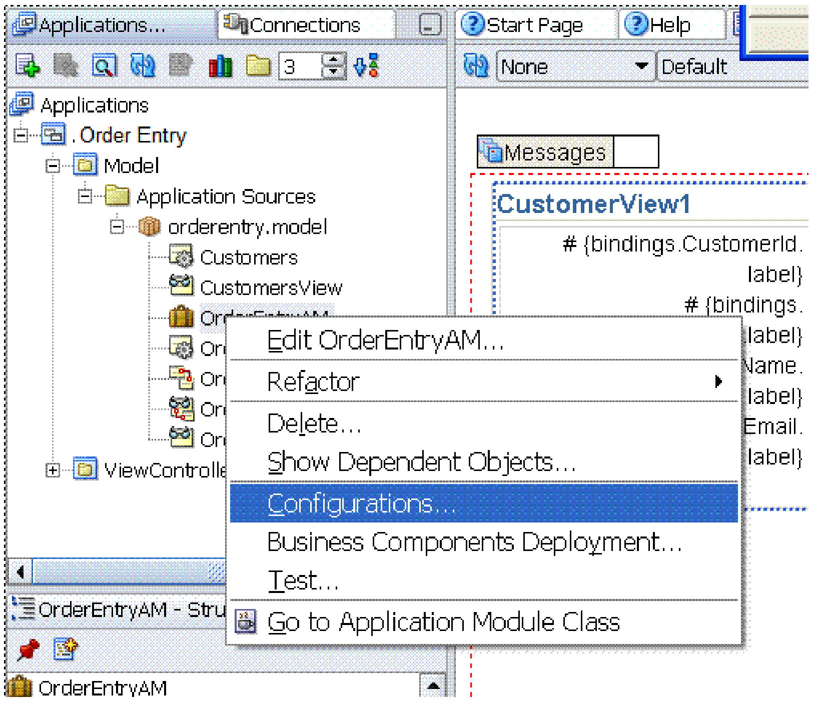
12.3.5 Configure the ADF/BC4J Application for the Oracle Database Lite Environment

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**.

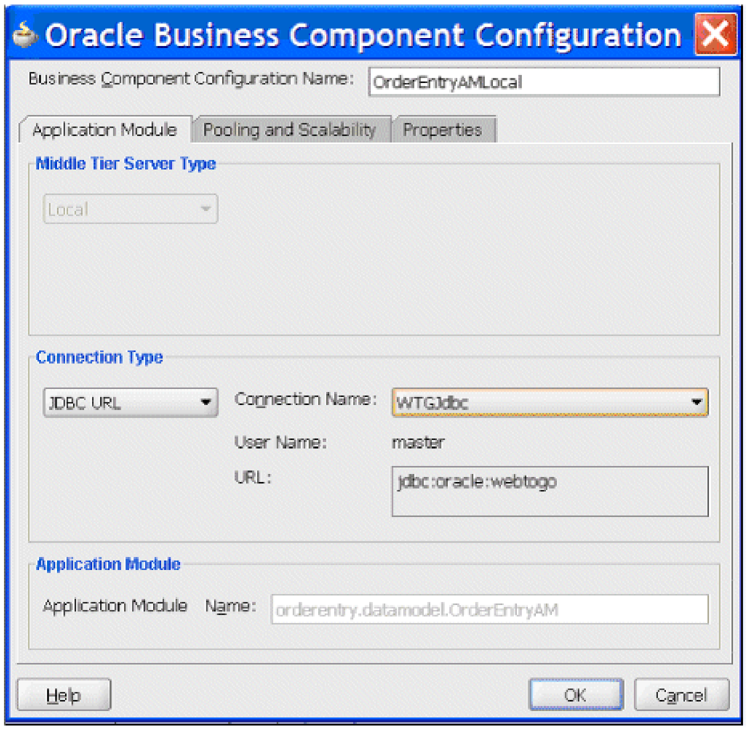
Figure 12–40 Selecting Configurations in JDeveloper



2. Select **JDBCName** and edit the connection. The Oracle Business Component Configuration window is displayed.

3. Select **WTGJdbc** for Connection Name and click **OK** twice.

Figure 12–41 Business Component Configuration Window



4. Save your project.

The ADF/BC4J application has been configured to use Oracle Database Lite connection

12.3.6 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.

12.4 Package the ADF/BC4J Application

In order to package the ADF/BC4J application, you must perform the following:

- Section 12.4.1, "Include the ADF Runtime Libraries with the ADF/BC4J Application"
- Section 12.4.2, "Package the Application from the Packaging Wizard"

12.4.1 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).

12.4.2 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 12–2.

Table 12–2 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 12–3 in the "Connect to Database" dialog:

Table 12–3 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.

12.5 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.

12.6 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 12.7, "Run the ADF/BC4J Application on the Mobile Client for Oracle Lite WEB OC4J".

12.7 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.

13 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.Net 2003 or 2005. If you use Visual Studio.Net 2005, you must install the ODBC 3.5 driver. See Section 5.1.3, "ODBC" for details.

- Section 13.1, "Plan the Mobile Application Demo for Win32"
- Section 13.2, "Description of Tasks for Win32 Demo"
- Section 13.3, "Administer the Application"
- Section 13.4, "Execute the Application on the Mobile Client for Web-to-Go"

13.1 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 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.

13.2 Description of Tasks for Win32 Demo

The following sections describe the command sequence for successfully creating the Win32 demo:

1. Section 13.2.1, "Create TASK Table on the Server Database"
2. Section 13.2.2, "Create Publication for Application"

3. Section 13.2.3, "Package the Application Using the Packaging Wizard"

13.2.1 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
```

13.2.2 Create Publication for Application

As described fully in Chapter 6, "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 6, "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 13.2.2.1, "Create a Project"
- Section 13.2.2.2, "Create Publication Item"
- Section 13.2.2.3, "Create Publication"

13.2.2.1 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 6.2, "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 `MOBILEADMIN/<mobileadmin_password>`, 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**.

13.2.2.2 Create Publication Item

For this project, you need to create the `taskpi` publication item.

Note: For more information, see Section 6.4, "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**.

13.2.2.3 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: taskpi
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**.

13.2.3 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:

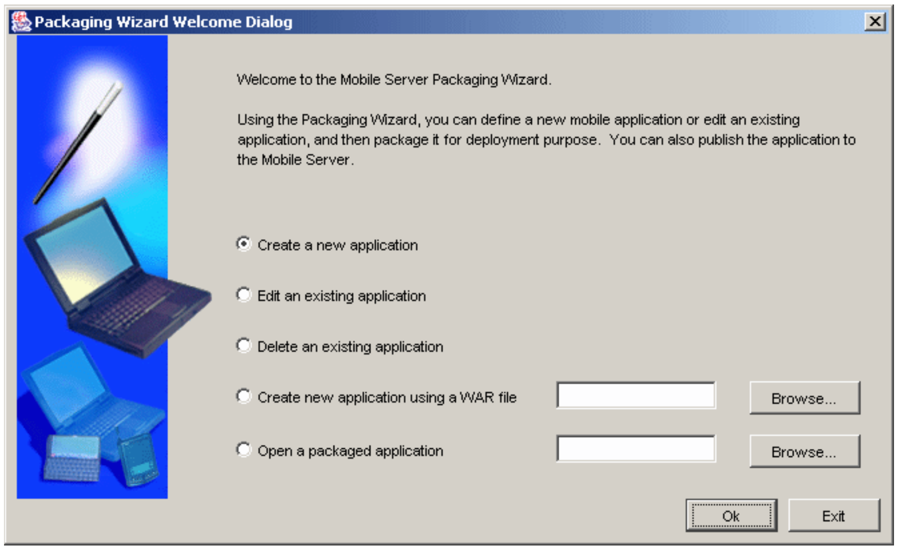
1. Start the Packaging Wizard, as follows:

```
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 13–1.

Note: Deleting an existing application merely deletes the application from the XML file and does not remove the application from the Mobile Server.

Figure 13–1 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

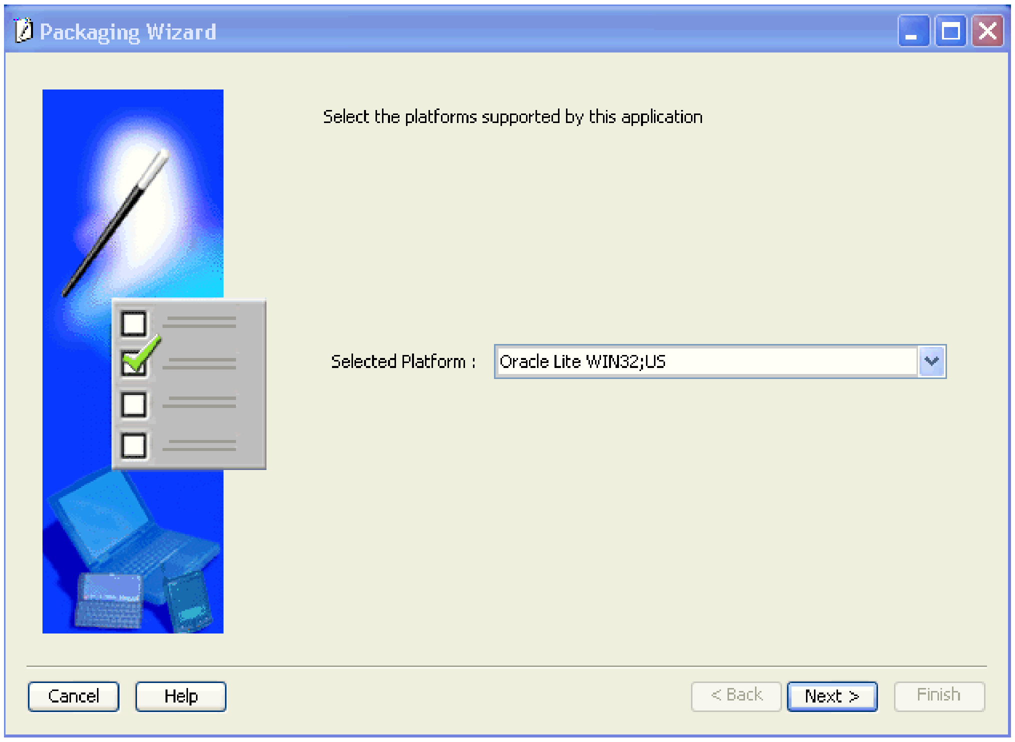
Delete an existing application

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 13–2 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 13–2 Selecting a Platform

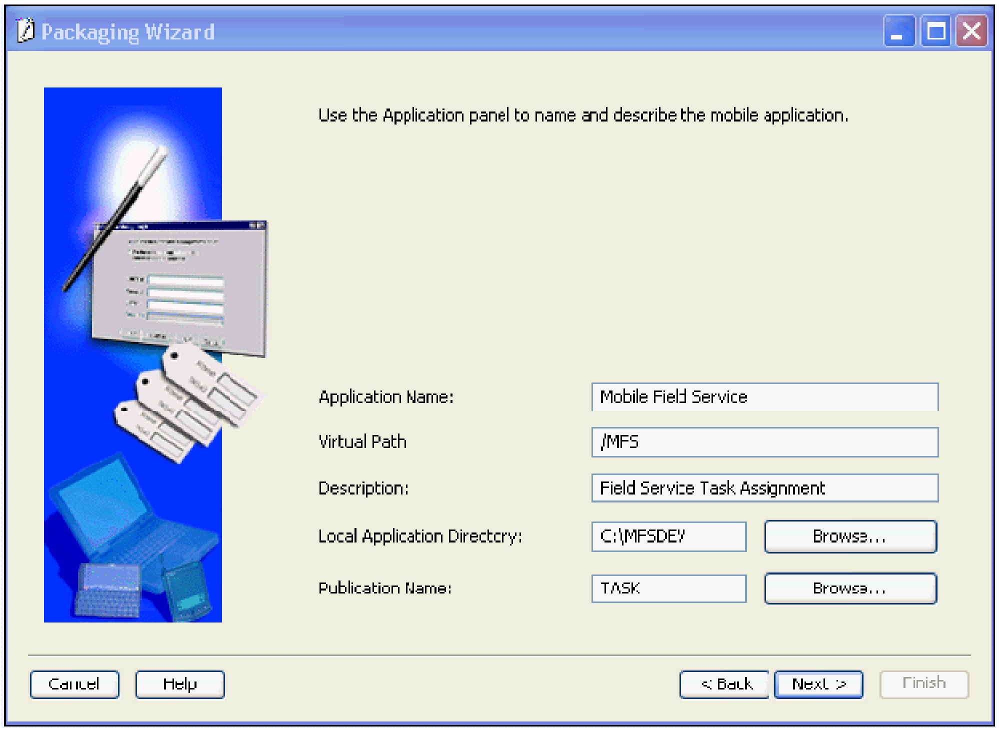


Select the platforms supported by this application

Selected Platform : Oracle Lite WIN32;US

4. As Figure 13–3 displays, the Application panel appears. Use the Application panel to modify "Mobile Field Service" application settings. As Table 13–1 describes, enter the specified values in the corresponding fields.

Figure 13–3 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

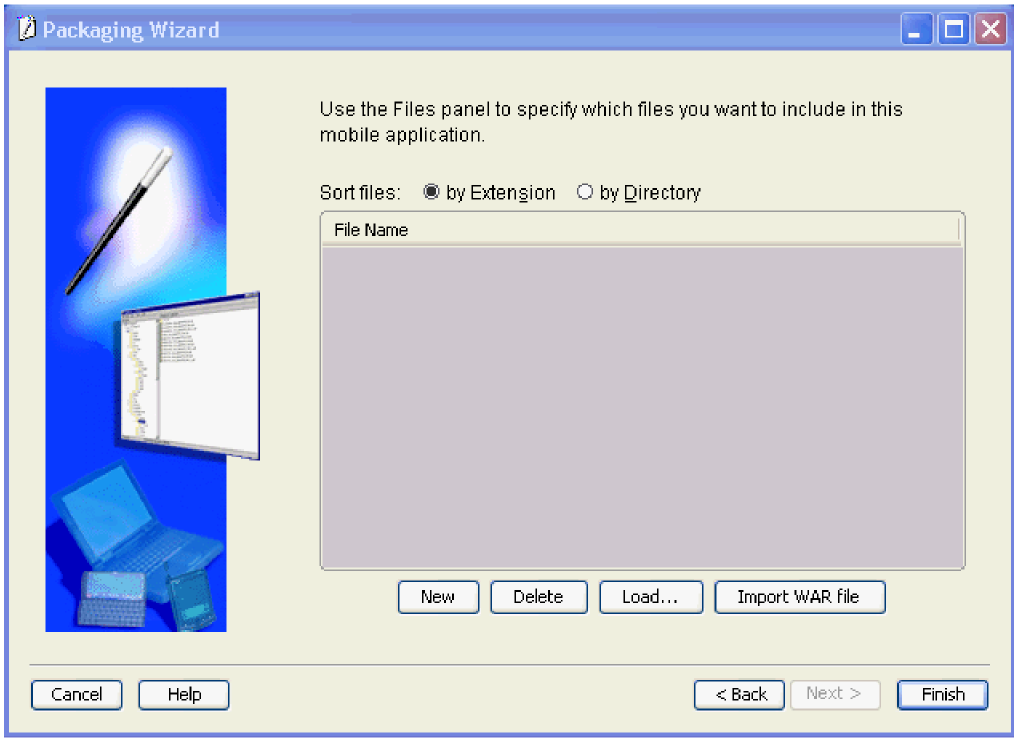
Table 13–1 The Task Application Values

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 13–4 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 13–4 Uploading Application Files

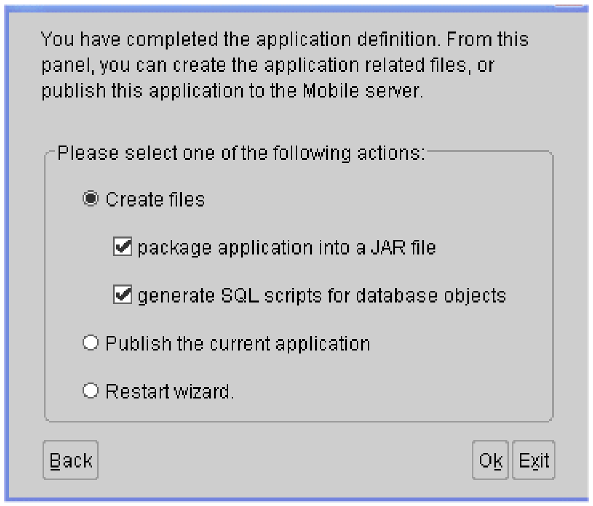


Use the Files panel to specify which files you want to include in this mobile application.

Sort files: [✓ by Extension] by Directory

6. Click **Next** till you arrive at the Application Definition Completed Dialog as shown in Figure 13–5.

Figure 13–5 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

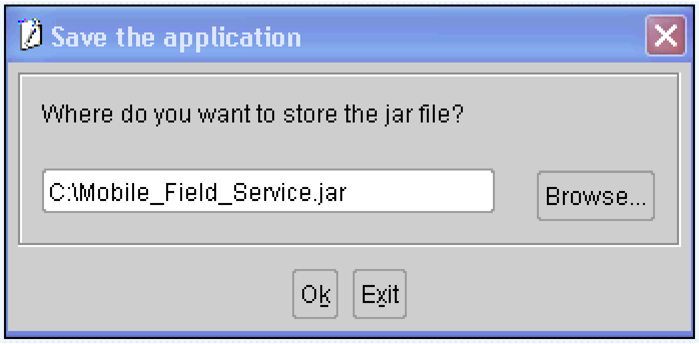
Restart wizard.

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 13–6 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 13–2 describes, enter the specified values.

Note: The Mobile Server must be up for successful publishing.

Table 13–2 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.

13.3 Administer the Application

This section describes how to administer the application that you created and deployed through the following tasks.

- Section 13.3.1, "Start the Mobile Server and the Mobile Manager"

- Section 13.3.2, "Using the Mobile Manager to Create New Users for the Task Application"
- Section 13.3.3, "Setting Application Properties"
- Section 13.3.4, "Granting User Access to the Application"
- Section 13.3.5, "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*.

13.3.1 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
   ```

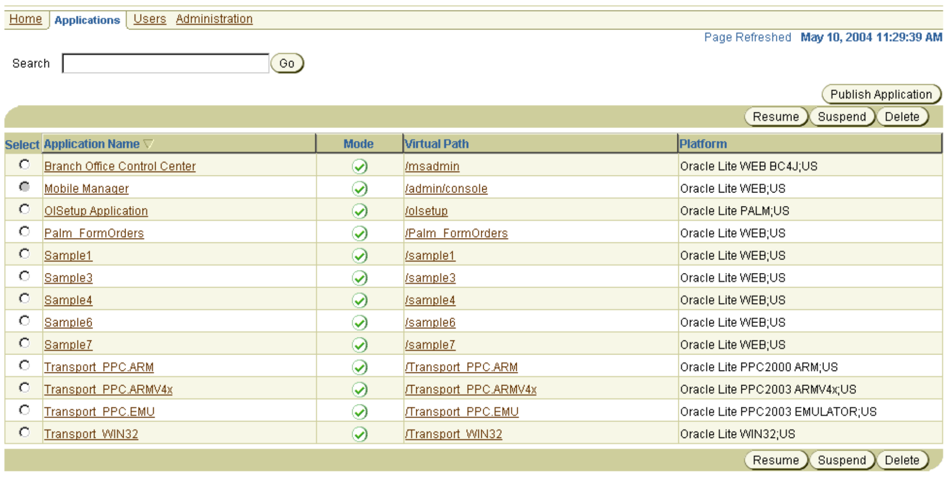
2. To start the Mobile Server for the first time and subsequent occasions, execute the `runmobileserver` command.
3. Start your Web browser and connect to the Mobile Server by enter the following URL:

   ```
   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 13–7 displays, the Applications page appears. Locate the Task application, which shows all applications that are published.

Figure 13–7 Applications Page



Page Refreshed May 10, 2004 11:29:39 AM

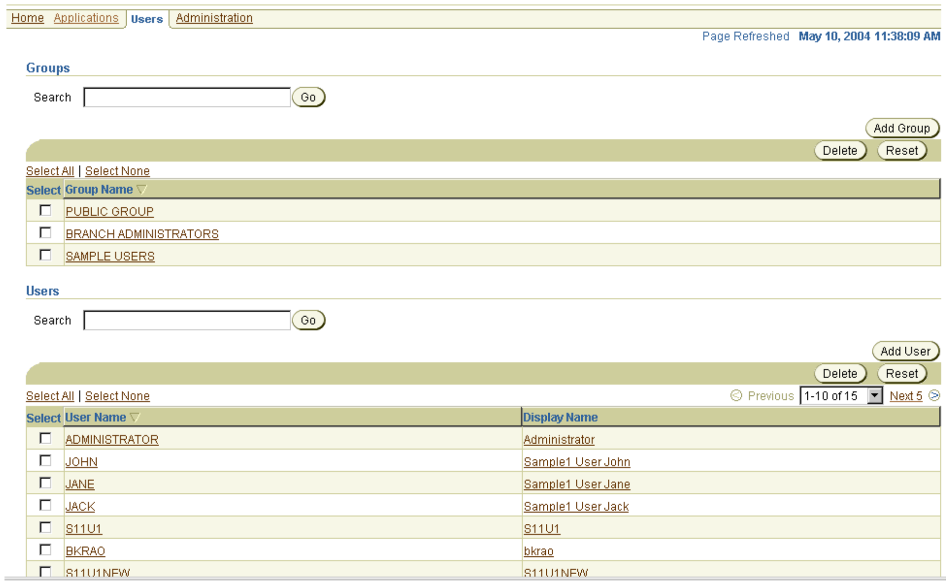
Select	Application Name	Mode	Virtual Path	Platform
○	Branch Office Control Center	✓	/msadmin	Oracle Lite WEB BC4J;US
○	Mobile Manager	✓	/admin/console	Oracle Lite WEB;US
○	OlSetup Application	✓	/olsetup	Oracle Lite PALM;US
○	Palm_FormOrders	✓	/Palm_FormOrders	Oracle Lite WEB;US
○	Sample1	✓	/sample1	Oracle 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

13.3.2 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 13–8 displays, the Users page appears.

Figure 13–8 Users Page



Page Refreshed May 10, 2004 11:38:09 AM

Groups

Search [ ] Go

Select	Group Name
☐	PUBLIC GROUP
☐	BRANCH ADMINISTRATORS
☐	SAMPLE USERS

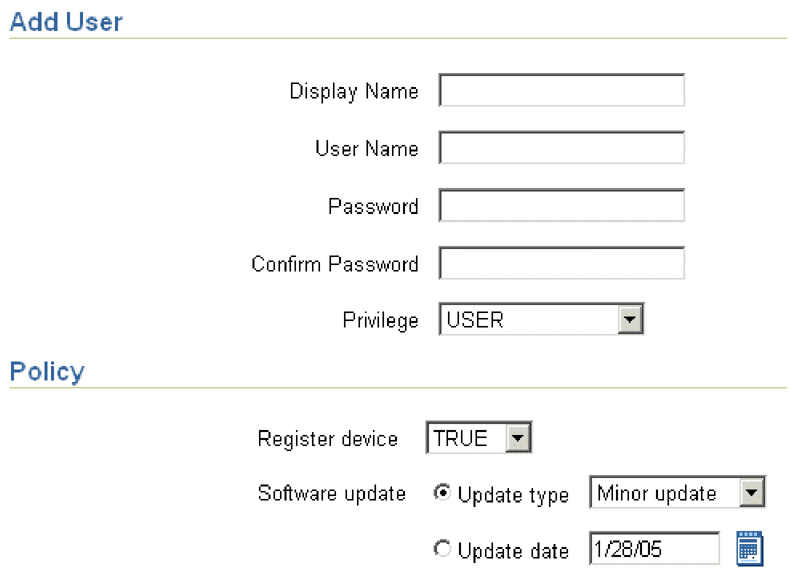
Users

Search [ ] Go

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 13–9 displays, the Add User page appears.

Figure 13–9 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 13–3, enter the following information in the Add User page and click **Save**.

Table 13–3 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.

13.3.3 Setting Application Properties

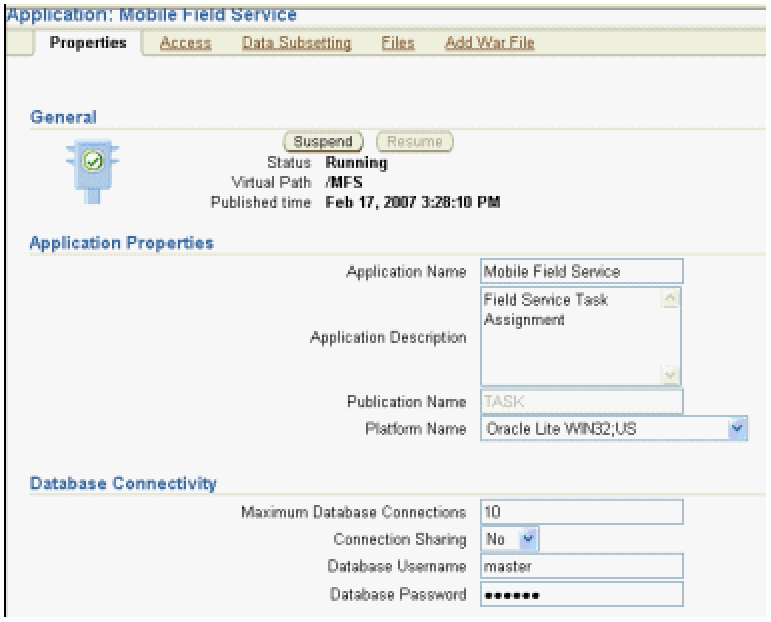
To set the Task application properties, perform the following steps:

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 13–10 displays, the Application Properties page lists application properties and database connectivity details.

Figure 13–10 Application Properties Page



Status Running
Virtual Path /MFS
Published time Feb 17, 2007 3:28:10 PM

Application Properties

Application Name	Mobile Field Service
Application Description	Field Service Task Assignment
Publication Name	TASK
Platform Name	Oracle Lite WIN32;US

Database Connectivity

Maximum Database Connections	10
Connection Sharing	No
Database Username	master
Database Password	••••••

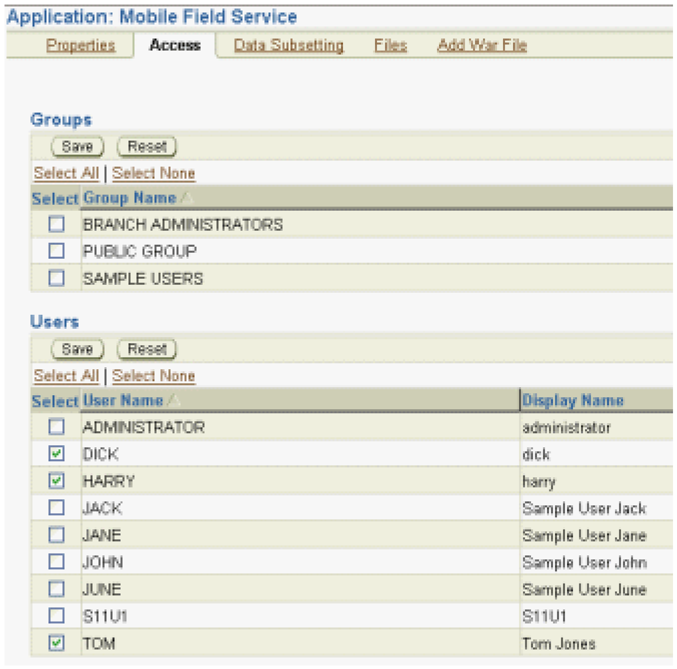
4. In the **Database Password** field, type the application demo schema password. In the past, this password was `master`. Click **Apply**. The Mobile Manager displays a confirmation message.

13.3.4 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 13–11 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 13–11 Access Page



Application: Mobile Field Service

Properties | Access | Data Subsetting | Files | Add War File

Groups

Save Reset

Select All | Select None

Select	Group Name
☐	BRANCH ADMINISTRATORS
☐	PUBLIC GROUP
☐	SAMPLE USERS

Users

Save Reset

Select All | Select None

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.

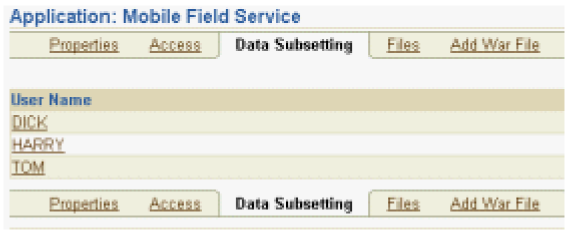
13.3.5 Defining Snapshot Template Values for the User

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 13–12 displays, the Data Subsetting page appears.

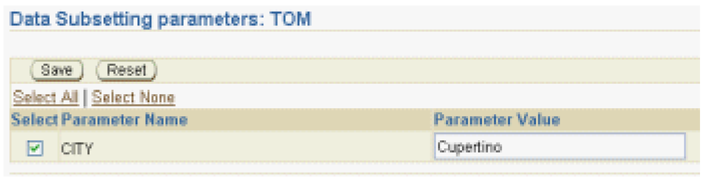
Figure 13–12 Data Subsetting Page


Application: Mobile Field Service

User Name
DICK
HARRY
TOM

2. Under the User Name column, click the user name link `Tom`. As Figure 13–13 displays, the Data Subsetting Parameters page appears.

Figure 13–13 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*.

13.4 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 13.4.1, "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 13.4.2, "Browse the TASK Snapshot and Update a Row"
- Section 13.4.3, "Develop your Mobile Field Service Application Using Oracle Database Lite"
- Section 13.4.4, "Republish the Application with the Application Program"

13.4.1 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.

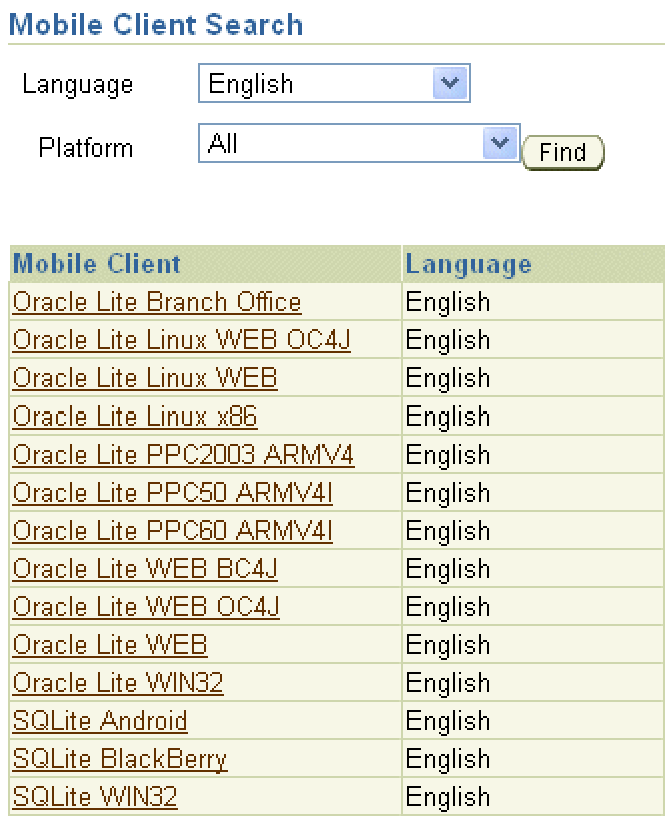
To install the Mobile Client for Win32, perform the following actions:

1. Start your Web browser and connect to the Mobile Server by entering the following URL.

   ```
   http://<mobile_server>/webtogo/setup
   ```

2. As Figure 13–14 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 13–14 Mobile Client Setup Page



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 PPC2003 ARMV4	English
Oracle Lite PPC50 ARMV4I	English
Oracle Lite PPC60 ARMV4I	English
Oracle Lite WEB BC4J	English
Oracle Lite WEB OC4J	English
Oracle Lite WEB	English
Oracle Lite WIN32	English
SQLite Android	English
SQLite BlackBerry	English
SQLite 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).

13.4.2 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.

13.4.3 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`.

13.4.4 Republish the Application with the Application Program

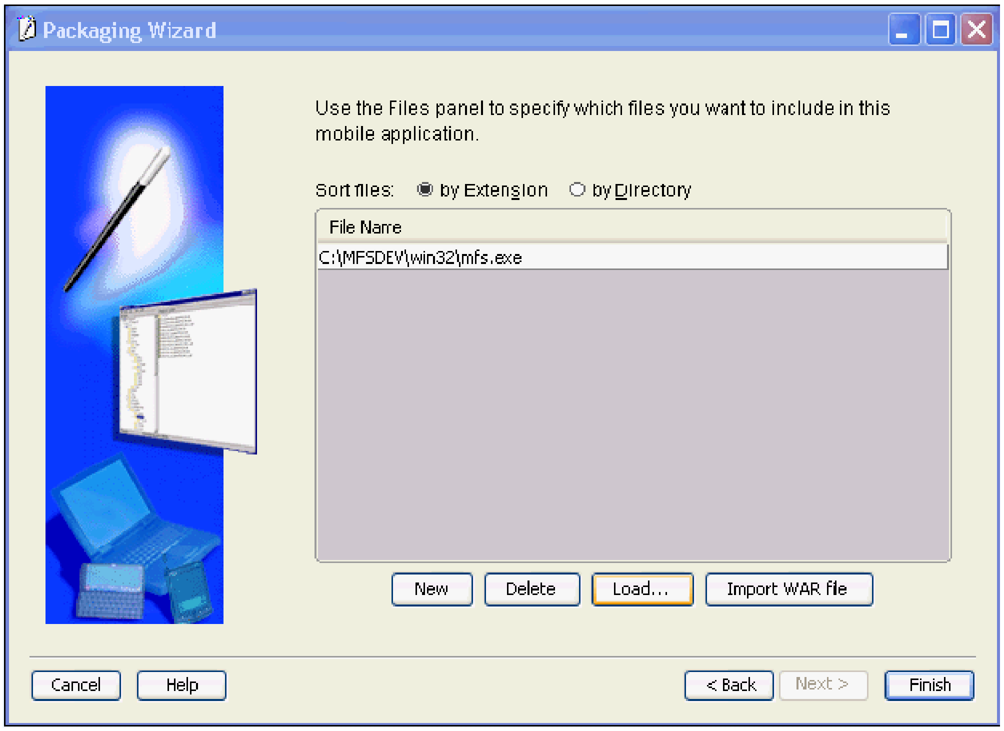
Use the Packaging Wizard to republish the application, as follows:

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 13–15, verify that the `mfs.exe` file is listed in the "File Name" window and click **Finish**.

Figure 13–15 Load the MFS.EXE File.



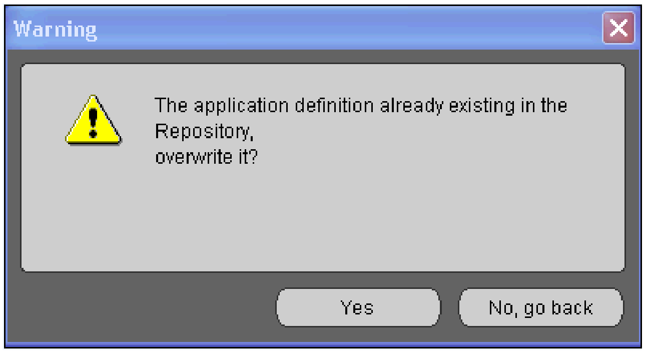
Use the Files panel to specify which files you want to include in this mobile application.

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 13–16 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.

14

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.Net 2003 or 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.

Note: If you use Visual Studio.Net 2005 and/or ADO.Net, you must install the ODBC 3.5 driver. See Section 5.1.3, "ODBC" for details.

The following sections detail the development process:

- Section 14.1, "Overview of the WinCE Sample Application"
- Section 14.2, "Develop the Application"
- Section 14.3, "Create Publication for Application"
- Section 14.4, "Package and Publish the Application"
- Section 14.5, "Administer the Application"
- Section 14.6, "Run the Application on the Windows Mobile/Pocket PC Device"

14.1 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.

14.1.1 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 14.1.1.1, "Application Development Computer Requirements"
- Section 14.1.1.2, "Client Device Requirements"

14.1.1.1 Application Development Computer Requirements

Table 14–1 lists the configuration and installation requirements for the Mobile application development computer.

Table 14–1 Application Development Computer Requirements

Requirement	Description
Windows User Login	The login user on the Windows 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.

14.1.1.2 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 14.6.1, "Install the Oracle Database Lite Mobile client for Pocket PC".

14.2 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 14.2.1, "Create Database Objects in the Oracle Server"
2. Section 14.2.2, "Write the Application Code"
3. Section 14.2.3, "Compile the Application"

14.2.1 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.

14.2.1.1 The WinCE Transport Application Database Objects

The WinCE Transport application uses the following database objects:

- Packages Table
- Routes Table
- Trucks Table

Table 14–2 lists the columns for the Packages table for storing information about the package.

Table 14–2 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 14–3 lists the columns for the Routes table for storing information about a route.

Table 14–3 Routes Table

Column	Description
`ROUTE_NO`	Route Number (Primary Key)
`ROUTE_NM`	Route Name
`EST_TIME`	Estimated Time

Table 14–4 lists columns for the Trucks table for storing information about the availability status and destination information for a truck.

Table 14–4 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.

14.2.2 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 14.2.2.1, "Transport Module (Transport.vb)"
- Section 14.2.2.2, "Main Form (frmMain.vb)"
- Section 14.2.2.3, "View Packages (frmView.vb)"
- Section 14.2.2.4, "Create Package (frmNew.vb)"

14.2.2.1 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.

14.2.2.2 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 4.4.2, "Using the OracleEngine to Synchronize" for more information on this class.

14.2.2.3 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 8.1.3, "Create Commands With the `OracleCommand` Class" in the *Oracle Database Lite Client Guide*.

```
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)
```

14.2.2.4 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 8.1.3, "Create Commands With the `OracleCommand` Class" in the *Oracle Database Lite Client Guide*.

```
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()
```

14.2.3 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).

14.2.3.1 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.

14.2.3.2 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 14.4, "Package and Publish the Application" for directions on how to package and publish the application.

14.3 Create Publication for Application

As described fully in Chapter 6, "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 6, "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 14.3.1, "Create a Project"
- Section 14.3.2, "Create Publication Items"
- Section 14.3.3, "Create Publication"

14.3.1 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 6.2, "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**.

14.3.2 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 6.4, "Create a Publication Item".

14.3.2.1 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**.

14.3.2.2 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**.

14.3.2.3 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**.

14.3.3 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**.

14.4 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 14.4.1, "Define the Application Using the Packaging Wizard"
2. Section 14.4.2, "Publish the Application"

14.4.1 Define the Application Using the Packaging Wizard

Using the Packaging Wizard, you can select and describe the Transport application.

14.4.1.1 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.

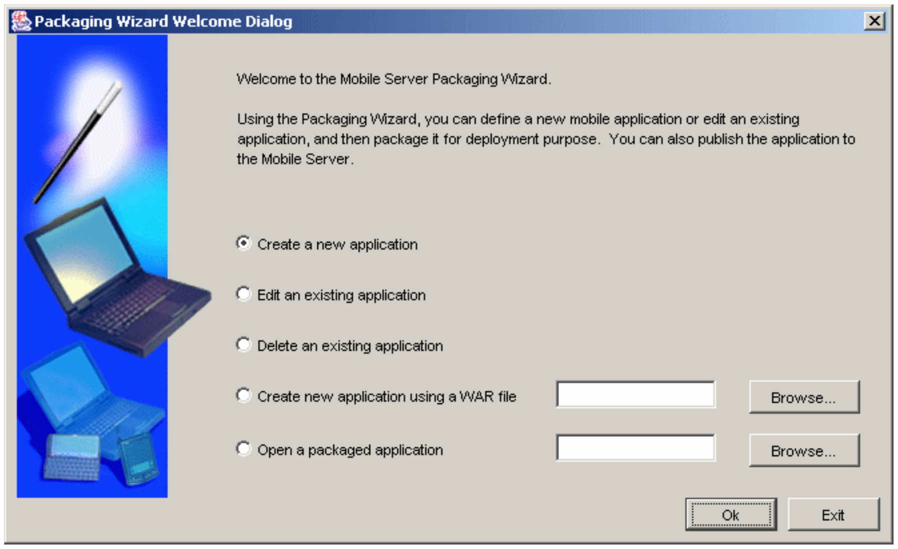
1. Using the Command Prompt, enter the following.

   ```
   cd ORACLE_HOME\Mobile\SDK\bin
   ```

   ```
   wtgpack
   ```

   As Figure 14–1 displays, the Packaging Wizard displays the Welcome panel. Select the **Create a new application** option and click **OK**.

Figure 14–1 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

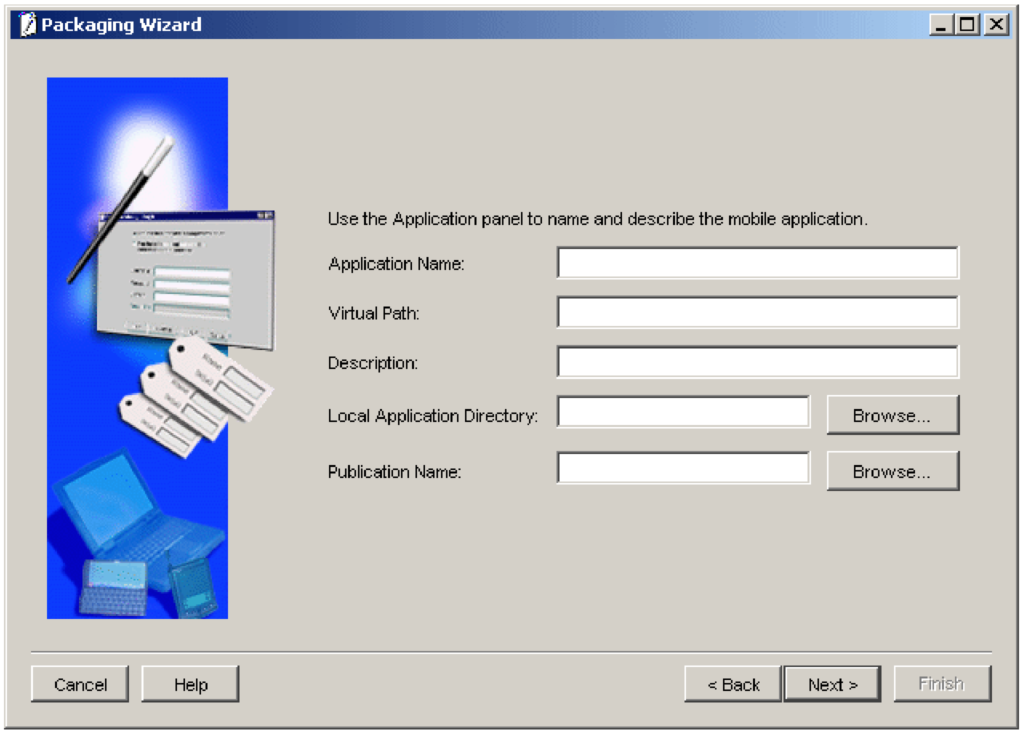
Create new application using a WAR file

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 14–5 describes, enter the WinCE Transport application settings. Figure 14–2 displays the Applications panel.

Figure 14–2 Applications Panel



Use the Application panel to name and describe the mobile application.

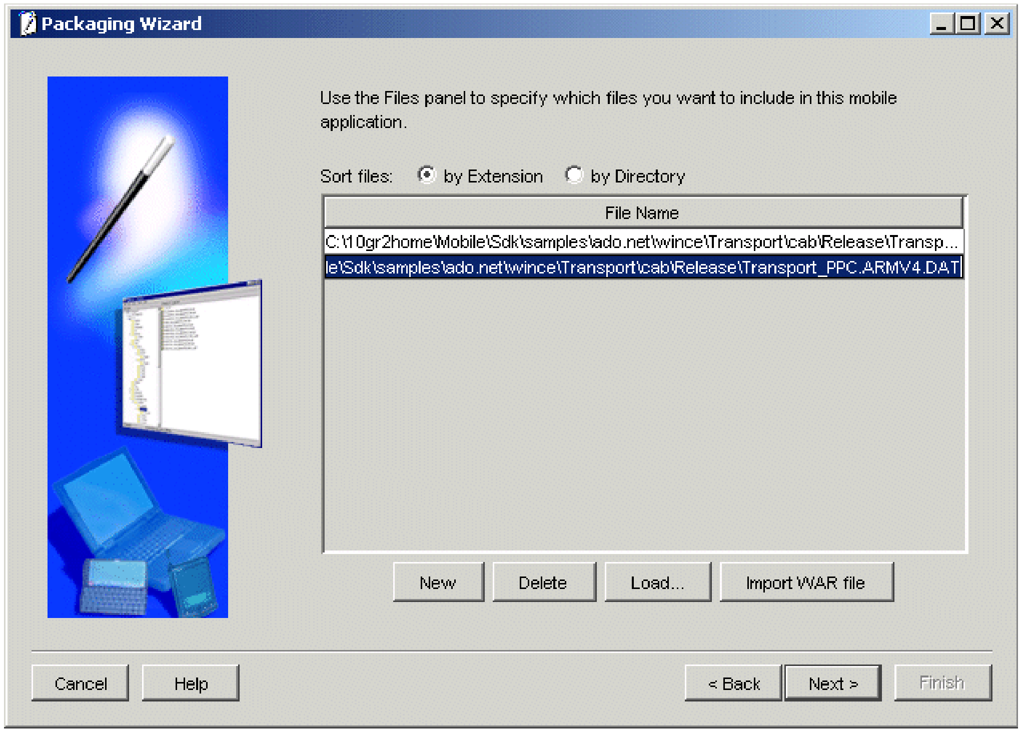
Application Name:	
Virtual Path:	
Description:	
Local Application Directory:	
Publication Name:	

Table 14–5 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 14–3 displays, the Files panel appears.

Figure 14–3 Files Panel


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:\10gr2home\Mobile\Sdk\samples\ado.net\wince\Transport\cab\Release\Transp...
le\Sdk\samples\ado.net\wince\Transport\cab\Release\Transport_PPC.ARMV4.DAT
New
Delete
Load...
Import WAR file
Cancel
Help
< Back
Next >
Finish

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**.

14.4.2 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 14–6 describes, enter the specified values.

Table 14–6 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 14–6 (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.

14.5 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 14.5.1, "Start the Mobile Server"
2. Section 14.5.2, "Launch the Mobile Manager"
3. Section 14.5.3, "Create a New User"
4. Section 14.5.4, "Set the Application Properties"
5. Section 14.5.5, "Grant User Access to the Application"

For more information on the Mobile Manager see the *Oracle Database Lite Administration and Deployment Guide*.

14.5.1 Start the Mobile Server

To start the Mobile Server in standalone mode, enter the following command using the Command Prompt.

```
> runmobileserver
```

14.5.2 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.

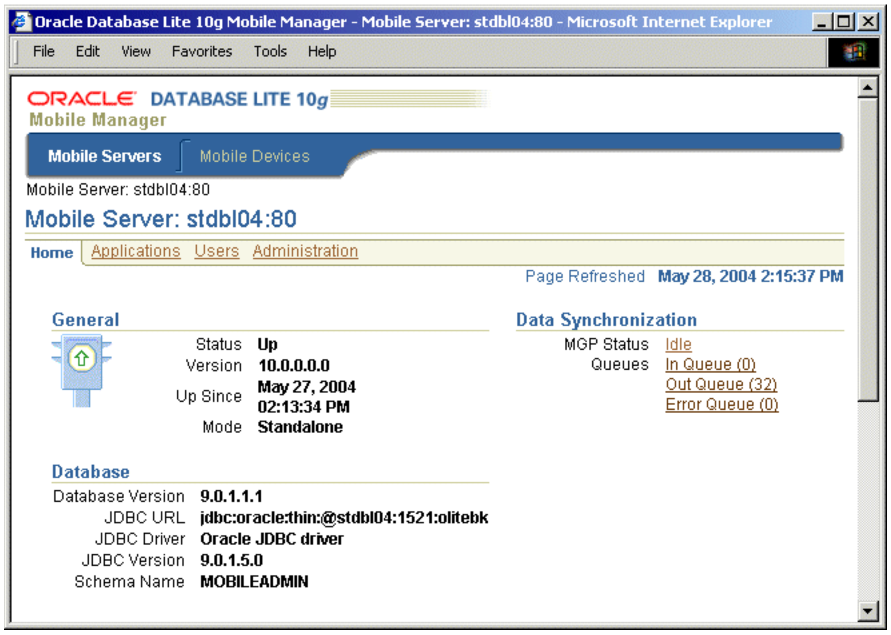
   ```
   http://<mobile_server>/webtogo
   ```

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 14–4 displays the Mobile Server home page.

Figure 14–4 Mobile Server Home Page


Oracle Database Lite 10g Mobile Manager - Mobile Server: stdbl04:80 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
ORACLE DATABASE LITE 10g
Mobile Manager
Mobile Servers
Mobile Devices
Home
Applications
Users
Administration

Mobile Server: stdbl04:80

Mobile Server: stdbl04:80

Page Refreshed May 28, 2004 2:15:37 PM

General

Status	Up
Version	10.0.0.0.0
Up Since	May 27, 2004 02:13:34 PM
Mode	Standalone

Data Synchronization

MGP Status	Idle
Queues	In Queue (0)
	Out Queue (32)
	Error Queue (0)

Database

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

14.5.3 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 14–7.
4. Click **Save**. The Mobile Manager displays a confirmation message.
5. Click **OK**.

Table 14–7 lists the values that you must enter in the **Add User** page.

Table 14–7 The Add User Page Description

Field	Value
Display Name	bob
User Name	bob

Table 14–7 (Cont.) The Add User Page Description

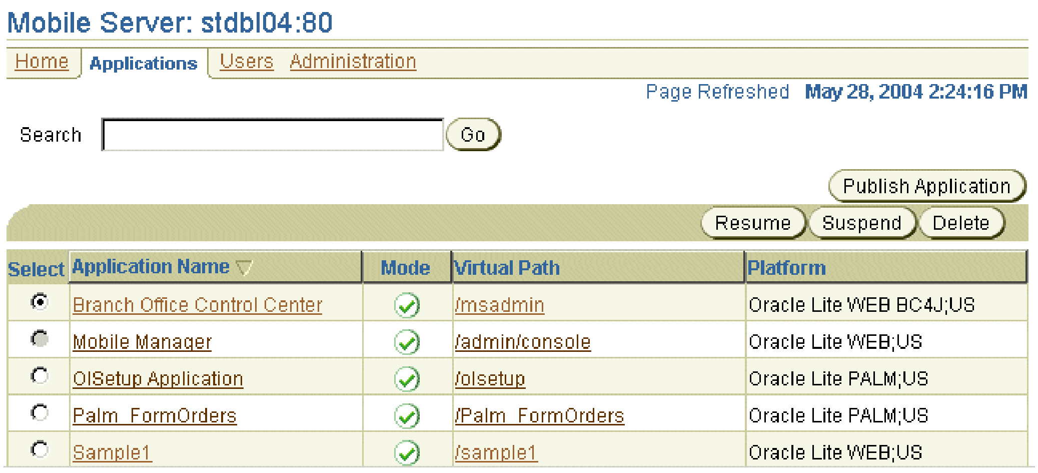
Field	Value
Password	bobhope
Password Confirm	Re-enter the password for confirmation
System Privilege	Select the "User" option

14.5.4 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 14–5 displays, The **Applications** page appears. You can search the list of available applications by application name.

Figure 14–5 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**.

14.5.5 Grant User Access to the Application

To grant user access to the Transport application, perform the following steps.

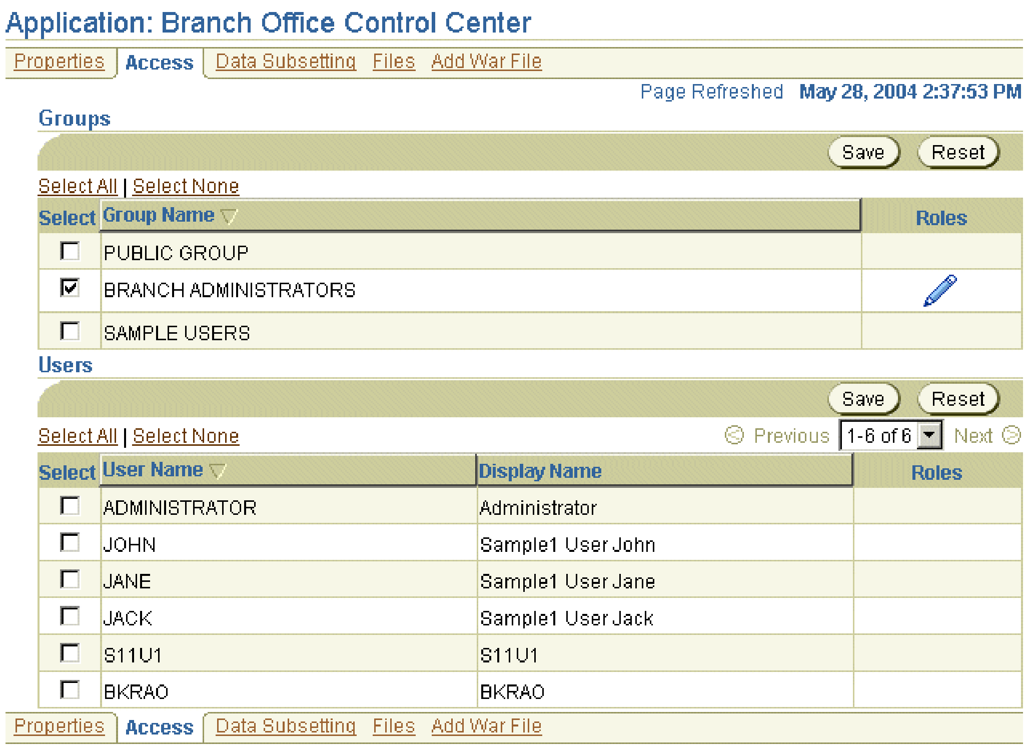
1. In the Transport application page, click the **Access** link. As Figure 14–6 displays, the Access page lists application users and application groups. To grant access to a user or a group of users to the Transport application, select the corresponding boxes.

   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 14–6 displays the Access page of the Transport application.

Figure 14–6 Access Page



Page Refreshed May 28, 2004 2:37:53 PM

Groups

Select	Group Name	Roles
☐	PUBLIC GROUP	
☑	BRANCH ADMINISTRATORS	✎
☐	SAMPLE USERS	

Users

1-6 of 6

Select	User Name	Display Name	Roles
☐	ADMINISTRATOR	Administrator	
☐	JOHN	Sample1 User John	
☐	JANE	Sample1 User Jane	
☐	JACK	Sample1 User Jack	
☐	S11U1	S11U1	
☐	BKRAO	BKRAO	

14.6 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 14.6.1, "Install the Oracle Database Lite Mobile client for Pocket PC"
2. Section 14.6.2, "Install and Synchronize the Transport Application and Data"

14.6.1 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
   ```

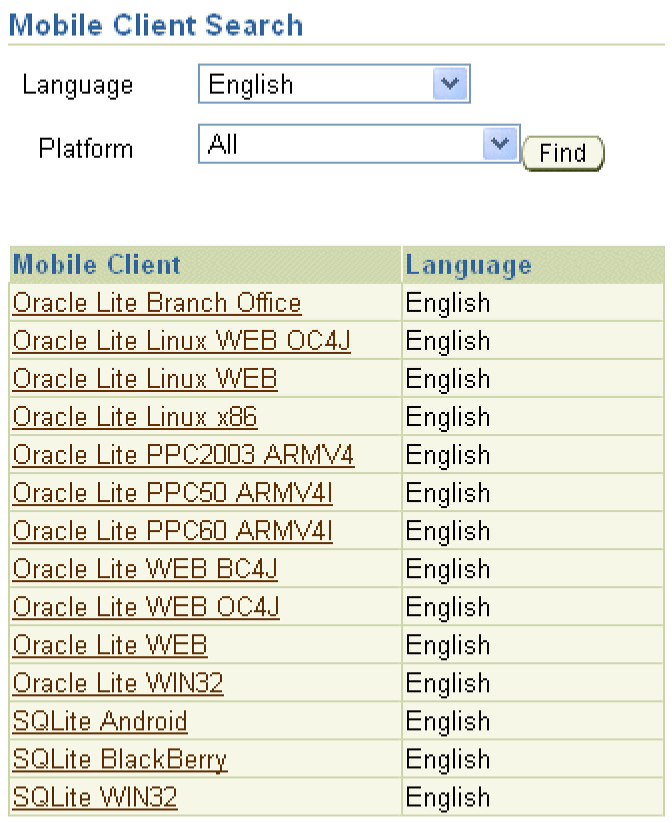
   Note: You must replace the `<mobile_server>` variable with the host name or IP address of your Mobile Server.

   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 14–7 displays the Mobile Client Setup page.

Figure 14–7 Mobile Client Setup Page



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 PPC2003 ARMV4	English
Oracle Lite PPC50 ARMV4I	English
Oracle Lite PPC60 ARMV4I	English
Oracle Lite WEB BC4J	English
Oracle Lite WEB OC4J	English
Oracle Lite WEB	English
Oracle Lite WIN32	English
SQLite Android	English
SQLite BlackBerry	English
SQLite WIN32	English

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.

14.6.2 Install and Synchronize the Transport Application and Data

To install the Transport application and data, perform the following steps.

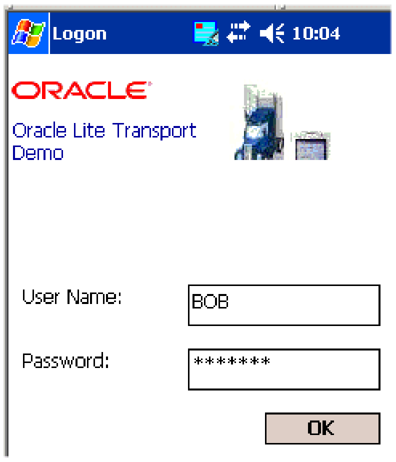
1. On the device, locate and tap the `msync` application icon in the programs group.
2. The msync dialog appears. To download the Transport application and snapshots for user BOB, enter data as described in Table 14–8.

Table 14–8 Values You Must Enter in the msync Dialog

Name	Value
UserName	bob
Password	bobhope (all lowercase)
Save password box	Select
Server	Machine name or IP address

Figure 14–8 displays the **msync** dialog on the Pocket PC.

Figure 14–8 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.

Index