Oracle® Database Lite

Developer's Guide

10*g* (10.2.0)

Part No. B15920-01

June 2005

ORACLE®

Oracle Database Lite Developer's Guide 10*g* (10.2.0)

Part No. B15920-01

Contents

19 Building Mobile Web Applications: A Tutorial

B Oracle Lite Database Utilities

Send Us Your Comments

Oracle Database Lite Developer's Guide 10*g* (10.2.0)

Part No. B15920-01

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.

Preface

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

Intended Audience

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

Documentation Accessibility

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

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

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

Structure

This guide includes the following topics:

- Chapter 1, "Overview"

  Provides an introduction to Oracle Database Lite 10*g* and gives an overview of the application development process using the Mobile Development Kit.

- Chapter 2, "The Oracle Database Lite RDBMS"

  Presents the Oracle Database Lite Relational Database Management System (RDBMS).

- Chapter 3, "Synchronization"

  Describes synchronization functions between Oracle Database Lite and an Oracle database using the Mobile Server and the Mobile Sync client application.

- Chapter 4, "Invoking Synchronization APIs from Applications"

  Describes APIs for COM, C, C++, and Java for programmatically starting the synchronization phases.

- Chapter 5, "Using Mobile Database Workbench to Create Publications"

  Use the Mobile Database Workbench to create publications and subscriptions.

- Chapter 6, "Using the Packaging Wizard"

  Describes how to use the packaging wizard when publishing applications.

- Chapter 7, "Developing Mobile Web Applications"

  Discusses Web based Mobile application development.

- Chapter 8, "Native Application Development"

  Discusses Mobile application development for native platforms.

- Chapter 9, "Java Application Development"

  Describes how to develop and test Java applications.

- Chapter 10, "JDBC Programming"

  Discusses the Oracle Database Lite support for JDBC programming.

- Chapter 11, "Java Stored Procedures and Triggers"

  Describes how to use Java stored procedures and triggers within the Oracle Database Lite support for JDBC programming.

- Chapter 12, "Using Simple Object Data Access (SODA) for PalmOS and PocketPC Platforms"

  Describes support for PalmOS and Pocket PC/Windows CE devices using the SODA interface.

- Chapter 13, "Developing Mobile Applications for Palm OS Devices"

  Discusses building Oracle Database Lite 10*g* applications for Palm devices. Oracle Database Lite 10*g* for Palm OS supports Simple Object Database Access (SODA) and Open Database Connectivity (ODBC) as programming interfaces. This document also describes how to build and run Oracle Database Lite 10*g* applications using Metrowerks CodeWarrior 9.

- Chapter 14, "Oracle Database Lite ADO.NET Provider"

Discusses ADO.NET support for Windows CE.

- Chapter 15, "Oracle Database Lite Transaction Support"

  Describes how to implement transactions.

- Chapter 16, "Oracle Database Lite Development Performance Considerations"

  Describes how you can increase performance for Oracle Database Lite.

- Chapter 17, "Security"

  Describes security configuration for Oracle Database Lite.

- Chapter 18, "Tracing"

  Describes how to initiate tracing.

- Chapter 19, "Building Mobile Web Applications: A Tutorial"

  Guides you through the relevant phases of implementing a Web application for Mobile devices through a tutorial.

- Chapter 20, "Building Offline Mobile Web Applications Using BC4J: A Tutorial"

  Enables you to create, deploy, and use a BC4J application through a tutorial.

- Chapter 21, "Building Offline Mobile Applications for Win32: A Tutorial"

  Guides you through the Mobile application development process for the Win32 platform through a tutorial.

- Chapter 22, "Building Offline Mobile Applications for Windows CE: A Tutorial"

  Describes how to build a Visual Basic.NET (Visual Studio.NET 2003) application using the Oracle Database Lite 10*g* ADO.NET interface for Pocket PC through a tutorial.

- Appendix A, "Optimizing SQL Queries"

  Provides tips on improving the performance of your SQL queries.

- Appendix B, "Oracle Lite Database Utilities"

  Describes the utilities you can use to manage your Oracle Lite database.

- Appendix C, "ODBC Support on Palm"

  Describes the Open Database Connectivity (ODBC) support provided in Oracle Database Lite 10*g* for the Palm OS Platform.

1 Overview

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

1.1 Introduction

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

Offline Mobile database applications can be developed in many ways. The most common way is to develop native C or C++ applications for specific Mobile platforms. C++ applications can access the Oracle Database Lite database using the Simple Object Data Access API (SODA), an easy-to-use C++ interface that is optimized for the object-oriented and SQL functionality of Oracle Database Lite. For more information about SODA, refer the SODA API documentation, which is installed as part of the Mobile Development Kit.

Applications that need a standard interface and work with multiple database engines can use either the Open Database Connectivity (ODBC) interface, Active Data Object (ADO) interface, or some other interface built on top of ODBC. ADO.NET can be used on Win32 and Windows CE. Another way to develop an offline Mobile database application is to use Java and the Java Database Connectivity (JDBC) interface. Oracle

Database Lite also offers a third way to develop offline Mobile database applications using the servlet based Web model called Web-to-Go.

Web-to-Go applications can be built using Web technologies, such as servlet, Java Sever Pages (JSP), applet, HTML, and JDBC.

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

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

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

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

1.2 Determining Application Development Environment

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

Platform	Programming Languages	Operating System	Hardware
Win32 on a laptop or notebook	When you develop on a laptop, you are using one of the Windows operating systems. You can use any of the languages mentioned in this book. However, C, C++ are better for creating applications with a good user interface. The languages available are C, C++, C#, Java, Visual Basic, JSPs and Servlets.	Windows 95/98/NT/2000	Pentium processor
PocketPC	C, C++, Visual Basic, Java applications (no Servlet or JSP support), SODA, ADO.NET	Windows CE (WinCE)	ARM, xScale
Linux	Java, C, C++	Linux Redhat 3.0	X86
Palm Pilot	C, C++, use MetroWerks CodeWarrior tool, SODA	PalmOS	ARM, DragonBall

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

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

1.2.1 Native Applications

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

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

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

1.2.2 Standalone Java Applications

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

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

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

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

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

   ```
   jdbc:polite:dsn
   ```

   Or if not local, connect as follows:

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

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

1.2.3 Web Applications

You can execute existing Web applications using the J2EE Java technologies, such as servlets and JSPs, in a disconnected mode without modifying the code base. Web-to-Go is a development option for Web applications, and can be executed on laptops using Windows 95/98/NT/2000. Web-to-Go applications use Java servlets and JSPs that may invoke JDBC to access the database, as opposed to using application APIs, such as C or C++.

For more information, see Chapter 7, "Developing Mobile Web Applications".

1.3 Oracle Database Lite Application Model and Architecture

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

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

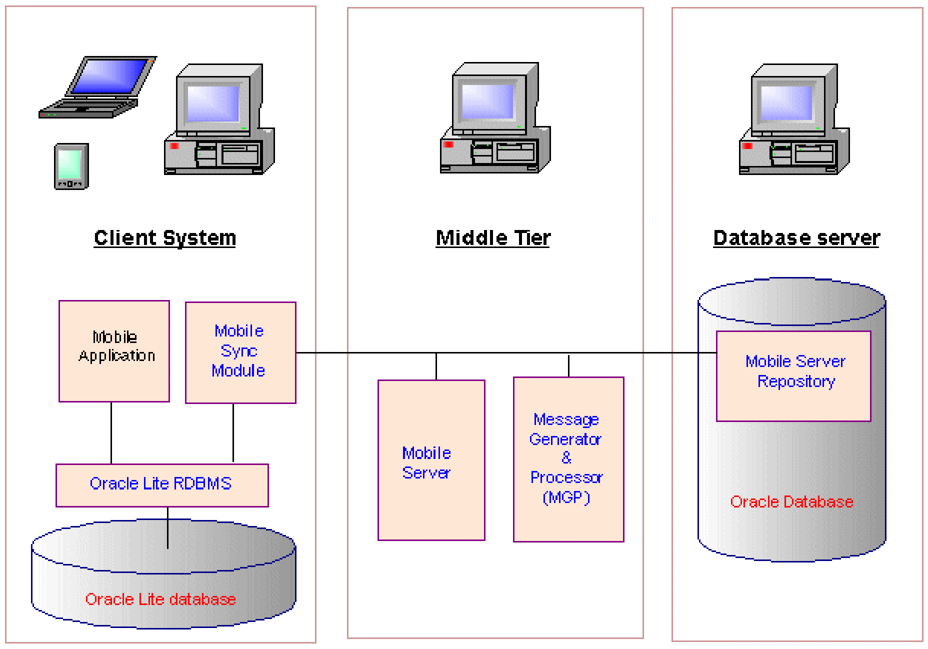
When a user synchronizes the Mobile client for the first time, the Mobile client creates an Oracle Database Lite database on the client machine for each subscription that is provisioned to the user. The Mobile client then creates a snapshot in this database for each publication item contained in the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

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

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

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

Figure 1–1 Oracle Database Lite Architecture



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

1.3.1 Oracle Database Lite RDBMS

The Oracle Database Lite RDBMS is a small footprint, Java-enabled, secure, relational database management system created specifically for laptop computers, handheld computers, PDAs, and information appliances. The Oracle Database Lite RDBMS runs on Windows 98/NT/2000/XP, Windows CE/Pocket PC, Linux, and Palm. Oracle Database Lite RDBMS provides JDBC, ODBC, and SODA interfaces to build database applications from a variety of programming languages such as Java, C/C++, and Visual Basic. These database applications can be used while the user is disconnected from the Oracle database server.

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

1.3.2 Mobile Sync

Mobile Sync (msync) is a small footprint application that resides on the Mobile device. Mobile Sync enables you to synchronize data between handheld devices, desktop and laptop computers and Oracle databases. Mobile Sync runs on Windows 98/NT/2000/XP, Windows CE/Pocket PC, Linux, and Palm.

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

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

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

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

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

1.3.3 Mobile Server

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

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

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

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

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

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

Using the Resource Manager API, you can programmatically perform the above tasks.

End users install Mobile applications in two steps. First, as the Mobile user, you browse the setup page on the Mobile Server and choose the setup program for the platform you want to use. The setup program only runs on Windows 32 platforms. For Windows 32 based client systems, you can download the setup program directly to the Windows 32 system and execute it to set up the Oracle Database Lite client. For Windows CE and Palm devices, you must download the setup program to your Windows 32 desktop first and execute it there. Then you must use ActiveSync for Windows CE or Hot Sync for Palm to install the Oracle Database Lite client on the device.

Second, you run the Mobile Sync (msync) command on your Mobile device, which prompts for the user name and password. The Mobile Sync application communicates with the Consolidator Manager module of the Mobile Server and downloads the applications and the data provisioning for the user.

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

1.3.4 Message Generator and Processor (MGP)

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

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

The Apply Phase

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

The Compose Phase

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

Applying Changes to the Server Database

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

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

1.3.5 Mobile Server Repository

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

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

1.4 Execution Models for Oracle Lite Database

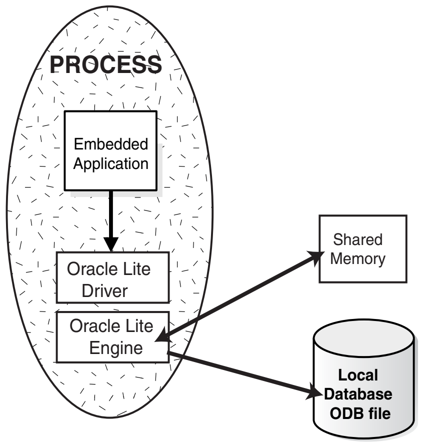
How your application integrates with the Oracle Lite database depends on how you design the execution model, as described in the following sections:

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

1.4.1 Embedded Application in Single Process

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

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



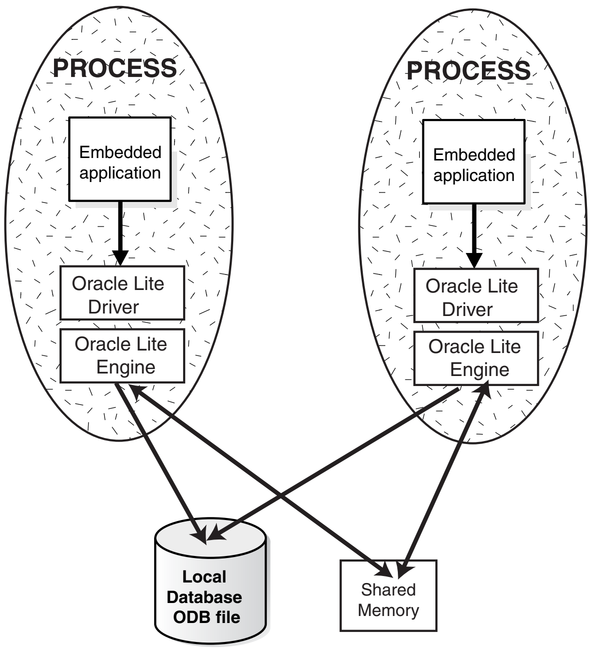
See Chapter 2.3, "Creating and Managing the Database in an Embedded Application" for more information on how to embed an Oracle Lite database into a standalone application.

1.4.2 Multiple Processes Accessing the Same Database

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

As shown in Figure 1–3, this type of scenario is automatically created when you include the same publication within two different applications during the packaging and deploying process.

Figure 1–3 Applications in Multiple Processes Accessing Single Database



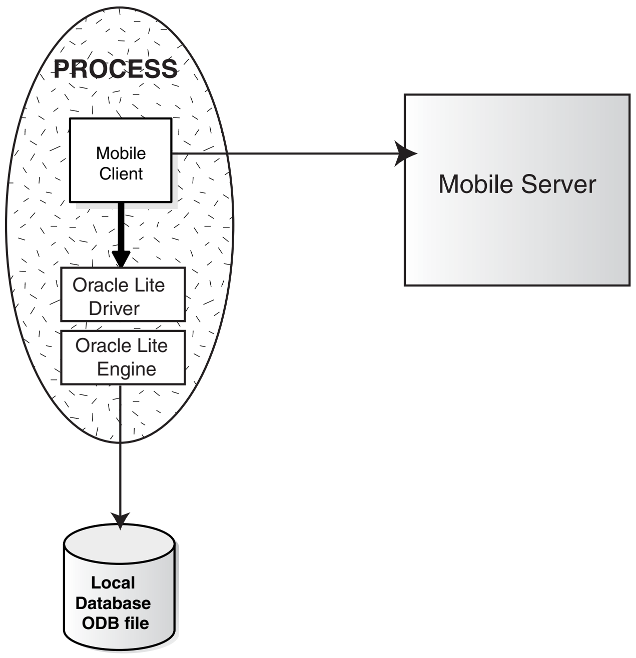
1.4.3 Multiple Embedded Application Clients Accessing Remote Database

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

1.4.4 Mobile Client in Single Process

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

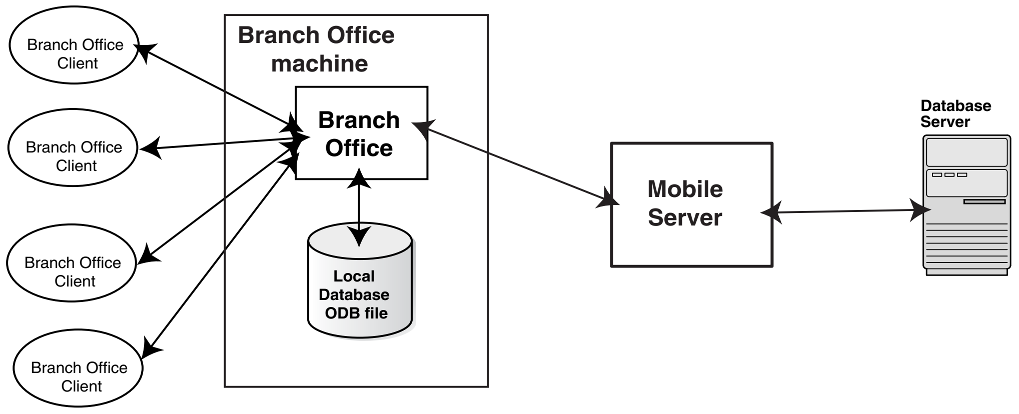
Figure 1–4 Diagram of Mobile Client and ODB Libraries in SIngle Process



1.4.5 Multiple Clients Accessing Remote Database

If you have several remote clients accessing the same data, you can use Branch Office to facilitate the remote applications. Figure 1–5 demonstrates how multiple remote Branch Office clients access the data from a single machine. This machine contains the Branch Office executables and the local Oracle Lite database, which all clients access for their information. When a synchronization is requested, information is communicated between the Branch Office and the back-end database through the Mobile Server.

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



See the Branch Office chapter in the *Oracle Database Lite Administration and Deployment Guide* for more information.

1.5 Mobile Development Kit (MDK)

Before you develop an offline application using Oracle Database Lite, you should install the Oracle Database Lite Mobile Development Kit (MDK) on the machine on which you intend to develop your application. for instructions on how to install the Mobile Development Kit, see the *Oracle Database Lite Getting Started Guide*.

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

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

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

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

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

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

The Samples directory `<`*`ORACLE_HOME`*`>\Mobile\Sdk\Samples` directory contains some sample applications. Chapter 2, "The Oracle Database Lite RDBMS", Chapter 2.12, "Using Oracle Database Lite Samples" describes the sample programs and explains how to run them. You should familiarize yourself with the various Oracle Database Lite features by perusing the source code and running the samples.

The `<`*`ORACLE_HOME`*`>\Mobile\Sdk\OLDB40` directory contains a starter database file named `polite.odb`.

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

At the Command Prompt, enter the following.

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

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

1.5.1 Mobile SQL (mSQL)

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

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

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

1.5.2 Using the Mobile Development 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 5, "Using Mobile Database Workbench to Create Publications".

1.5.3 Using the Packaging Wizard

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

1. Create a new Mobile application.
2. Edit an existing Mobile application.

3. Publish an application to the Mobile Server.

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

For detailed information on how to use the Packaging Wizard, see Chapter 6, "Using the Packaging Wizard".

1.6 Supported Platforms

Your development environment must include Oracle Database Lite 10*g* as the encompassing platform. For developing native applications on the Oracle Database Lite 10*g* platform, the following operating system platforms are supported:

- Microsoft Windows NT/2000/XP
- Windows Mobile 2003 Second Edition software for Pocket PCs (Windows CE 4.2)

  The following Windows CE chipsets are supported:

  - Pocket PC 2003 (ARM, xScale, Emulator)
  - Pocket PC (ARM)
- Palm OS 3.5 through 5.2

1.7 Java Support

For more information, refer Chapter 8, "Native Application Development", Section 1.7, "Java Support".

1.8 Data Source Name

For more information, refer Chapter 8, "Native Application Development", Section 1.8, "Data Source Name".

Part I

Developing Applications for Oracle Database Lite

The chapters in this part describe the APIs that you can use when developing your applications for the Oracle Database Lite environment. In addition, topics such as security, performance, and tracing are included.

Part I contains the following chapters:

2 The Oracle Database Lite RDBMS

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

2.1 Oracle Lite Database Overview

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

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

Oracle Database Lite creates all ODB files with an automatic name and assigns a data source name (DSN). The DSN is used to connect to the database using ODBC, JDBC or

ADO.NET APIs. In order to make the connection, you must know the DSN name for your ODB file. When you install the Mobile Development Kit, a default database is installed with database name of `polite.odb` and DSN name of `polite`. However, when you synchronize, an Oracle Lite database is created for each publication (under a directory named after each user). The DSN for these ODB files is a combination of the user and publication names.

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

2.2 Creating and Managing the Database for a Mobile Client

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

The Oracle Lite database for the Mobile client is automatically created on the first synchronization request. In addition, the data is replicated and updated with the data on the Oracle database automatically for you. See Section 3.2, "What is The Process for Setting Up a User For Synchronization?" for techniques that can be used to create publication items on the Mobile Server, which then automatically creates snapshots on the client when you synchronize with the database.

2.3 Creating and Managing the Database in an Embedded Application

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

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

2.3.1 Install Oracle Database Lite Runtime

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

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

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

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

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

2.3.3 Creating a Unique Oracle Lite Database for an Embedded Application

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

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

2.3.3.1 Creating a Data Source Name with ODBC Administrator

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

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

2.3.3.1.1 Creating DSN on a Windows System Create the DSN on a Windows system through the Microsoft ODBC Administrator, which is a tool that manages the `ODBC.INI` file and associated registry entries in Windows 98/NT/2000/XP. Within this tool, add the data source name for your ODB file and specify the database file you want to dedicate as the default for the data source name. For more information on DSN properties, see Table 2–1 and Table 2–2.

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

2.3.3.1.2 Creating DSN on a LINUX System In order to create a DSN on a LINUX platform, add the DSN in the `ODBC.INI` file, which is located in the *`ORACLE_HOME`*`\olite\bin` directory. In this file, add the DSN in its own section, where the section name is the DSN name. For example, the following `ODBC.INI` example contains two DSN configurations:

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

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

[Politecl]
Description=Oracle Lite 40 Data Source
DataDirectory=/home/olite
ServerHostName=localhost
ServerPortNumber=10000
Database=polite
IsolationLevel=Read Committed
Autocommit=Off
CursorType=Static
```

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

Table 2–1 POLITE.INI DSN Parameters

DSN Parameter	Description
`Description`	An optional description for the data source. Use only for Windows environment.
`Data Directory`	The path to the data directory where the database resides. This is an existing path.
`Database`	Oracle Database Lite database name to be created. Do not include the `.ODB` extension.
`Default Isolation Level`	Determines the degree to which operations in different transactions are visible to each other. For more information on the supported isolation levels, refer to Section 15.2, "What Are the Transaction Isolation Levels?" for more information. The default level is `Read Committed`. Other options are `Repeatable Read`, `Single User`, and `Serializable`.

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

DSN Parameter	Description
`Autocommit`	Commits every database update operation in a transaction when that operation is performed. Auto-commit values are Off and On. The default value is Off. ▪ On: DML and DDLs are automatically committed. ▪ Off: An application has to explicitly issue the transaction commit or rollback commands. **Note**: In the Microsoft ODBC SDK, the ODBC driver defaults to auto-commit mode. However, the default for Oracle Database Lite is manual-commit mode. In this environment, if you execute SQLEndTrans / SQLTransact call with SQL_COMMIT option using the ODBC driver, you receive a SQL_SUCCESS, because ODBC believes that auto-commit is on. However, no commit actually occurs, because ODBC transfers the transaction to Oracle Database Lite, whose default is manual-commit. You must configure the Microsoft ODBC Driver Manager to transfer control of the SQLEndTrans / SQLTransact API call to Oracle Database Lite by explicitly setting autocommit to OFF in ODBC. When you do this, ODBC does not try to autocommit, but gives control of the transaction to Oracle Database Lite. To set auto-commit to off, execute either the `SQLSetConnectAtrr` or `SQLSetConnectOption` method with `SQL_AUTOCOMMIT_OFF` as the value of the `SQL_AUTOCOMMIT` option. Then, the `SQLEndTrans / SQLTransact` calls will commit as defaulted within Oracle Database Lite. Thus, if you want auto-commit on, turn it on only within Oracle Database Lite.
`Cursor Type`	▪ *Forward Only*: Default. A non-scrollable cursor which only moves forward but not backward through the result set. As a result, the cursor cannot go back to previously fetched rows. ▪ *Dynamic*: Capable of detecting changes to the membership, order, or values of a result set after the cursor is opened. If a dynamic cursor fetches rows that are subsequently deleted or updated by another application, it detects those changes when it fetches those rows again. ▪ *Keyset Driven*: Does not detect change to the membership or order of a result set, but detects changes to the values of rows in the result set. ▪ *Static*: Does not detect changes to the membership, order or values of a result set after the cursor is opened. If a static cursor fetches a row that is subsequently updated by another application, it does not detect the changes even if it fetches the row again. See Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types" for details on the restrictions when combining cursor types and isolation levels.

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

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

Parameter	Description
`ServerHostName`	Provide the server machine hostname or IP address where the database service is running.
`ServerPortNumber`	The port number where the database service is listening for incoming requests.

Table 2–2 (Cont.) DSN Configuration Parameters for Multi-User Service on LINUX

Parameter	Description
`ServerDSN`	The server-side DSN. Thus, the client DSN name on the client machine can be different from the DSN on the server mahcine. This is required only if the client and server machines are not the same and the `Database Directory` and `Database` parameters are not required.

2.3.3.2 Creating a New Oracle Lite Database for the Embedded Application

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

```
CREATEDB myDSN myDBname sysPwd
```

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

```
CREATEDB polite mydb manager
```

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

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

2.3.3.3 Connecting to Your New Oracle Lite Database

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

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

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

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

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

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

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

2.3.4 Creating Users for Your Embedded Oracle Lite Database

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

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

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

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

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

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

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

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

2.3.4.1 Pre-Defined Roles

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

Table 2–3 Pre-Defined Roles

Role Name	Privileges Granted To Role
`ADMIN`	Enables the user to create other users and grant privileges other than `DBA` and `ADMIN` on any object in the schema: `CREATE SCHEMA`, `CREATE USER`, `ALTER USER`, `DROP USER`, `DROP SCHEMA`, `GRANT`, `REVOKE`
`DBA`	Enables the user to issue the following DDL statements which otherwise can only be issued by `SYSTEM`: All ADMIN privileges, `CREATE TABLE`, `CREATE ANY TABLE`, `CREATE VIEW`, `CREATE ANY VIEW`, `CREATE INDEX`, `CREATE ANY INDEX`, `ALTER TABLE`, `ALTER VIEW`, `DROP TABLE`, `DROP VIEW`, and `DROP INDEX`.

Table 2–3 (Cont.) Pre-Defined Roles

Role Name	Privileges Granted To Role
`RESOURCE`	The `RESOURCE` role grants the same level of control as the `DBA` role, but only over the user's own schema. The user can execute any of the following commands in a SQL statement: `CREATE TABLE`, `CREATE VIEW`, `CREATE INDEX`, `CREATE CONSTRAINT`, `ALTER TABLE`, `ALTER VIEW`, `ALTER INDEX`, `ALTER CONSTRAINT`, `DROP TABLE`, `DROP VIEW`, `DROP INDEX`, `DROP CONSTRAINT`, and `GRANT` or `REVOKE` privileges on any object under a user's own schema.

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

2.3.4.2 Building and Populating Demo Tables

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

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

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

You can also enter:

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

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

2.3.5 Packaging Your Embedded Application With the Oracle Database Lite Runtime

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

1. Copy the following files from the Mobile Development Kit library, which is located in *`ORACLE_HOME`*`/Mobile/Sdk`, into the directory in your PATH where your application DLLs are located.
   - `olite40.msb`: Oracle Database Lite message file
   - `oljdbc40.dll`: JDBC JNI library
   - `olobj40.dll`: Oracle Database Lite object kernel
   - `olod2040.dll`: Oracle Database Lite ODBC driver
   - `olsql40.dll`: Oracle Database Lite SQL runtime library
   - `olstddll.dll`: Oracle Lite Common library
2. If you are using the Multi-User Service, copy `olsv2040.exe` and `olsvmsg.dll` into your PATH where your application DLLs are located.

3. To use any Java program with Oracle Database Lite, make sure that the `olite40.jar` file, which is installed in `OLITE_HOME/bin`, is in the application `CLASSPATH`. If the Java program uses the multi-user service, also place this JAR file in the SYSTEM `CLASSPATH`. This JAR file contains the JDBC driver for Oracle Database Lite. Your environment must provide a Java Runtime Environment from Sun, version JDK 1.4.2 version or higher.
4. If you want to support the mSQL command-line tool for querying and managing the Oracle Lite database, then you must place the following files in the `PATH`:
   - `msql.dll`
   - `msql.exe`
   - `msql.jar`
5. Manage ODBC for creating the DSN and registering the ODBC driver. On Linux, modify the `ODBC.INI` file. On Windows, perform the following:
   a. To use Microsoft ODBC for the ODBC environment—including DSN creation support—or to create and manage DSN names programmatically, place the `olad2040.dll` in the `PATH`. This DLL provides a plug-in to programmatically access the ODBC administration tool—`odbcad32`—that is used to create DSNs.
   b. Register the ODBC driver for the product in the Windows Registry, as follows:

```
KEY:HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI\Oracle Lite 40 ODBC
Driver
VALUE:32Bit = 1
VALUE:ApiLevel = 0
VALUE:ConnectFunctions = YYN
VALUE:Driver = <path to olod2040.dll>
VALUE:DriverODBCVer = 02.00
VALUE:SQLLevel = 0
VALUE:Setup = <path_to_olad2040.dll>
KEY:HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI\ODBC DRIVERS
VALUE:Oracle Lite 40 ODBC Driver = Installed
```

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

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

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

2.4 Data Access APIs

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

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

- ADO.NET—See Section 2.4.3, "ADO.NET" for more information.

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

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

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

2.4.1 JDBC

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

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

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

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

2.4.2 ODBC

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

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

For more information on ODBC, see the following:

- Microsoft ODBC documentation.
- The Oracle Database Lite ODBC sample application, as described in Section 2.12, "Using Oracle Database Lite Samples".
- Section 11.4.2.1, "Returning Multiple Rows in ODBC".
- Appendix C, "ODBC Support on Palm"

2.4.3 ADO.NET

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

See Chapter 14, "Oracle Database Lite ADO.NET Provider" for a full description.

2.4.4 SODA

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

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

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

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

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

2.5 Oracle Database Lite Multi-User Service

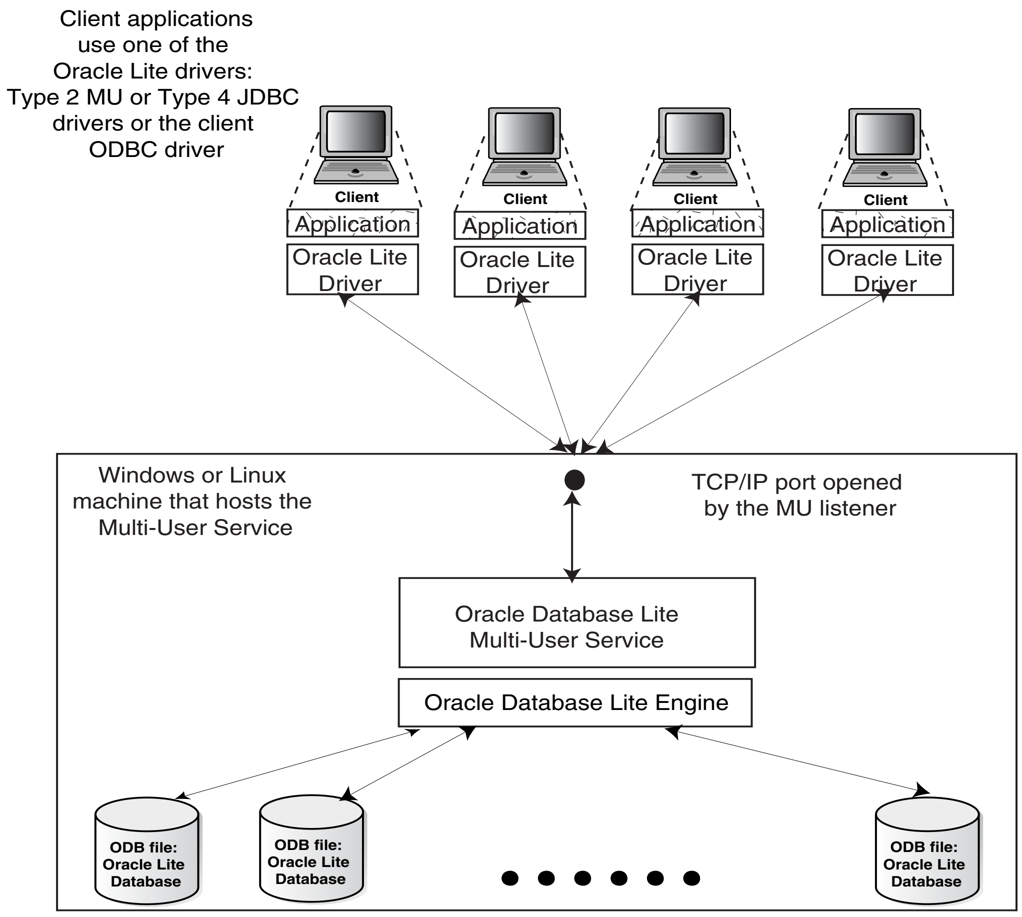
If you want to have multiple users accessing a single entry point for the Oracle Lite database, then use one of the following multi-user services:

- For multiple clients accessing a single Mobile client that synchronizes with a Mobile Server, use the Branch Office service. See Chapter 10, "Manage Your Branch Office" for full details.
- For multiple clients executing an application that accesses the same database, set up a listener to receive requests from each of these clients. See Section 2.5.1, "Accessing the Multi-User Oracle Database Lite Database Service" for full details.

2.5.1 Accessing the Multi-User Oracle Database Lite Database Service

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

Figure 2–1 Diagram of Multi-User Service


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

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

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

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

   The client connection can use the ODBC 2.0 client driver, the JDBC Type 2 MU driver, or the JDBC Type 4 driver. In addition, the client provides the following in the connection string: remote host and port where the multi-user service is listening for incoming calls and the DSN of the Oracle Lite database (ODB file) where the data is stored.

2. Multi-user service receives incoming call with the DSN from the remote client.
3. The multi-user service parses the DSN name and completes the request by connecting to the Oracle Lite database that maps to the DSN name.

The following sections describe how to install and configure the multi-user Oracle Database Lite database service:

- Section 2.5.1.1, "Installation and Configuration"
- Section 2.5.1.2, "Starting the Multi-User Service"

- Section 2.5.1.5, "Debugging the Multi-User Service"
- Section 2.5.1.6, "Creating DSNs"
- Section 2.5.1.7, "Accessing the Database"
- Section 2.5.1.8, "Verifying the Connection Using mSQL"

2.5.1.1 Installation and Configuration

To install and configure the Oracle Database Lite multi-user service, perform the following steps:

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

   `<OLITE_HOME>\Mobile\Sdk\bin`

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

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

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

   ```
   olsv2040.exe /install [/account=AccountName]
                [/wdir=WorkingDirectory] [/port=ServerPort]
   ```
   where the optional parameters can be as follows:

   - `AccountName`: Provide either the `DomainName\UserName` or `.\UserName`.
   - `WorkingDirectory`: If you use '`.`' in SQL scripts that load Java classes, you must specify a working directory.
   - `ServerPort`: Defaults to 1531.

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

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

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

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

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

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

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

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

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

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

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

2.5.1.2 Starting the Multi-User Service

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

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

2.5.1.3 Stopping the Multi-User Service

To stop the multi-user service, use one of the following commands:

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

2.5.1.4 Querying the Multi-User Service

You can query the multi-user status for the following details:

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

Issue the following command to see these details:

```
olsv2040.exe /query
```

The results of this command are as follows:

```
OliteService reports the following status:
  The service is running...
     port= 1531
     wdir = C:\WINDOWS\SYSTEM32

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

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

```
Installed Service startup parameters
  port = 1531
  wdir = \%WINDIR%\SYSTEM32
```

where the port number is 1531 and the working directory is `C:\WINDOWS\SYSTEM32`. The service is installed under the `LocalSystem` account, where the startup parameters for installation are port = 1531 and working directory = `\%WINDIR%\SYSTEM32`. In addition, the Start Type is `SERVICE_AUTO_START` and the binary path as `C:\Oracle\product\Mobile\Sdk\bin\olsv2040.exe`, which is where you installed the Oracle Lite Multi-User Service.

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

2.5.1.5 Debugging the Multi-User Service

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

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

2.5.1.6 Creating DSNs

To access the database using an ODBC or VB application, you must create the DSN enabled from the embedded connection. When you add a DSN using the ODBC Administration tool, choose the **Oracle Lite 40 ODBC Driver(Client)**, which creates a client DSN. If you are executing the service on the same machine where the client application is running, leave the Database Host Name, Database Port Number, and Database Host DSN value empty. The remaining values must be included in the same manner as the 'Oracle Lite ODBC Driver' DSN. If you start the service on a port other than 1531, you must specify the Database Port Number.

2.5.1.7 Accessing the Database

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

2.5.1.8 Verifying the Connection Using mSQL

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

Connect to `A-DSN` on the local host on port 1531.

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

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

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

Connect to `A-DSN` on the local host on port 1531 using the Type4 JDBC driver.

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

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

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

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

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

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

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

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

2.6.2 Oracle Database Lite Load Utility (OLLOAD)

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

2.7 Backing Up an Oracle Lite Database

For either the Mobile client or embedded solutions, you can back up the Oracle Lite database either by using the Oracle database `backupdb` utility or by copying the files to another location. For more information on teh `backupdb` utility, see the Oracle Database documentation.

Oracle Database Lite occupies one file with dependent log files that can be backed up by copying to another location. Before any files can be copied, disconnect all applications that access the database and shut down the multi-user service, if running. Once that has been accomplished, you can copy the `*.odb`, `*.opw`, and `*.plg` files to another directory to make a backup of the database.

2.8 Encrypting a Database

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

If you do not want to use AES encryption, then you can insert your own encryption module to supplant AES; see Section 17.2, "Providing Your Own Encryption Module for the Client Oracle Lite Database" for complete details.

For information on encrypting the database used by the Mobile client, see the `ENCRYPT_DB` parameter in the `POLITE.INI` Appendix in the *Oracle Database Lite Administration and Deployment Guide*; for information on encrypting the database used by an embedded application, see Section B.4, "ENCRYPDB".

2.9 Discover Oracle Lite Database Version Number

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

2.10 Migrate your Oracle Lite Database

If you need to migrate your Oracle Lite database, use the `MIGRATE` utility, which is described in Section B.6, "MIGRATE".

2.11 Support for Linguistic Sort

Linguistic sort is a feature for the ASCII version of Oracle Database Lite. It produces culturally acceptable order of strings for a specified language or collation sequence. The ASCII version supports several code pages defined by single-byte 8-bit encoding schemes. Each of these code pages is a super set of 7-bit ASCII, and the additional accented characters necessary to support a group of European languages are included in the upper 128 bytes. A new string comparison mechanism is provided that produces strings in a linguistically correct order by mapping each collation element of a string to the corresponding 8-bit value of the supported code page.

2.11.1 Creating Linguistic Sort Enabled Databases

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

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

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

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

2.11.2 How Collation Works

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

2.11.3 Collation Element Examples

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

2.11.3.1 Sorting Normal Characters

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

Example 1

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

Example 2

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

Example 3

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

Example 4

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

Example 5

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

2.11.3.2 Reverse Sorting of French Accents

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

Example

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

2.11.3.3 Sorting Contracting Characters

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

Example

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

2.11.3.4 Sorting Expanding Characters

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

2.11.3.5 Sorting Numeric Characters

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

Example

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

2.12 Using Oracle Database Lite Samples

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

Table 2–4 Sample File Directory

Tool	Location of Sample Applications	Description
Blob Manager	<*ORACLE_HOME*>`\Mobile\Sdk\samples\odbc\blob_vb_w32`	Demonstrates the use of the Oracle BLOB datatype and the Visual Basic ODBC programming methods and object manipulation. See Section 2.12.1, "Executing the BLOB Manager Example" for more information.
Java	<*ORACLE_HOME*>`\Mobile\Sdk\samples\jdbc`	Demonstrates programming with JDBC. See Chapter 10, "JDBC Programming" for more information.
ODBC	<*ORACLE_HOME*>`\Mobile\Sdk\samples\odbc\odbc`	Provides ODBC programs written in C.

Table 2–4 (Cont.) Sample File Directory

Tool	Location of Sample Applications	Description
Visual Basic	`<ORACLE_HOME>\Mobile\Sdk\samples\odbc\vb_32`	Demonstrates the ease of querying tables in Oracle Database Lite with Visual Basic tools. See Section 2.12.2, "Executing the Visual Basic Sample Application" for more information.
MFS	`<ORACLE_HOME>\Mobile\Sdk\samples\odbc\mfs`	The MFS sample was documented as a tutorial. See the MFS example used in Chapter 21, "Building Offline Mobile Applications for Win32: A Tutorial".

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

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

- Section 2.12.1, "Executing the BLOB Manager Example"
- Section 2.12.2, "Executing the Visual Basic Sample Application"
- Section 2.12.3, "Executing the ODBC Examples"

2.12.1 Executing the BLOB Manager Example

To install the BLOB Manager example, open the `<ORACLE_HOME>\Mobile\Sdk\samples\odbc\blob_vb_w32\setup` folder and run `setup.exe`. After you complete the installation, click **Start** and select **BLOB Manager** from the Programs menu.

To uninstall the example, perform the following:

1. Click **Start**.
2. Select **Settings**.
3. Select **Control Panel**.
4. Select **Add/Remove**.
5. Select **BLOB Manager**.
6. Click **Add/Remove**.

You need at least version 3.51.2723.0 of `MSJET35.dll` to run the example.

Run the `setup.exe` and BLOB Manager from the Programs menu before you open the Visual Basic project file and execute the project with Visual Basic. Running the program from the Programs menu automatically prepares the table in the database for you.

Note: BLOB Manager is for demonstration purposes. It assumes that you have installed the default database with the default DSN: `POLITE`. If this is not the case, create the `POLITE` DSN using the ODBC Administrator. Also, verify that `SYSTEM` is a valid user for the database.

2.12.2 Executing the Visual Basic Sample Application

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

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

1. Section 2.12.2.1, "Open Visual Basic"
2. Section 2.12.2.2, "View the Sample Application Tables and Data"
3. Section 2.12.2.3, "Open the Sample Application"
4. Section 2.12.2.4, "View and Manipulate the Data in the EMP Table"

2.12.2.1 Open Visual Basic

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

2.12.2.2 View the Sample Application Tables and Data

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

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

Table 2–5 ODBC Logon Dialog Description

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

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

2.12.2.3 Open the Sample Application

1. To open the sample application, select **Open Project** from the File menu.

2. In the dialog box, navigate to <*ORACLE_HOME*>`\Mobile\Sdk\Examples\VB` directory.
3. Select `update.mak`, and click **Open**.

> **Note:** If you do not see the `update.mak` file listed, select **Files** of type *.* to show all file types. You should see the file in the list.

4. From the Run menu, select **Start** to open the sample application and display the `EMP` table.

2.12.2.4 View and Manipulate the Data in the EMP Table

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

2.12.3 Executing the ODBC Examples

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

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

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

The following sections describe the functionality of the samples:

- Section 2.12.3.1, "ODBCTBL"
- Section 2.12.3.2, "ODBCVIEW"
- Section 2.12.3.3, "ODBCFUNC"
- Section 2.12.3.4, "ODBCTYPE"
- Section 2.12.3.5, "LONG"

2.12.3.1 ODBCTBL

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

2.12.3.2 ODBCVIEW

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

2.12.3.3 ODBCFUNC

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

2.12.3.4 ODBCTYPE

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

2.12.3.5 LONG

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

2.13 Limitations of the Oracle Database Lite Engine

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

3 Synchronization

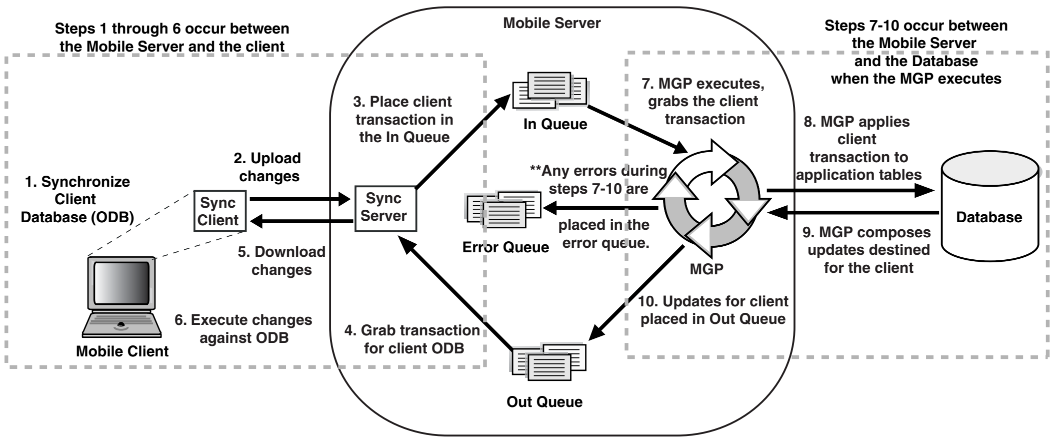
The Oracle Lite database contains a subset of data stored in the Oracle database. This subset is stored in snapshots in the Oracle Lite database. Unlike a base table, a snapshot keeps track of changes made to it in a change log. Users can make changes in the Oracle Lite database and can synchronize them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

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

3.1 How Does Synchronization Work?

The full description of how synchronization works is in the "Managing Synchronization" chapter in the *Oracle Database Lite 10g Administration and Deployment Guide*. Each component and its function is described in the administration guide. The following graphic depicts these components for your reference:

Figure 3–1 Synchronization Architecture


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

1. User initiates a synchronization from the Mobile client. Note that the Mobile client may be a Windows platform client or a PDA.
2. Mobile client software gathers all of the client changes into a transaction and the Sync Client uploads the transaction to the Sync Server on the Mobile Server.
3. Sync Server places the transaction into the In-Queue.
4. Sync Server gathers all transactions destined for the Mobile client from the Out-Queue.
5. Sync Server transfers these transactions down to the Sync Client.
6. Mobile client downloads and applies all changes for client Oracle Lite database.
7. All transactions compiled from all Mobile clients are gathered by the MGP out of the In-Queue.
8. The MGP applies all transactions for the Mobile clients to their respective application tables.
9. Any updates destined for any Mobile client is composed into a transaction by the MGP process.
10. MGP places outgoing transactions for Mobile clients into the Out-Queue, waiting for the next client synchronization for the Sync Server to gather the updates to the client.

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

3.1.1 How Updates Are Propagated to the Back-End Database

The synchronization process applies client operations to the tables in the back-end database according to a table weight assigned to the publication items, as follows:

1. The operations for each publication item are processed according to table weight. The publication creator assigns the table weight to publication items within a specific publication. This value can be an integer between 1 and 1023. For example, a publication can have more than one publication item of weight "2" which would have `INSERT` operations performed after those for any publication item of a lower weight within the same publication. You define the order weight for tables when you add a publication item to the publication. See Section 3.3.1.6.2, "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 lowest to highest table weight order.

For details and an example of exactly how the weights and SQL operations are processed, see Section 3.3.1.6.2, "Using Table Weight".

Note: This order of executing operations can cause constraint violations. See Section 3.9, "Synchronizing With Database Constraints" for more information.

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

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

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

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

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

- Declaratively, using MDW to create the publication and the Packaging Wizard to package and publish the applications. This is the recommended method. See Section 3.2.1, "Creating a Snapshot Definition Declaratively" for details.

- Programmatically, using the Resource Manager and the Consolidator Manager APIs to invoke certain advanced features or customize an implementation. This technique is recommended for advanced users requiring specialized functionality. See Section 3.2.2, "Creating the Snapshot Definition Programmatically" for details.

Once created and subscribed, the user can be synchronized, either from the user initiating it from the device or programmatically from within an application. This chapter discusses how to start the synchronization programmatically in Section 3.7, "Client Synchronization".

On the back-end of the synchronization process, you can customize how the apply and compose phase are executed. See Section 3.5, "Customize the Apply and Compose Phases Using the Consolidator Manager APIs".

3.2.1 Creating a Snapshot Definition Declaratively

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

- Verify that the base tables exist on the server database; if not, create the base table.
- Use MDW to define an application and its publication items (snapshot definitions).
- Use the Packaging Wizard to publish the application to the Mobile Server. This creates the publication items associated with the application. See Chapter 6, "Using the Packaging Wizard" for details.
- Use the Mobile Manager to create a subscription for a given user.
- Install the application on the development machine.
- Synchronize the Mobile client with the Mobile Server to create the client-side snapshots, which creates the Mobile client Oracle Lite database automatically.

See Chapter 5, "Using Mobile Database Workbench to Create Publications" and Chapter 6, "Using the Packaging Wizard" for full details.

3.2.1.1 Manage Snapshots

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

The following sections describes how to manage snapshots using MDW:

- Section 3.2.1.1.1, "Read-only Snapshots"
- Section 3.2.1.1.2, "Updatable Snapshots"
- Section 3.2.1.1.3, "Refresh a Snapshot"
- Section 3.2.1.1.4, "Snapshot Template Variables"

3.2.1.1.1 Read-only Snapshots Read-only snapshots are used for querying purposes. Changes made to the master table are replicated to the snapshot by the Mobile client. See Section 5.6.1, "Associate Publication Item With Publication" for instructions on how to define the publication item as read-only.

3.2.1.1.2 Updatable Snapshots

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/columns 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. See Section 5.6.1, "Associate Publication Item With 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. If the base tables do not have a primary key, a snapshot cannot be updated and becomes read-only.

3.2.1.1.3 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.
- The fast refresh method refreshes the existing data in the snapshot. In general, the simpler your snapshot definition, the faster it is updated.

See Section 5.2, "Create a Publication Item" and Section 3.8, "Understanding Your Refresh Options"

3.2.1.1.4 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
```

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

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

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

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

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

You define this select statement in your publication item. See Section 5.2.1, "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 the "Managing Application Parameter Input (Data Subsetting)" section in the *Oracle Database Lite Administration and Deployment Guide* for information on how to define the value of the variable. This value can only be defined after the application is published and the user is associated with it.

Table 3–1 provides a sample set of snapshot query values specified for separate users.

Table 3–1 Snapshot Query Values for Separate Users

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

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

Table 3–2 provides another sample snapshot query value.

Table 3–2 Snapshot Query Value for User Names

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

3.2.2 Creating the Snapshot Definition Programmatically

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

Note: The Consolidator Manager API can only create a publication, which cannot be packaged with an application. In addition, a publication created with the Consolidator Manager API cannot be packaged with an application. See Section 3.3, "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 Web-to-Go API Specification section, which is located off the *`ORACLE_HOME`*`/Mobile/index.htm` page.

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

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

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

3.3 Creating Publications Using Oracle Database Lite APIs

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

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

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

- Declaratively, using MDW to create the publication and the Packaging Wizard to package and publish the applications. This is the recommended method. This method is described fully in Chapter 5, "Using Mobile Database Workbench to Create Publications" and Chapter 6, "Using the Packaging Wizard".
- Programmatically, using the Consolidator Manager or Resource Manager APIs to invoke certain advanced features or customize an implementation. This technique is recommended for advanced users requiring specialized functionality.
    - Publications created with the Consolidator Manager API cannot be packaged with an application. See Section 3.3.1, "Defining a Publication With Java Consolidator Manager APIs".
    - Use the Resource Manager APIs to create the publication, package it with an application, and publish it to the Mobile Server. See the `oracle.mobile.admin.MobileResourceManager` Javadoc in the API Specification section, which is located off the *`ORACLE_HOME`*`/Mobile/index.htm` page.

3.3.1 Defining a Publication With Java Consolidator Manager APIs

While we recommend that you use MDW (see Chapter 5, "Using Mobile Database Workbench to Create Publications") or the Packaging Wizard (see Chapter 6, "Using the Packaging Wizard") for creating your publications, you can also create them, including the publication items and the user, with the Consolidator Manager API. Choose this option if you are performing more advanced techniques with your publications.

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

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

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

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

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

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

1. Section 3.3.1.1, "Create the Mobile Server User"
2. Section 3.3.1.2, "Create Publications"
3. Section 3.3.1.3, "Create Publication Items"
4. Section 3.3.1.4, "Data Subsetting: Defining Client Subscription Parameters for Publications"
5. Section 3.3.1.5, "Create Publication Item Indexes"
6. Section 3.3.1.6, "Adding Publication Items to Publications"
7. Section 3.3.1.7, "Creating Client-Side Sequences for the Downloaded Snapshot"
8. Section 3.3.1.8, "Subscribing Users to a Publication"
9. Section 3.3.1.9, "Instantiate the Subscription"
10. Section 3.3.1.10, "Bringing the Data From the Subscription Down to the Client"
11. Section 3.3.1.11, "Modifying a Publication Item"
12. Section 3.3.1.12, "Callback Customization for DML Operations"
13. Section 3.3.1.13, "Restricting Predicate"

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

- `<`*`ORACLE_HOME`*`>\jdbc\lib\classes12.zip`
- `<`*`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\classes\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\jewt-nls.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\classes\wtgpack.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\Server\bin\webtogo.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\Server\bin\jzlib.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\Server\bin\aes.jar`
- `<`*`ORACLE_HOME`*`>\Mobile\Server\bin\repository.jar`

3.3.1.1 Create the Mobile Server User

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

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

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

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

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

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

3.3.1.1.1 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
```

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

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

3.3.1.2 Create Publications

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

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

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

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

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

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

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

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

The following code creates a publication item `P_SAMPLE11-M` based on the `ORD_MASTER` database table and adds it to the publication with the `createPublicationItem` method:

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);
```

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

- The view must contain a parent table with a primary key defined.
- `INSTEAD OF` triggers must be defined for data manipulation language (DML) operations on the view. See Section 3.8, "Understanding Your Refresh Options" for more information.
- All base tables of the view must be published.

3.3.1.4 Data Subsetting: Defining Client Subscription Parameters for Publications

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

Creating a Data Subset Example

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

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

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

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

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

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

3.3.1.5 Create Publication Item Indexes

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

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

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

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

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

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

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

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

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

3.3.1.6 Adding Publication Items to Publications

Once you create a publication item, you must associate it with a publication. To change the definition, you can either drop the publication item and then recreate it with the new definition, or use schema evolution depending on your requirements. See `dropPublicationItem` and `alterPublicationItem` respectively in the *Oracle Database Lite API Specification* for more information.

Once you have finished creating the publication items, add it to the desired publication using the `addPublicationItem` method, as follows:

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

3.3.1.6.1 Defining Conflict Rules When adding a publication item to a publication, the user can specify winning rules to resolve synchronization conflicts in favor of either the client or the server. See Section 3.11, "Resolving Conflict Resolution with Winning Rules" for more information.

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

Irregardless of the table weight, the insert, update, and delete client operations are always 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 lowest to highest 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 `dept` are processed.
8. All update operations for `mgr` are processed.
9. All update operations for `emp` are processed.

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

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

3.3.1.7 Creating Client-Side Sequences for the Downloaded Snapshot

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

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

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

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

Note: The sequence name is case-sensitive.

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

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

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

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

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

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

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

3.3.1.8 Subscribing Users to a Publication

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

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

3.3.1.9 Instantiate the Subscription

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

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

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

> **Note:** If you need to set subscription parameters for data subsetting, this must be completed before instantiating the subscription. See Section 3.3.1.4, "Data Subsetting: Defining Client Subscription Parameters for Publications" for more information.

3.3.1.10 Bringing the Data From the Subscription Down to the Client

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

3.3.1.11 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.

Use the `alterPublicationItem` method 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.

```
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.

3.3.1.12 Callback Customization for DML Operations

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

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

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

Table 3–3 Mobile DML Operation Parameters

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

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

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

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

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

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

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

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

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

3.3.1.12.1 DML Procedure Example

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

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

ord_master Table

```
SQL> desc ord_master
Name                                      Null?    Type
----------------------------------------- -------- -------------
ID                                        NOT NULL NUMBER(9)
```

```
DDATE                                               DATE
STATUS                                              NUMBER(9)
NAME                                                VARCHAR2(20)
DESCRIPTION                                         VARCHAR2(20)
```

Code Example

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

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

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

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

3.3.1.13 Restricting Predicate

A restricting predicate can be assigned to a publication item as it is added to a publication.The predicate is used to limit data downloaded to the client. The parameter, which is for advanced use, can be null. When using a restricting predicate, the synchronization uses the high priority replication mode. For using a restricting predicate, see Section 16.1.3, "Priority-Based Replication".

3.4 Client Device Database DDL Operations

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

This is typically used for adding constraints and check values.

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

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

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

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

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

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

3.5 Customize the Apply and Compose Phases Using the Consolidator Manager APIs

The following sections describe how you can customize the apply and compose phases of synchronization:

- Section 3.5.1, "Compose Phase Customization Using MyCompose"
- Section 3.6.2, "Customize What Occurs Before and After Compose/Apply Phases for a Single Publication Item"

3.5.1 Compose Phase Customization Using MyCompose

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

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

1. Section 3.5.1.1, "Create a Class That Extends MyCompose to Perform the Compose"
2. Section 3.5.1.2, "Implement the Extended MyCompose Methods in the User-Defined Class"
3. Section 3.5.1.3, "Use Get Methods to Retrieve Information You Need in the User-Defined Compose Class"
4. Section 3.5.1.4, "Register the User-Defined Class With the Publication Item"

3.5.1.1 Create a Class That Extends MyCompose to Perform the Compose

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

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

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

3.5.1.2 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.

Note: To retrieve information, use the methods described in Section 3.5.1.3, "Use Get Methods to Retrieve Information You Need in the User-Defined Compose Class".

needCompose Method

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

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

The parameters for the `needCompose` method are listed in Table 3–4:

Table 3–4 needCompose Parameters

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

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

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

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

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

This sample uses subsidiary methods discussed in Section 3.5.1.3, "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.

doCompose Method

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

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

The parameters for the `doCompose` method are listed in Table 3–5:

Table 3–5 doCompose Parameters

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

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

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

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

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

        Statement st = conn.createStatement();
        rowCount = st.executeUpdate(sql);
        st.close();
        return rowCount;
    }
```

This code uses subsidiary methods discussed in Section 3.5.1.3, "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.

init Method

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

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

The parameter for the `init` method is described in Table 3–6:

Table 3–6 init Parameters

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

destroy Method

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

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

The parameter for the `destroy` method is described in Table 3–7:

Table 3–7 destroy Parameters

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

3.5.1.3 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.

- getPublication
- getPublicationItem
- getPubItemDMLTableName
- getPubItemPK
- getBaseTables
- getBaseTablePK
- baseTableDirty
- getBaseTableDMLLogName
- getMapView

getPublication

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

```
public String getPublication()
```

getPublicationItem

The `getPublicationItem` method returns the publication item name.

```
public String getPublicationItem()
```

getPubItemDMLTableName

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

```
public String getPubItemDMLTableName()
```

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

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

The parameters for `getPubItemDMLTableName` are listed in Table 3–8:

Table 3–8 getPubItemDMLTableName View Structure Parameters

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

getPubItemPK

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

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

getBaseTables

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
```

getBaseTablePK

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

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

The parameters for `getBaseTablePK` are listed in Table 3–9:

Table 3–9 getBaseTablePK Parameters

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

baseTableDirty

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

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

The parameters for `baseTableDirty` are listed in Table 3–10:

Table 3–10 baseTableDirty Parameters

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

getBaseTableDMLLogName

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

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

The parameters for `getBaseTableDMLLogName` are listed in Table 3–11:

Table 3–11 getBaseTableDMLLogName Parameters

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

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

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

The parameters for `getBaseTableDMLLogName` view structure are listed in Table 3–12:

Table 3–12 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.

getMapView

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

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

The structure of the map table view is as follows:

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

The parameters of the map table view are listed in Table 3–13:

Table 3–13 getMapView View Structure Parameters

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

3.5.1.4 Register the User-Defined Class With the Publication Item

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

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

3.6 Customize What Occurs Before and After Synchronization Phases

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

- Section 3.6.1, "Customize What Occurs Before and After Every Phase of Each Synchronization"
- Section 3.6.2, "Customize What Occurs Before and After Compose/Apply Phases for a Single Publication Item"

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

You can customize 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. If you want to customize certain activity for only a specific publication item, see Section 3.6.2, "Customize What Occurs Before and After Compose/Apply Phases for a Single Publication Item" for more information.

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

- Section 3.6.1.1, "NullSync"
- Section 3.6.1.2, "BeforeProcessApply"
- Section 3.6.1.3, "AfterProcessApply"
- Section 3.6.1.4, "BeforeProcessCompose"
- Section 3.6.1.5, "AfterProcessCompose"
- Section 3.6.1.6, "BeforeProcessLogs"
- Section 3.6.1.7, "AfterProcessLogs"

- Section 3.6.1.8, "BeforeClientCompose"
- Section 3.6.1.9, "AfterClientCompose"
- Section 3.6.1.10, "Example Using the Customize Package"

3.6.1.1 NullSync

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

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

3.6.1.2 BeforeProcessApply

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

```
procedure BeforeProcessApply;
```

3.6.1.3 AfterProcessApply

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

```
procedure AfterProcessApply;
```

3.6.1.4 BeforeProcessCompose

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

```
procedure BeforeProcessCompose;
```

3.6.1.5 AfterProcessCompose

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

```
procedure AfterProcessCompose;
```

3.6.1.6 BeforeProcessLogs

The `BeforeProcessLogs` procedure is called before the logs are generated for the compose phase of the MGP process.

```
procedure BeforeProcessLogs;
```

3.6.1.7 AfterProcessLogs

The `AfterProcessLogs` procedure is called after the logs are generated for the compose phase of the MGP process.

```
procedure AfterProcessLogs;
```

3.6.1.8 BeforeClientCompose

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

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

3.6.1.9 AfterClientCompose

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

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

3.6.1.10 Example Using the Customize Package

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

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

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

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

3.6.2 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. After you create the PL/SQL package and register it (with the publication item), then when the publication item is being processed, MGP calls the appropriate procedures from your package.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
// register SAMPLE3.SAMP3_PKG as the callback for MGP processing of
// P_SAMPLE3 publication item.

cm.createPublicationItem("P_SAMPLE3","SAMPLE3","ADDRESS", "F",
    "SELECT * FROM SAMPLE3.ADDRESS", "SAMPLE3", "SAMP3_PKG");
```

In the previous code example, the following is required:

- `stmt`, which is used when creating the package definition, is an instance of `java.sql.Statement`
- `cm`, which is used when registering the callback package, is an instance of `oracle.lite.sync.ConsolidatorManager`
- The callback package must have the following procedures defined:
  - `BeforeCompose (clientname varchar2);`
  - `AfterCompose (clientname varchar2);`
  - `BeforeApply (clientname varchar2);`
  - `AfterApply (clientname varchar2);`
  - `BeforeTranApply (tranid number);`
  - `AfterTranApply (tranid number);`

3.7 Client Synchronization

You can modify the client-side application to start the synchronization programmatically, which is discussed in the following sections:

- Section 3.7.1, "Performing Synchronization Upload and Download Phases for the Client Using Mobile Sync APIs"
- Section 3.7.2, "Using Mobile Sync for Palm"

3.7.1 Performing Synchronization Upload and Download Phases for the Client Using Mobile Sync APIs

To execute the upload portion of synchronization from the client (see steps 1 and 2 in Figure 3–1) from within your C, C++, or Java application, use the Mobile Sync APIs.

Note: To activate synchronization within a Palm OS application, use the SODA API (see Chapter 12, "Using Simple Object Data Access (SODA) for PalmOS and PocketPC Platforms"). Currently, there are no APIs to perform the upload activity on the UNIX platforms.

To start the upload, perform the following steps:

1. Initialize the synchronization parameters.
2. Set up the transport parameters.
3. Initialize the synchronization options and environment, such as username, password, and selective synchronization.
4. Perform the synchronization.

The following sections demonstrates how you can perform these steps in each of the allowed programming languages:

- Section 3.7.1.1, "Starting Synchronization Upload and Download Phases With the COM Interface"
- Section 3.7.1.2, "Starting Synchronization Upload and Download Phases With C or C++ Applications"
- Section 3.7.1.3, "Starting Synchronization Upload and Download Phases With Java Applications"
- Section 3.7.1.4, "Starting Synchronization Upload and Download Phases With the ADO.NET Provider"

3.7.1.1 Starting Synchronization Upload and Download Phases With the COM Interface

You can perform the upload phase of the synchronization process in your application with the COM interface—which uses the `mSync_com.dll` and `ocapi.dll` libraries. You can use COM interface languages—such as Visual Basic and VBScript—when implementing your application. See Section 4.1, "Synchronization API For the COM Interface" for full details.

3.7.1.2 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 4.2, "Synchronization APIs For C or C++ Applications" for full details.

3.7.1.3 Starting Synchronization Upload and Download Phases With Java Applications

You can initiate and monitor synchronization from a Java client application. See Section 4.3, "Synchronization API for Java Applications" for more information.

3.7.1.4 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 14.1.6, "Data Synchronization With the OracleSync Class" for full details.

3.7.2 Using Mobile Sync for Palm

You can modify how to synchronize data using Mobile Sync (msync) for Palm between the Palm pilot and the Mobile Server. The user can run the application (which can be found in Oracle Database Lite program group) manually and configure various settings. See Section 13.11, "Using Mobile Sync for Palm" for alternate methods to invoke sync with pre-configured settings:

3.8 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.

There are basically three types of refresh for your publication item that can be used to define how to synchronize, as follows:

- Fast Refresh: The most common method of synchronization is a fast refresh publication item where changes are uploaded by the client, and changes for the client are downloaded. Meanwhile, the MGP periodically collects the changes uploaded by all clients and applies them to database tables. It then composes new data, ready to be downloaded to each client during the next synchronization, based on predefined subscriptions.
- Complete Refresh: During a complete refresh, all data for a publication is downloaded to the client. For example, during the very first synchronization session, all data on the client is refreshed from the Oracle database. This form of synchronization takes longer because all rows that qualify for a subscription are transferred to the client device, regardless of existing client data.
- Queue-Based: The developer creates their own queues to handle the synchronization data transfer. This can be considered the most basic form of publication item, for the simple reason that there is no synchronization logic created with it. The synchronization logic is left entirely in the hands of the developer. A queue-based publication item is ideally suited for scenarios that do not require actual synchronization but require something somewhere in between.

For instance, data collection on the client. With data collection, there is no need to worry about conflict detection, client state information, or server-side updates. Therefore, there is no need to add the additional overhead normally associated with a fast refresh or complete refresh publication item.

The following sections describe the refresh options in more detail:

- Section 3.8.1, "Fast Refresh"
- Section 3.8.2, "Complete Refresh for Views"
- Section 3.8.3, "Queue-Based Refresh"

3.8.1 Fast Refresh

Publication items are created for fast refresh by default. Under fast refresh, only incremental changes are replicated. The advantages of fast refresh are reduced overhead and increased speed when replicating data stores with large amounts of data where there are limited changes between synchronization sessions.

The Mobile Server performs a fast refresh of a view if the view meets the following criteria:

- Each of the view base tables must have a primary key.
- All primary keys from all base tables must be included in the view column list.
- If the item is a view, and the item predicate involves multiple tables, then all tables contained in the predicate definition must have primary keys and must have corresponding publication items.

The view requires only a unique primary key for the parent table. The primary keys of other tables may be duplicated. For each base table primary key column, you must provide the Mobile Server with a hint about the column name in the view. You can accomplish this by using the `primaryKeyHint` method of the Consolidator Manager object. See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.8.2 Complete Refresh for Views

A complete refresh is simply a complete execution of the snapshot query. When application synchronization performance is slow, tune the snapshot query. Complete refresh is not optimized for performance. Therefore, to improve performance, use the fast refresh option. The Consperf utility analyzes only fast refresh publication items.

Publication items can be created for complete refresh using the `C` refresh mode in the `createPublicationItem` API from the Consolidator Manager API. When this mode is specified, client data is completely refreshed with current data from the server after every sync. An administrator can force a complete refresh for an entire publication through an API call. This function forces complete refresh of a publication for a given client.

See the Javadoc in the *Oracle Database Lite API Specification* for more information.

The following lists what can cause a complete refresh, ordered from most likely to least likely:

1. The same Mobile user synching from multiple devices on the same platform, or synching from different platforms when the publications are not platform specific.
2. Republishing the application.

3. An unexpected server apply condition, such as constraint violations, unresolved conflicts, and other database exceptions.
4. Modifying the application, such as changing subsetting parameters or adding/altering publication items. This refresh only affects the publication items.
5. A force refresh requested by server administrator or a force refresh requested by the client.
6. Restoring an old Oracle Lite database (ODB file).
7. Two separate applications using the same backend store.
8. An unexpected client apply conditions, such as a moved or deleted database, database corruption, memory corruption, other general system failures.
9. Loss of transaction integrity between the server and client. The server fails post processing after completing the download and disconnects from the client.
10. Data transport corruptions.

3.8.3 Queue-Based Refresh

You can create your own queue. Mobile Server uploads and downloads changes from the user. Perform customized apply/compose modifications to the back-end database with your own implementation. See the Section 3.16, "Customizing Replication With Your Own Queues" for more information.

3.9 Synchronizing With Database Constraints

When you have database constraints on your table, you must develop your application in a certain way to facilitate the synchronization of the data and keeping the database constraints. The following sections detail each constraint and what issues you must take into account:

- Section 3.9.1, "Synchronization And Database Constraints"
- Section 3.9.2, "Primary Key is Unique"
- Section 3.9.3, "Foreign Key Constraints"
- Section 3.9.4, "Unique Key Constraint"
- Section 3.9.5, "Not Null Constraint"

3.9.1 Synchronization And Database Constraints

Oracle Database Lite does not keep a record of the SQL operations executed against the database; instead, only the final changes are saved and synchronized to the back-end database.

For example, if you have a client with a unique key constraint, where the following is executed against the client Oracle Lite database:

1. Record with primary key of one and unique field of ABC is deleted.
2. Record with primary key of 4 and unique field of ABC is inserted.

When this is synchronized, according the Section 3.3.1.6.2, "Using Table Weight" discussion, the insert is performed before the delete. This would add a duplicate field for ABC and cause a unique key constraint violation. In order to avoid this, you should defer all constraint checking until after all transactions are applied. See Section 3.9.3.2, "Defer Constraint Checking Until After All Transactions Are Applied".

Another example of how synchronization captures the end result of all SQL modifications is as follows:

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

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

3.9.2 Primary Key is Unique

When you have multiple clients, each updating the same table, you must have a method for guaranteeing that the primary key is unique across all clients. Oracle Database Lite provides you a sequence number that you can use as the primary key, which is guaranteed to be unique across all Oracle Database Lite clients.

For more information on the sequence number, see Section 3.3.1.7, "Creating Client-Side Sequences for the Downloaded Snapshot".

3.9.3 Foreign Key Constraints

A foreign key exists in a details table and points to a row in the master table. Thus, before a client adds a record to the details table, the master table must first exist.

For example, two tables `EMP` and `DEPT` have referential integrity constraints and are an example of a master-detail relationship. The `DEPT` table is the master table; the `EMP` table is the details table. The `DeptNo` field (department number) in the `EMP` table is a foreign key that points to the `DeptNo` field in the `DEPT` table. The `DeptNo` value for each employee in the `EMP` table must be a valid `DeptNo` value in the `DEPT` table.

When a user adds a new employee, first the employee's department must exist in the `DEPT` table. If it does not exist, then the user first adds the department in the `DEPT` table, and then adds a new employee to this department in the `EMP` table. The transaction first updates `DEPT` and then updates the `EMP` table. However, Oracle Database Lite does not store the sequence in which these operations were executed.

Oracle Database Lite does not keep a record of the SQL operations executed against the database; instead, only the final changes are saved and synchronized to the back-end database. For our employee example, when the user replicates with the Mobile Server, the Mobile Server could initiate the updates the `EMP` table first. If this occurs, then it attempts to create a new record in `EMP` with an invalid foreign key value for `DeptNo`. Oracle database detects a referential integrity violation. The Mobile Server rolls back the transaction and places the transaction data in the Mobile Server error queue. In this case, the foreign key constraint violation occurred because the operations within the transaction are performed out of their original sequence.

In order to avoid this violation, you can do one of two things:

- Section 3.9.3.1, "Set Update Order for Tables With Weights"
- Section 3.9.3.2, "Defer Constraint Checking Until After All Transactions Are Applied"

3.9.3.1 Set Update Order for Tables With Weights

Set the order in which tables are updated on the back-end Oracle database with weights. To avoid integrity constraints with a master-details relationship, the master table must always be updated first in order to guarantee that it exists before any records are added to a details table. In our example, you must set the `DEPT` table with a lower weight than the `EMP` table to ensure that all records are added to the `DEPT` table first.

You define the order weight for tables when you add a publication item to the publication. For more information on weights, see Section 3.3.1.6.2, "Using Table Weight".

3.9.3.2 Defer Constraint Checking Until After All Transactions Are Applied

You can use a PL/SQL procedure avoid foreign key constraint violations based on out-of-sequence operations by using `DEFERRABLE` constraints in conjunction with the `BeforeApply` and `AfterApply` functions. `DEFERRABLE` constraints can be either `INITIALLY IMMEDIATE` or `INITIALLY DEFERRED`. The behavior of `DEFERRABLE INITIALLY IMMEDIATE` foreign key constraints is identical to regular immediate constraints. They can be applied interchangeably to applications without impacting functionality.

The Mobile Server calls the `BeforeApply` function before it applies client transactions to the server and calls the `AfterApply` function after it applies the transactions. Using the `BeforeApply` function, you can set constraints to `DEFFERED` to delay referential integrity checks. After the transaction is applied, call the `AfterApply` function to set constraints to `IMMEDIATE`. At this point, if a client transaction violates referential integrity, it is rolled back and moved into the error queues.

To prevent foreign key constraint violations using `DEFERRABLE` constraints:

1. Drop all foreign key constraints and then recreate them as `DEFERRABLE` constraints.
2. Bind user-defined PL/SQL procedures to publications that contain tables with referential integrity constraints.
3. The PL/SQL procedure should set constraints to `DEFERRED` in the `BeforeApply` function and `IMMEDIATE` in the `AfterApply` function as in the following example featuring a table named `SAMPLE3` and a constraint named `address.14_fk`:

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

3.9.4 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 3.3.1.7, "Creating Client-Side Sequences for the Downloaded Snapshot" for more information.

3.9.5 Not Null Constraint

When you have a not null constraint on the client or on the server, you must ensure that this constraint is also on the table on the other side. If you execute the `setPubItemColOption` method in the `ConsolidatorManager` API, you can set a column as not null—whether it is set with the not null constraint in the server table or not. Provide `Consolidator.NOT_NULL` as the input parameter for `nullType`. The constraint is then enforced on the table in the client Oracle Lite database. It is not enforced in the server back-end database tables. You must define this constraint explicitly on the server table.

3.10 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

3.10.1 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.

3.10.2 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.

3.11 Resolving Conflict Resolution with Winning Rules

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:

- 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 does not automatically resolve synchronization errors. Instead, the Mobile Server rolls back the corresponding transactions, and moves the transaction operations into the Mobile Server error queue. It is up to the administrator to view the error queue and determine if the correct action occurred. If not, the administrator must correct and re-execute the transaction. If it did execute correctly, then purge the transaction from the error queue.

One type of error is a synchronization conflict, which is detected in any of the following situations:

- The client and the server update the same row.
- The client deletes the same row that the server updates.
- The client updates a row at the same time that the server deletes it when the "server wins" conflict rule is specified. This is considered a synchronization error for compatibility with Oracle database advanced replication.
- Both the client and server create rows with the same primary key values.
- For systems with delayed data processing, where the client data is not directly applied to the base table—for instance, in a three-tiered architecture—a situation could occur when a client inserts a row and then updates the same row, while the row has not yet been inserted into the base table. In that case, if the `DEF_APPLY` parameter in `C$ALL_CONFIG` is set to `TRUE`, an `INSERT` operation is performed, instead of the `UPDATE`. It is up to the application developer to resolve the resulting primary key conflict. If, however, `DEF_APPLY` is not set, a "`NO DATA FOUND`" exception is thrown.

All the other errors, including nullity violations and foreign key constraint violations are synchronization errors. See Section 3.9, "Synchronizing With Database Constraints" for more information.

All synchronization errors are placed into the error queue. For each publication item created, a separate and corresponding error queue is created. The purpose of this queue is to store transactions that fail due to unresolved conflicts. The administrator can attempt to resolve the conflicts, either by modifying the error queue data or that of the server, and then attempt to re-apply the transaction.

The administrator can change the transaction operations and re-execute or purge transactions from the error queue from either of the following:

- The Mobile Manager GUI—See the "Managing Synchronization" chapter in the *Oracle Database Lite Administration and Deployment Guide* on how to update the client transaction in the error queue and re-execute the statement using the Mobile Manager GUI.
- The Consolidator Manager API and accessing the Mobile Server error queue tables directly and customize the conflict rules, as described in the following sections:
  - Section 3.11.1, "Resolving Conflicts Using the Error Queue"
  - Section 3.11.2, "Customizing Synchronization Conflict Resolution Outcomes"

3.11.1 Resolving Conflicts Using the Error Queue

The error queue stores transactions that fail due to unresolved conflicts. The administrator can do one of the following:

- Attempt to resolve the conflict 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 the conflict was resolved to your satisfaction, then you can purge the transaction from the error queue with the `purgeTransaction` method of the Consolidator Manager object.

The data for the error queue for each base table is stored in `CEQ$<base_table_name>`; the Mobile Server error queue is stored in `C$EQ<base_table_name>`.

When you view the error queue, either on the database or through the Mobile Manager GUI, you can see what the conflict was that caused the data to not apply. If you determine what the problem is, you can reapply the data by modifying the DML operation appropriately and then re-executing.

For example, if you have a record that did not update because the update occurred just before a crash, the DML operation will read as `ERROR`.

1. On the database, modify the DML operation from Error (E) to Update (U).
2. View the modified DML operation in the error queue.
3. Re-apply the transaction with the `executeTransaction` method. The operation will be applied again. If there is a problem, the operation is placed back into the error queue.

The `executeTransaction` method re-executes transactions in the error queue, as follows:

```
consolidatorManager.executeTransaction("DAVIDL", 100002);
```

The `purgeTransaction` method purges a transaction from the Mobile Server error queue. You would eliminate a transaction from the error queue if you are content with how it was resolved or if you do have already taken care of the incident that caused the error in the first place.

```
consolidatorManager.purgeTransaction("DAVIDL", 100001);
```

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.11.2 Customizing Synchronization Conflict Resolution Outcomes

You can customize synchronization conflict resolution by performing the following:

1. Configure the winning rule to Client Wins.
2. Attach `BEFORE INSERT`, `UPDATE`, and `DELETE` triggers to database tables.
3. Create a custom DML procedure.

The triggers in the database compare old and new row values and resolve client changes, as you specify in the triggers.

3.12 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 3.12.1, "Creating Secondary Indexes on Client Device"
- Section 3.12.2, "Virtual Primary Key"

3.12.1 Creating Secondary Indexes on Client Device

The first time a client synchronizes, the Mobile Server automatically enables a Mobile client to create the database objects on the client in the form of snapshots. By default, the primary key index of a table is automatically replicated from the server. You can create secondary indexes on a publication item through the Consolidator Manager APIs. See the *Oracle Database Lite API Javadoc* for specific API information. See Section 3.3.1.5, "Create Publication Item Indexes" for an example.

3.12.2 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 can be created for more than one column, but the API must be called separately for each column that you wish to assign a virtual primary key. The following methods create and drop a virtual primary key.

Use the `createVirtualPKColumn` method to create a virtual primary key column.

```
consolidatorManager.createVirtualPKColumn("SAMPLE1", "DEPT", "DEPT_ID");
```

Use the `dropVirtualPKColumns` method to drop a virtual primary key.

```
consolidatorManager.dropVirtualPKColumns("SAMPLE1", "DEPT");
```

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.13 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.OPER` type, then the privileges needed are those necessary for executing the Mobile Server without any schema modifications. The OPER is given 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.

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

Publication items can be defined for database objects existing on remote database instances outside of the Mobile Server repository. Local private synonyms of the remote objects should be created in the Oracle database. Execute the following SQL script located in the `<`*`ORACLE_HOME`*`>\Mobile\server\admin\consolidator_rmt.sql` directory on the remote schema in order to create Consolidator Manager logging objects.

The synonyms should then be published using the `createPublicationItem` method of the `ConsolidatorManager` object. If the remote object is a view that needs to be published in updatable mode and/or fast-refresh mode, the remote parent table must also be published locally. Parent hints should be provided for the synonym of the remote view similar those used for local, updatable and/or fast refreshable views.

Two additional methods have been created, `dependencyHint` and `removeDependencyHint`, to deal with non-apparent dependencies introduced by publication of remote objects.

Remote links to the Oracle database must be established prior to attempting remote linking procedures, please refer to the *Oracle SQL Reference* for this information.

Note: The performance of synchronization from remote databases is subject to network throughput and the performance of remote query processing. Because of this, remote data synchronization is best used for simple views or tables with limited amount of data.

The following sections describe how to manage remote links:

- Section 3.14.1, "Publishing Synonyms for the Remote Object Using CreatePublicationItem"
- Section 3.14.2, "Creating or Removing a Dependency Hint"

3.14.1 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`.

3.14.2 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.

3.15 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:

3.15.1 getDownloadInfo Method

The `getDownloadInfo` method returns the `DownloadInfo` object. The `DownloadInfo` object contains a set of `PublicationSize` objects and access methods. The `PublicationSize` objects carry the size information of a publication item. The method `Iterator iterator()` can then be used to view each `PublicationSize` object in the `DownloadInfo` object.

```
DownloadInfo dl = consolidatorManager.getDownloadInfo("S11U1", true, true);
```

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.15.2 DownloadInfo Class Access Methods

The access methods provided by the `DownloadInfo` class are listed in Table 3–14:

Table 3–14 DownloadInfo Class Access Methods

Method	Definition
`iterator`	Returns an Iterator object so that the user can traverse through the all the `PublicationSize` objects that are contained inside the `DownloadInfo` object.
`getTotalSize`	Returns the size information of all `PublicationSize` objects in bytes, and by extension, the size of all publication items subscribed to by that user. If no historical information is available for those publication items, the value returned is '-1'.
`getPubSize`	Returns the size of all publication items that belong to the publication referred to by the string `pubName`. If no historical information is available for those publication items, the value returned is '-1'.
`getPubRecCount`	Returns the number of all records of all the publication items that belong to the publication referred by the string `pubName`, that will be synchronization during the next synchronization.
`getPubItemSize`	Returns the size of a particular publication item referred by `pubItemName`. It follows the following rules in order. **1.** If the publication item is empty, it will return '0'. **2.** If no historical information is available for those publication items, it will return '-1'.
`getPubItemRecCount`	Returns the number of records of the publication item referred by `pubItemName` that will be synced in the next synchronization.

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.15.3 PublicationSize Class

The access methods provided by the `PublicationSize` class are listed inTable 3–15:

Table 3–15 PublicationSize Class Access Methods

Parameter	Definition
`getPubName`	Returns the name of the publication containing the publication item.
`getPubItemName`	Returns the name of the publication item referred to by the `PublicationSize` object.
`getSize`	Returns the total size of the publication item referred to by the `PublicationSize` object.
`getNumOfRows`	Returns the number of rows of the publication item that will be synchronized in the next synchronization.

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

Sample Code

```
import   java.sql.*;
import   java.util.Iterator;
import   java.util.HashSet;

import   oracle.lite.sync.ConsolidatorManager;
import   oracle.lite.sync.DownloadInfo;
import   oracle.lite.sync.PublicationSize;

public class TestGetDownloadInfo
{

   public static void main(String argv[]) throws Throwable
   {
// Open Consolidator Manager connection
      try
      {
// Create a ConsolidatorManager object
         ConsolidatorManager cm = new ConsolidatorManager ();
// Open a Consolidator Manager connection
         cm.openConnection ("MOBILEADMIN", "MANAGER",
                    "jdbc:oracle:thin:@server:1521:orcl", System.out);
// Call getDownloadInfo
         DownloadInfo dlInfo = cm.getDownloadInfo ("S11U1", true, true);
// Call iterator for the Iterator object and then we can use that to transverse
// through the set of PublicationSize objects.
         Iterator it = dlInfo.iterator ();
// A temporary holder for the PublicationSize object.
         PublicationSize ps = null;
// A temporary holder for the name of all the Publications in a HashSet object.
         HashSet pubNames = new HashSet ();
// A temporary holder for the name of all the Publication Items in a HashSet
// object.
         HashSet pubItemNames = new HashSet ();
// Traverse through the set.
         while (it.hasNext ())
         {
```

```
// Obtain the next PublicationSize object by calling next ().
            ps = (PublicationSize)it.next ();

// Obtain the name of the Publication this PublicationSize object is associated
// with by calling getPubName ().
            pubName = ps.getPubName ();
            System.out.println ("Publication: " + pubName);

// We save pubName for later use.
            pubNames.add (pubName);

// Obtain the Publication name of it by calling getPubName ().
            pubItemName = ps.getPubItemName ();
            System.out.println ("Publication Item Name: " + pubItemName);

// We save pubItemName for later use.
            pubItemNames.add (pubItemName);

// Obtain the size of it by calling getSize ().
            size = ps.getSize ();
            System.out.println ("Size of the Publication: " + size);

// Obtain the number of rows by calling getNumOfRows ().
            numOfRows = ps.getNumOfRows ();
            System.out.println ("Number of rows in the Publication: "
                                + numOfRows);
         }

// Obtain the size of all the Publications contained in the
// DownloadInfo objects.
         long totalSize = dlInfo.getTotalSize ();
         System.out.println ("Total size of all Publications: " + totalSize);

// A temporary holder for the Publication size.
         long pubSize = 0;

// A temporary holder for the Publication number of rows.
         long pubRecCount = 0;

// A temporary holder for the name of the Publication.
         String tmpPubName = null;

// Transverse through the Publication names that we saved earlier.
         it = pubNames.iterator ();
         while (it.hasNext ())
         {
// Obtain the saved name.
            tmpPubName = (String) it.next ();

// Obtain the size of the Publication.
            pubSize = dlInfo.getPubSize (tmpPubName);
            System.out.println ("Size of " + tmpPubName + ": " + pubSize);

// Obtain the number of rows of the Publication.
            pubRecCount = dlInfo.getPubRecCount (tmpPubName);
            System.out.println ("Number of rows in " + tmpPubName + ": "
                                + pubRecCount);
         }

// A temporary holder for the Publication Item size.
```

```
         long pubItemSize = 0;

// A temporary holder for the Publication Item number of rows.
         long pubItemRecCount = 0;

// A temporary holder for the name of the Publication Item.
         String tmpPubItemName = null;

// Traverse through the Publication Item names that we saved earlier.
         it = pubItemNames.iterator ();
         while (it.hasNext ())
         {
// Obtain the saved name.
            tmpPubItemName = (String) it.next ();

// Obtain the size of the Publication Item.
            pubItemSize = dlInfo.getPubItemSize (tmpPubItemName);
            System.out.println ("Size of " + pubItemSize + ": " + pubItemSize);

// Obtain the number of rows of the Publication Item.
            pubItemRecCount = dlInfo.getPubItemRecCount (tmpPubItemName);
            System.out.println ("Number of rows in " + tmpPubItemName + ": "
                                + pubItemRecCount);
         }
         System.out.println ();

// Close the connection
         cm.closeConnection ();
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

3.16 Customizing Replication With Your Own Queues

Application developers can manage the replication process programmatically by using queue-based publication items, as described in Section 3.16.4, "Create a Publication Item as a Queue". Normally the MGP manages both the in queues and the out queues, this API allows the application developer to manage queue operations during a synchronization session using a PL/SQL package described in Section 3.16.3, "Queue Interface PL/SQL Procedure" and by creating the queues themselves.

3.16.1 Queue Interface Operation

When data arrives from the client it is placed in the publication item in queues. The Sync Server calls `UPLOAD_COMPLETE` once the data has been committed. All records in the current synchronization session are given the same transaction identifier. Data Synchronization has a Queue Control Table (`C$INQ`) that indicates which publication item in queues have received new transactions using this transaction identifier. You can refer to this table to determine which queues need processing.

Before the Sync Server begins the download phase of the synchronization session, it calls `DOWNLOAD_INIT`. This procedure allows customization of any settings which need to be set or modified to determine which data is sent to 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 which items will be downloaded to the client.

Similar to the in queue, every record in the out queue should be associated with it a transaction identifier (TRANID$$). The Sync Server passes the `last_tran` parameter to indicate the last transaction that the client has successfully applied. New out queue records which have not been downloaded to the client before should be marked with the value of `curr_tran` parameter. The value of `curr_tran` is always greater than that of `last_tran`, though not necessarily sequential. The Sync Server only downloads records from the out queues when the value of TRANID$$ is greater than last_tran. When the data is downloaded, the Sync Server calls DOWNLOAD_COMPLETE.

3.16.2 Queue Creation

You need to create the out queue in the Mobile Server repository manually using SQL. You may also wish to create the in queue as well although the Sync Server creates this if one does not exist. Connect to your repository and execute the following statements to create in queues and out queues with the following structure:

Note: The name that you provide in the `+name` field is the name of the queue-based publication item.

Out queue

```
'CTM$'+name
(
CLID$$CS   VARCHAR2 (30),
..
publication_item_store_columns (c1..cN),
..
TRANID$$   NUMBER (10),
DMLTYPE$$  CHAR (1) CHECK (DMLTYPE$$  IN ('I','U','D'),
)
```

In queue

```
'CFM$'+name
(
CLID$$CS   VARCHAR2 (30),
TRANID$$   NUMBER (10),
SEQNO$$    NUMBER (10),

DMLTYPE$$  CHAR (1) CHECK (DMLTYPE$$  IN ('I','U','D'),
..
publication_item_store_columns (c1..cN),
..
)
```

The Sync Server creates a queue control table, C$INQ, and a temporary table, C$PUB_LIST_Q. You can examine the queue control table to determine which publication items have received new transactions.

Queue Control Table

```
'C$INQ'
(
CLIENTID   VARCHAR2 (30),
TRANID$$   NUMBER,
STORE      VARCHAR2 (30),

)
```

Temporary Table

```
'C$PUB_LIST_Q'
(
NAME   VARCHAR2 (30),
COMP_REF   CHAR(1),
CHECK(COMP_REF IN('Y','N'))

)
```

The parameters for the manually created queues are listed in Table 3–16:

Table 3–16 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 Outqueue only.
STORE	Represents the publication item name in the queue control table (C$INQ) only.
NAME	The publication item name in the temporary table (C$PUB_LIST_Q) only.
COMP_REF	This value is either 'Y' for yes, or 'N' for no and is a flag used for determining the refresh mode of publication items.

3.16.3 Queue Interface PL/SQL Procedure

The following PL/SQL package specification defines the callouts needed by the queue interface:

Sample Code

```
CREATE OR REPLACE PACKAGE CONS_QPKG AS
/*
 *     notifies that inq has new transaction
*/
PROCEDURE UPLOAD_COMPLETE(
     CLIENTID      IN     VARCHAR2,
     TRAN_ID      IN     NUMBER     -- IN queue tranid
     );
/*
 *     init data for download
*/
PROCEDURE DOWNLOAD_INIT(
     CLIENTID      IN     VARCHAR2,
     LAST_TRAN      IN      NUMBER,
     CURR_TRAN      IN      NUMBER,
     HIGH_PRTY      IN      VARCHAR2
     );
/*
 *  notifies when all the client's data is sent
*/
PROCEDURE DOWNLOAD_COMPLETE(
     CLIENTID      IN      VARCHAR2
     );

END CONS_QPKG;
/
```

3.16.4 Create a Publication Item as a Queue

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 of the 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 or view name.

3.16.5 Defining a PL/SQL Package Outside the Repository

The PL/SQL package can be defined outside of the Mobile Server repository; although it must refer 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, as follows:

  ```
  consolidatorManager.registerQueuePkg("ASL.QUEUES_PKG");
  ```

- The `getQueuePkg` method returns the name of the currently registered procedure.
- The `unRegisterQueuePkg` method removes the currently registered procedure.

Note: See the Javadoc in the *Oracle Database Lite API Specification* for more information.

3.17 Synchronization Performance

There are certain optimizations you can do to increase performance. See Section 16.1, "Increasing Synchronization Performance" for a full description.

3.18 Troubleshooting Synchronization Errors

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

- Section 3.18.1, "Foreign Key Constraints in Updatable Publication Items"

3.18.1 Foreign Key Constraints in Updatable Publication Items

Replicating tables between Oracle database and clients in updatable mode can result in foreign key constraint violations if the tables have referential integrity constraints. When a foreign key constraint violation occurs, the server rejects the client transaction.

- Section 3.18.1.1, "Foreign Key Constraint Violation Example"
- Section 3.18.1.2, "Avoiding Constraint Violations with Table Weights"
- Section 3.18.1.3, "Avoiding Constraint Violations with BeforeApply and After Apply"

3.18.1.1 Foreign Key Constraint Violation Example

For example, two tables `EMP` and `DEPT` have referential integrity constraints. The `DeptNo` (department number) attribute in the `DEPT` table is a foreign key in the `EMP` table. The `DeptNo` value for each employee in the `EMP` table must be a valid `DeptNo` value in the `DEPT` table.

A Mobile Server user adds a new department to the `DEPT` table, and then adds a new employee to this department in the `EMP` table. The transaction first updates `DEPT` and then updates the `EMP` table. However, the database application does not store the sequence in which these operations were executed.

When the user replicates with the Mobile Server, the Mobile Server updates the `EMP` table first. In doing so, it attempts to create a new record in `EMP` with an invalid foreign key value for `DeptNo`. Oracle database detects a referential integrity violation. The Mobile Server rolls back the transaction and places the transaction data in the Mobile Server error queue. In this case, the foreign key constraint violation occurred because the operations within the transaction are performed out of their original sequence.

Avoid this violation by setting table weights to each of the tables in the master-detail relationship. See Section 3.18.1.2, "Avoiding Constraint Violations with Table Weights" for more information.

3.18.1.2 Avoiding Constraint Violations with Table Weights

Mobile Server uses table weight to determine in which order to apply client operations to master tables. Table weight is expressed as an integer and are implemented as follows:

1. Client `INSERT` operations are executed first, from lowest to highest table weight order.
2. Client `DELETE` operations are executed next, from highest to lowest table weight order.
3. Client `UPDATE` operations are executed last, from lowest to highest table weight order.

In the example listed in Section 3.18.1.1, "Foreign Key Constraint Violation Example", a constraint violation error could be resolved by assigning `DEPT` a lower table weight than `EMP`. For example:

```
(DEPT weight=1, EMP weight=2)
```

You define the order weight for tables when you add a publication item to the publication. For more information on setting table weights in the publication item, see Section 3.3.1.6.2, "Using Table Weight".

3.18.1.3 Avoiding Constraint Violations with BeforeApply and After Apply

You can use a PL/SQL procedure avoid foreign key constraint violations based on out-of-sequence operations by using `DEFERRABLE` constraints in conjunction with the `BeforeApply` and `AfterApply` functions. See Section 3.9.3.2, "Defer Constraint Checking Until After All Transactions Are Applied" for more information.

3.19 Datatype Conversion Between the Oracle Server and Client Oracle Lite Database

Before you publish your application, you create the tables for your applications in the Oracle database. Thus, when the first synchronization occurs, Oracle Database Lite takes the Oracle database datatypes and converts them to corresponding allowed datatypes in the Oracle Lite database on the client. Table 3–17 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 B in the *Oracle Database Lite SQL Reference*.

Table 3–17 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**
Oracle Database Datatypes												
INTEGER			X									
VARCHAR2									X			
VARCHAR									X			
CHAR										X		
SMALLINT		X										
FLOAT				X								
DOUBLE PRECISION					X							
NUMBER	X	X	X	X	X	X						
DATE							X					
LONG RAW								X				
LONG									X			
BLOB											X	
CLOB												X

"X" indicates that the datatype can be mapped to this Oracle Lite database datatype. To save on space, 1 B represents **TINYINT**, 2 B represents **SMALLINT**, and 4 B represents **INTEGER**.

4 Invoking Synchronization APIs from Applications

The following sections describe the APIs available to start synchronization programmatically within your application, whether the application is COM, C, C++, or Java:

- Section 4.1, "Synchronization API For the COM Interface"
- Section 4.2, "Synchronization APIs For C or C++ Applications"
- Section 4.3, "Synchronization API for Java Applications"
- Section 4.4, "msync/OCAPIs/mSyncCom"

4.1 Synchronization API For the COM Interface

You can perform the upload phase of the synchronization process in your application with the COM interface—which uses the `mSync_com.dll` and `ocapi.dll` libraries. You can use COM interface languages—such as Visual Basic and VBScript—when implementing your application.

The interface is contained in the msync library. Use `MSync.ISync` as the interface name.

The COM Interface supports the following synchronization features:

1. Setup client-side user profiles with data such as user name, password, and server.
2. Assign table level synchronization options.
3. Choose the type of transport.
4. Enable users to start the synchronization process.
5. Track progress of the synchronization.

Note: The COM Interface API and samples are installed in the *`ORACLE_HOME`*`\Mobile\SDK\Examples\mysncCom` subdirectory in the `mSync_com.dll` library:

The following sections discuss how to implement the upload phase of synchronization using the COM Interface:

- Section 4.1.1, "COM Interface ISyncOption Interface"
- Section 4.1.2, "COM Interface SyncParam Settings"

- Section 4.1.3, "COM Interface TransportParam Parameters"
- Section 4.1.4, "Selective Synchronization"
- Section 4.1.5, "ISyncProgressListener Interface"
- Section 4.1.6, "Initiate Synchronization with the ISync Interface"

4.1.1 COM Interface ISyncOption Interface

The `ISyncOption` interface contains the `MSync.SyncOption` class, which defines the parameters of the synchronization process. Table 4–1 lists the `SyncOption` public methods.

Table 4–1 ISyncOption Public Methods

Name	Description
`void load()`	Loads the profile of the last synchronization user.
`void save()`	Saves settings to the user profile.
`void getPublication (BSTR app_name, BSTR * pub_name)`	Uses the Web-to-Go application name and returns the publication name.
`void setSyncFlag(BSTR pub_name, BSTR tbl_name, short syncFlag)`	Sets selective sync on table level. You must perform an intial synchronization before you can issue a selective sync. Passing `pub_name`, `tbl_name`=null, `syncFlag` = 0 turns off `syncFlag` for every table in that publication. Passing `pub_name`, `tbl_name`, `syncFlag` = 1 turns on `syncFlag` for that specific table.

Table 4–2 lists the `ISyncOption` public properties.

Table 4–2 ISyncOption Public Properties

Name	Description
`username`	Name of the user.
`password`	User password.
`syncParam`	A name/value string that designates the synchronization preferences. For information on how to set up the string, see Section 4.1.2, "COM Interface SyncParam Settings".
`transportType`	Type of transport to use. Currently, only `HTTP` is supported; however, if you set the security to SSL in `syncParam`, `HTTPS` is used under the covers.
`transportParam`	A name/value string that designates the transport to the Mobile Server. See Section 4.1.3, "COM Interface TransportParam Parameters" for information on how to set up this string.
`BSTR app_name(in)`	Web-to-Go application name.
`BSTR& pub_name(out)`	Publication name.

The following Visual Basic code demonstrates how to start a synchronization session using default settings.

Note: On Windows CE, the `ISyncOption` interface object must be Dim'ed as follows:

```
Dim syncOpt as MSync.SyncOption
```

```
Set syncOpt = CreateObject("MSync.SyncOption")
' Load last sync info
syncOpt.load
' Change user name to Sam
syncOpt.username = "Sam"
Set sync = CreateObject("MSync.Sync")
' Tell ISync to use this option
sync.setOptionObject (syncOpt)
' Do sync
sync.DoSync
```

For information on the `Sync` object used in this example, see Section 4.1.6, "Initiate Synchronization with the ISync Interface".

4.1.2 COM Interface SyncParam Settings

You can set support parameters for the synchronization session by providing a string that consists of name or name/value pairs in the `SyncParam` object.

The structure of the string with the name/value pairs is as follows:

```
"name=value;name2=value2;name3=value3, ...;"
```

The names are not case sensitive, but the values are. The names that can be used are listed in Table 4–3.

Table 4–3 COM Interface SyncParam Settings

Name	Value/Options	Description
`reset`	N/A	Clears all entries in the environment before applying any remaining settings.
`security`	`SSL`	Choose either SSL encryption or AES stream encryption.
`pushonly`	N/A	Upload changes from the client to the server only, as download is not allowed. This is useful when the data transfer is a one way transmission from the client to server.
`noapps`	N/A	Do not download any new or updated applications. This is useful when synchronizing over a slow connection or on a slow network.
`syncDirection`	`SendOnly`	`SendOnly` is the same as `pushonly`.
	`ReceiveOnly`	`ReceiveOnly` allows no changes to be posted to the server. You must implement your own transport for this setting.
`noNewPubs`	N/A	The server will not send any new publications that were created since the last synchronization; only data from current publications are updated.
`fullrefresh`	N/A	Forces a complete refresh of all data for all publications.

For example, the following `SyncParam` string enables SSL security and disables application deployment for the current synchronization session:

```
syncOpt.syncParam = "security=SSL; noapps;"
```

The following string resets all previous settings, then requests a push of the changes:

```
syncOpt.syncParam = "reset;pushOnly;"
```

4.1.3 COM Interface TransportParam Parameters

With the `TransportParam` object, you can set the host, port, proxy server, and cookie for the Mobile Server. The format of the `TransportParam` string is used to set specific parameters using a string of a name or name/value pairs.

For example,

```
"name=value;name2=value2;name3=value3, ...;"
```

The names are not case sensitive, but the values are. Table 4–4 lists the `TransportParam` parameters.

Table 4–4 COM Interface TransportParam Parameters

Name	Value	Description
`reset`	N/A	Clears 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.

The following example directs the Mobile Sync engine to use the server at `test.oracle.com` through the proxy `proxy.oracle.com` at port 8080.

```
syncOpt.TransportParam =
      "server=test.oracle.com;proxy=proxy.oracle.com;proxyPort=8080;"
```

4.1.4 Selective Synchronization

The Mobile application can select the way specific tables are synchronized. That is, you can implement selective synchronization at the publication and table level by using the `mSync.SyncOption` interface to determine which publication and publication items are allowed to synchronize. The list of tables can be changed dynamically during runtime.

Note: You must perform an intial synchronization before you can issue a selective sync.

Use the `setSyncFlag` method to designate selective synchronization:

```
void setSyncFlag(BSTR pub_name, BSTR  tbl_name, short syncFlag)
```

Table 4–5 setSyncFlag Parameters

Parameter	Description
`pub_name`	The publication name is optional. If set to null, then the value of `syncFlag` is applied to all publications.
`tbl_name`	The table name (in the form `<client database>.<table name>`) is optional. If set to null, the parameter means all tables.
`syncFlag`	The synchronization flag is set to 1 to turn ON the `syncFlag` or to 0 to turn OFF the `syncFlag`.

The following sample code demonstrates how to turn off synchronization for all but one table. The table name in this sample is `ORD_DETAIL`. First the synchronization flag is set to 0 to turn off synchronization for all tables and then it is set to 1 to set on synchronization only for the specified table.

```
Dim syncOpt As MSYNC.SyncOption
syncOpt = CreateObject("MSync.SyncOption")
'Turn off sync flag for all tables.
syncOpt.setSyncFlag("", "", 0)
'Turn on sync flag only for the OrdersODB.ORD_DETAIL table.
syncOpt.setSyncFlag("", "OrdersODB.ORD_DETAIL", 1)
```

4.1.5 ISyncProgressListener Interface

You can track the status of the synchronization with the `progress` method of the `ISyncProgressListener` interface, which you create to return updates from the `ISync` interface. Basically, the `ISyncProgressListener` interface collects progress updates during synchronization. Table 4–6 lists the progress method and its parameters.

Table 4–6 ISyncProgressListener Abstract Method

Name	Description
`HRESULT progress(` `[in] int` *progressType*, `int` *param1*, `int` *param2*);	Called by the synchronization engine when new progress information is available. The `progressType` is set to one of the progress type constants defined in the `ISyncProgressListener` Constants table. Current is the current count completed, and total is the maximum. When current value equals the total value, then the stage is completed. The unit for total and current differs depending on the `progressType`.

The information that the `ISyncProgressListener` tracks is listed in Table 4–7.

Table 4–7 ISyncProgressListener Constants

Name	Progress Type
PT_INIT	Reports that the synchronization engine is in the initializing stage. The current and total counts are both set to 0.
PT_PREPARE_SEND	Reports that the synchronization engine is preparing local data to be sent to the server. This includes getting locally modified data. For streaming implementations, this is much shorter.
PT_SEND	Reports that the synchronization engine is sending data to the network. The total count denotes the number of bytes to be sent, and current is the byte count sent currently.

Table 4–7 (Cont.) ISyncProgressListener Constants

Name	Progress Type
PT_RECV	Reports that the engine is receiving data from the server. The total count denotes the number of bytes to be received, and current is the byte count received currently.
PT_PROCESS_RECV	Reports that the engine is applying the newly received data from the server to local data stores.
PT_COMPLETE	Reports that the engine has completed the synchronization process.

The following Visual Basic example demonstrates how to report events:

```
' Define the ISync object with events
Dim WithEvents sync As MSync.sync

' Create the callback.
' The name of the callback is the name of the ISync object (not the class), and
' underscore and then the function name - progress
Private Sub sync_progress(ByVal progressType As Long, ByVal param1 As Long, ByVal
param2 As Long)
    Desc = ""
    ' Decipher the progressType
    Select Case progressType
        Case PT_SEND
            Desc = "Sending data..."
        Case PT_RECV
            Desc = "Receiving..."
    End Select
End Sub
```

4.1.6 Initiate Synchronization with the ISync Interface

The `ISync` interface, whose methods are described in Table 4–8, allows the user to initiate the synchronization process.

Table 4–8 ISync Interface Abstract Methods

Name	Description
`HRESULT doSync()`	Start the synchronization process. This blocks access until the synchronization process is completed.
`void abort()`	Aborts the current synchronization. This can be called from a progress listener callback.
`HRESULT setOption( ISyncOption *syncObj)`	Sets the pointer to `SyncOption` to use for the next synchronization. If this function is not called before `doSync()`, the last saved option will be used. See Section 4.1.1, "COM Interface ISyncOption Interface" for information on `SyncOption`.

The following Visual Basic example demonstrates how to start a synchronization session using default settings.

```
Dim sync As Msync.sync
Set sync = CreateObject("MSync.Sync")
sync.DoSync
```

If `SyncOption` is not provided, the interface loads the last saved information to perform synchronization.

4.2 Synchronization APIs For C or C++ Applications

You can initiate and monitor synchronization from a C or C++ client application. The synchronization methods for the C/C++ interface are contained in `ocapi.h` and `ocapi.dll`, which are located in the <*ORACLE_HOME*>`\Mobile\bin` directory.

A C++ example is provided in the <*ORACLE_HOME*>`\Mobile\Sdk\Samples\sync\msync\src` directory. The source code is contained in `SimpleSync.cpp`. The executable—`SimpleSync.exe`—is in the <*ORACLE_HOME*>`\Mobile\Sdk\Samples\sync\msync\bin` directory.

The functions available for setting up and initiating the synchronization are as follows:

1. Section 4.2.1, "Initializing the Environment With ocSessionInit"
2. Section 4.2.2, "Managing the C/C++ Data Structures"
3. Section 4.2.3, "Retrieving Publication Information With ocGetPublication"
4. Section 4.2.4, "Managing User Settings With ocSaveUserInfo"
5. Section 4.2.5, "Manage What Tables Are Synchronized With ocSetTableSyncFlag"
6. Section 4.2.6, "Start the Synchronization With the ocDoSynchronize Method"
7. Section 4.2.7, "Clear the Synchronization Environment Using ocSessionTerm"

4.2.1 Initializing the Environment With ocSessionInit

The `ocSessionInit` function initializes the synchronization environment—which is contained in the `ocEnv` structure. For a full description of `ocEnv`, see Section 4.2.2.1, "ocEnv Data Structure".

Syntax

```
int ocSessionInit( ocEnv env );
```

Table 4–9 lists the `ocSessioninit` parameter and its description.

Table 4–9 ocSessionInit Parameters

Name	Description
`env`	An `ocEnv` class, which contains the synchronization environment.

This call initializes the `ocEnv` structure and restores any user settings that were saved in the last `ocSaveUserInfo` call (See Section 4.2.4, "Managing User Settings With ocSaveUserInfo"). An `ocEnv` structure is passed as the input parameter, where the `ocEnv` should be already created by the caller. If the caller wants to overwrite user preference information after the `ocSessionInit()` call, it can be done by calling `ocSaveUserInfo`.

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
rc = ocSessionInit(env);
```

4.2.2 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.

4.2.2.1 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 4–10 lists the field name, type, usage, and corresponding description of the `ocEnv` structure parameters.

Table 4–10 ocEnv Structure Field Parameters

Field	Type	Usage	Description
username	`char[32]`	Required. Set before calling `ocSessionInit`.	Name of the user to authenticate.
password	`char[32]`	Required. Set before calling `ocSessionInit`.	User password (clear text).
newPassword	`char[32]`	Optional. Can set after calling `ocSessionInit`.	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. Can set after calling `ocSessionInit`.	If set to 1, the password is saved locally and is loaded the next time `ocSessionInit` is called.
appRoot	`char[32]`	Optional. Can set after calling `ocSessionInit`.	Directory to where the application will be copied. If first character is null, then it uses the default directory.
priority	Short	Optional. Can set after calling `ocSessionInit`.	0= OFF (default) 1= ON; Only high priority table or rows are synchronized when turned on.
secure	Short	Optional. Can set after calling `ocSessionInit`.	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. Can set after calling `ocSessionInit`.	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.
trType	Enum	Required. Set before calling `ocSessionInit`.	If set to 0 (`OC_BUILDIN_HTTP`), then use HTTP built-in transport driver. If set to `OC_USER_METHOD`, then use user provided transport functions.
exError	ocError	Read-only information updated by OCAPI.	Extended error code - either OS or OKAPI error code.

Table 4–10 (Cont.) ocEnv Structure Field Parameters

Field	**Type**	**Usage**	**Description**
transportEnv	ocTransportEnv		Transport buffer. See Section 4.2.2.2, "ocTransportEnv Data Structure".
progressProc	fnProgress	Optional. Can set after calling `ocSessionInit`.	If not null, points to the callback for progress listening.
totalSendDataLen	Long		Reserved
totalRecieveDataLen	Long		Reserved
userContext	Void*	Optional. Can set after calling `ocSessionInit`.	Can be set to anything by the caller for context information (such as progress dialog handle, renderer object pointer, and so on.
ocContext	Void*		Reserved.
logged	Short		Reserved.
bufferSize	Long		Reserved (for Wireless/Nettech only).
pushOnly	Short	Optional. Can set after calling `ocSessionInit`.	If set to 1, then only push changes to the server.
syncApps	Short	Optional. Can set after calling `ocSessionInit`.	Set to 1 (by default), performs application deployment. If set to 0, then no applications will be received from the server.
syncNewPublications	Short	Optional. Can set after calling `ocSessionInit`.	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. Can set after calling `ocSessionInit`.	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. Can set after calling `ocSessionInit`.	If set to 1, log sync start time is recorded in the `conscli.odb` file.

Table 4–10 (Cont.) ocEnv Structure Field Parameters

Field	**Type**	**Usage**	**Description**
updateLog	Short	Optional. Can set after calling `ocSessionInit`.	Debug only. If set to 1, logs server-side insert and update row information to the publication odb.
options	Short	Optional. Can set after calling `ocSessionInit`.	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. ■ If set, OCAPI will try to abort the current sync session. ■ `OCAPI_OPT_FULLREFRESH` Forces OCAPI to purge all existing data and do a full refresh.
cancel	Short	Caller can set to 1 on next operation. `ocDoSyncrhonize` returns with -9032.	

The environment structure contains fields that the caller can update to change the way Mobile Sync module works. The following example demonstrates how to set the fields within the `ocEnv` structure.

```
typedef struct ocEnv_s {
 // User info
char username[MAX_USERNAME];    // Mobile Sync Client id
char password[MAX_USERNAME];    // Mobile Sync Client password for
                                // authentication during sync
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

                        // For Palm 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;
```

4.2.2.2 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])
```

4.2.3 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.

Syntax

```
ocError ocGetPublication(ocEnv* env, const char* application_name,
char* buf, int buf_len);
```

The parameters for the `ocGetPublication` function are listed in Table 4–11 below. The table lists the name of the `ocGetPublication` parameter and provides a description of it.

Table 4–11 ocGetPublication Parameters

Name	Description
`ocEnv* env`	Pointer to an ocEnv structure buffer to hold the return synchronization environment.
const char* application_ name(in)	This is the name of the application.
`char* buf(out)`	The buffer where the publication name will be stored.
int buf_len(in)	The buffer length. It 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.

This function gets the publication name from the Web-to-Go application name and stores it in the buffer.

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;
}
```

4.2.4 Managing User Settings With ocSaveUserInfo

Saves user settings to the `conscli.odb` database file.

Syntax

```
int ocSaveUserInfo( ocEnv *env );
```

The parameter for `ocSaveUserInfo` function is listed in a table below.

Table 4–12 lists the ocSaveUserInfo parameter and its description.

Table 4–12 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.

- Username
- Password
- SavePassword
- URL
- UseProxy
- ProxyServer
- ProxyPort
- Secure

For more information on how to use these fields, see Section 4.2.2, "Managing the C/C++ Data Structures".

4.2.5 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.

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 4–13 lists the name and description of parameters for the `ocSetTableSyncFlag` function.

Table 4–13 ocSetTableSyncFlag Parameters

Name	Description
`env`	Pointer to the synchronization environment.
`publication_name`	The name of the publication which is being synchronized. If the value for the `publication_name` is `NULL`, it means all publications in the database. This string is the same as the `client_name_template` parameter of the Consolidator Manager `CreatePublication` method. In most cases, you will use `NULL` for this parameter. For more information, see Section 3.5.4, "Creating Publications" in Chapter 3, "Synchronization".
`table_name`	This is the name of the snapshot. It is the same as the name of the `store`, the third parameter of `CreatePublicationItem()`. For more information, see Section 3.5.5, "Creating Publication Items" in Chapter 3, "Synchronization".
`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 synchronization of specific tables.

   Example:

   ```
   ocSetTableSyncFlag(&env, <publication_name>,<table_name>,1)
   ```

4.2.6 Start the Synchronization With the ocDoSynchronize Method

Starts the synchronization process.

Syntax

```
int ocDoSynchronize( ocEnv *env );
```

The parameter for the `ocDoSynchronize` method is listed in a table below.

Table 4–14 lists the name and description of the `ocDoSynchronize` parameter.

Table 4–14 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.

A return value of 0 indicates that the function has been executed successfully. Otherwise, the value is an error code.

4.2.7 Clear the Synchronization Environment Using ocSessionTerm

Clears and performs a cleanup of the synchronization environment.

Syntax

```
int ocSessionTerm( ocEnv *env );
```

The parameter for `ocSessionTerm` function is listed in a table below.

Table 4–15 lists the ocSessionTerm parameter and its description.

Table 4–15 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.

4.3 Synchronization API for Java Applications

The following sections describe how you can use Java on a Pocket PC device to build your own client synchronization initiation:

- Section 4.3.1, "Overview"
- Section 4.3.2, "Sync Class"
- Section 4.3.3, "SyncException Class"
- Section 4.3.4, "SyncOption Class"
- Section 4.3.5, "Java Interface SyncParam Settings"
- Section 4.3.6, "Java Interface TransportParam Parameters"
- Section 4.3.7, "SyncProgress Listener Service"

4.3.1 Overview

Using the Java interface for Mobile Sync client-side synchronization tasks, programs written in Java can use the functionality provided by the OCAPI library. The Java interface resides in the `oracle.lite.msync` package.

The Java interface provides for the following functions:

- Setting client side user profiles containing data such as user name, password, and server
- Starting the synchronization process

- Tracking the progress of the synchronization process

The Java interface consists of two files, `mSync.jar` and `msync_java.dll`. To use the Java interface, the `mSync.jar` file must be included in the classpath. The `mSync.jar` file is located in the following directory.

`<ORACLE_HOME>\Mobile\classes`

The `msync_java.dll` file is located in the following directory.

`<ORACLE_HOME>\Mobile\bin`

There are four parts to the Java interface. They are:

- `Sync` Class
- `SyncException` Class
- `SyncOption` Class
- `SyncProgressListener` Interface

The following sections describe the Java interface.

4.3.2 Sync Class

This class initiates synchronization by using the provided synchronization options. The parameters for the constructor are listed in Table 4–16.

Constructors

`Sync(SyncOption option)`

Table 4–16 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 4–17.

Table 4–17 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());
}
```

4.3.3 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 4–18:

Constructors

`SyncException()`

`SyncException(int errorCode, string errorMessage)`

Table 4–18 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 4–19.

Table 4–19 SyncExceptionClass Public Methods

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

4.3.4 SyncOption Class

The `SyncOption` class is used to define the parameters for the synchronization process. It can either be constructed manually, or can save or load data from the user profile.

Constructors

```
SyncOption()

SyncOption
   ( String user,
     String password,
     String syncParam,
     String transportDriver,
     String transportParam)
```

The parameters for the `SyncOption` constructor are listed in Table 4–20:

Table 4–20 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 4.3.5, "Java Interface SyncParam Settings" for more information.
`transportDriver`	A string containing the name of the transport driver. Currently, only "HTTP" is supported.
`transportParam`	A string containing all the parameters needed for the specified driver to operate. See Section 4.3.6, "Java Interface TransportParam Parameters" for more information.
`priority`	A boolean value which limits synchronization to server tables flagged as high priority, otherwise all tables are synchronized.
`pushOnly`	A boolean value which makes synchronization push only.

Public Methods

These methods load and save the user profile. The parameters of the public methods are listed in Table 4–21:

Table 4–21 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()`	
`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 setTransportDriver(String driverName) String getTransportDriver()`	This is used to set and get the driver name. Release 5.0.2 supports the "HTTP" driver.
`void setTransportParam(String transportParam) String getTransportParam()`	Set and get the transport parameters.

Example

The following code example demonstrates how to start a synchronization session using the default settings:

```
SyncOption opt = new SyncOption

("sam","lion","pushonly","HTTP","server=server1;proxy=www-proxy.us.oracle.com;prox
yPort=80");
```

```
opt.save();
```

4.3.5 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 4–22.

Table 4–22 Java Interface SyncParamSettings

Name	Value/Options	Description
`"reset"`	N/A	Clear all entries in the environment before applying any remaining settings.
`"security"`	`SSL` or `AES`	Use the appropriate selection to choose either SSL or AES stream encryption.
`"push only"`	N/A	Use this setting to upload changes from the client to the server only, do not download. This is useful when data transfer is one way, client to server.
`"noapps"`	N/A	Do not download any new or updated applications. This is useful when synchronizing over slow connection or on a slow network.
`"syncDirection"`	`"sendonly"` `"receiveonly"`	"SendOnly" is the same as "pushonly". "ReceiveOnly" allows no changes to be posted to the server.
`"noNewPubs"`	N/A	This setting prevents any new publications created since the last synchronization from being sent, and only synchronizes data from the current publications.
`"tableFlag"`	`"enable"`	The "enable" setting allows [Publication.Item] to be synchronized, "disable" prevents synchronization.
`[Publication.Item]`	`"disable"`	
`"fullrefresh"`	N/A	Forces a complete refresh.
`"clientDBMode"`	`"EMBEDDED"` or `"CLIENT"`	If set to "EMBEDDED", access to the database is by conventional ODBC, if set to "CLIENT" access is by multi-client ODBC.

Example 1

The first example enables SSL security and disables application deployment for the current synchronization session:

```
"security=SSL; noapps;"
```

Example 2

The second example resets all previous settings, activates upload for the "Dept" table only:

```
"reset;pushOnly;tableFlag[TestApp.Emp]=disable;tableFlag[TestApp.Dept]=enable;"
```

4.3.6 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 4–23.

Table 4–23 TransportParam Parameters

Name	Value	Description
`"reset"`	N/A	Clear all entries in the environment before applying the rest of the settings.
`"server"`	server hostname	The hostname or IP address of the Mobile Server.
`"proxy"`	proxy server hostname	The hostname or IP address of the proxy server.
`"proxyPort"`	port number	The port number of the proxy server.
`"cookie"`	cookie string	The cookie to be used for transport.

Example

The example directs the Mobile Sync engine to use the server at "test.oracle.com" through the proxy "proxy.oracle.com" at port 8080:

```
"server=test.oracle.com;proxy=proxy.oracle.com;proxyPort=8080;"
```

4.3.7 SyncProgress Listener Service

The SyncProgressListener is an interface that allows progress updates to be trapped during synchronization.

This class initiates synchronization by using the provided synchronization options. The parameters for the method are listed in Table 4–24:

Method

```
void progress

   (int progressType,

    int completed);
```

Table 4–24 SyncProgressListener Abstract Method

Parameter	Description
`progressType`	This is set to one of the constants listed in Table 4–25.
`completed`	This is the percentage of completion for specific `progressType`.

The names of the constants which report the synchronization progress are listed in Table 4–25.

Table 4–25 SyncProgressListener Interface Constants

Constant Name	Progress Type
`PT_INT`	States that the synchronization engine is in the initializing stage. The current and total counts are set to 0.
`PT_PREPARE_SEND`	States that the synchronization engine is preparing local data to be sent to the server. This includes getting locally modified data. For streaming implementations this takes a shorter amount of time.
`PT_SEND`	States that the synchronization engine is sending data to the network. The total count equals the number of bytes to be sent, and the current count equals the byte count being sent currently.
`PT_RECV`	States that the synchronization engine is receiving data from the server. The total count equals the number of bytes to be received, and the current count equals the byte count being received currently.
`PT_PROCESS_RECV`	States that the synchronization engine is applying the newly received data from the server to the local data stores.
`PT_COMPLETE`	States that the synchronization engine has completed the synchronization process.

Example

This simple class implements the `SyncProgressListener`.

```
class myProgressTracker implements SyncProgress Listener;

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

4.4 msync/OCAPIs/mSyncCom

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

5 Using Mobile Database Workbench 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 sections describe how to first create a project and then create the other objects contained within a publication.

- Section 5.1, "Create a Project"
- Section 5.2, "Create a Publication Item"
- Section 5.3, "Create a Sequence"
- Section 5.4, "Test and Load a Script Into The Project"
- Section 5.5, "Load a Resource Into the Project"
- Section 5.6, "Create a Publication"
- Section 5.7, "Import Existing Publications and Objects from Repository"
- Section 5.8, "Create a Virtual Primary Key"
- Section 5.9, "Test a Publication by Performing a Synchronization"
- Section 5.10, "Deploy the Publications in the Project to the Repository"

5.1 Create a Project

Create a new project with the Project Wizard. The project is the vehicle that contains your iterative approach to defining publications, publication items, sequences, scripts and resources. The project can be saved and restored from your file system, so that you can continue to modify any of the contained objects within it.

You cannot perform any action on developing your publications without first creating the project.

You must have access to the back-end database with the Oracle Mobile Repository and already defined the tables and schema that you are going to be using in your publication items before entering the project wizard.

Perform the following to create the project:

1. Click **File->New->Project** to start the Project Wizard.
2. An Introductory screen appears. If you do not want this introductory screen to display each time you start a new project, check the "Skip This Page Next Time" box.
3. Define the project name. Enter the project name and location for your new project, as follows:
   - Project name: This name can be any valid Java identifier. The name cannot contain any spaces. For example, your project name may be something like `MY_NEW_PROJECT`.
   - Project location: Enter a valid location in the directory on your machine that has write permission to store the project. On a Windows machine, you could enter the location as `c:\myprojects`. To browse for a directory, click **Browse**.

   Click **Next** to move to the next step in the wizard.
4. Provide the Mobile Repository access information. Because you are interacting with the repository to create and manipulate synchronization objects, including the SQL scripts for the publication items, you need access to the Mobile Repository. Enter the following access information:
   - User name and password: Specify the Mobile Server repository user name and password (with administrator privilege). This is the same user name and password with which the repository was originally created. For example, the default Mobile Server repository user name and password is `mobileadmin/manager`.
   - JDBC driver type: Select the JDBC driver that is used to connect to the Oracle database that hosts the Mobile Server repository. At this time, the only driver that you can choose is the Oracle Thin driver.
   - Host name or IP address, port number, and SID: Enter the location, port, and SID of the database that contains the Mobile Server Repository. For the location, you can either enter the host name or the IP address. The port and SID are configured within the definition of the Oracle database and are used to access the database. For example, the host, port and SID could be `my-pc1`, 1521, and `orcl`.

   Click **Next** to move to the next step in the wizard. Once you click Next, the wizard verifies that the database connection information is correct. If incorrect, the wizard prompts you to re-enter the information. You can only advance if you enter the correct information where the Mobile Server Repository is located.
5. Specify schema username and password. Enter the user and password of the schema owner for the schema that you are using for the Mobile application. The Mobile application schema contains all database tables, views, synonyms used to build the snapshots for the application.

   Note: All schema objects for an application exist in the same back-end repository, which is why the Oracle database host, port and SID are only read-only on this screen.

Click **Next** to move on to the last screen in the Project Wizard. As you click **Next**, MDW verifies that the username and password that you entered are valid for connecting to the application schema in the back-end database. If these are not valid, you cannot advance until you supply a valid username and password.

6. A summary page appears. Once the creation of the project is completed, this page displays all of the information about your new project.
   - Click **Back** to modify any of the information supplied.
   - Click **Finish** to complete the project creation.
   - Click **Cancel** to abort creation of this project.

5.2 Create a Publication Item

The Publication Item Wizard steps you through the process of creating a publication item in the project. A publication item encapsulates a snapshot definition. It can be based on a table, view or synonym in the Master Application schema in the back-end database. If you use a synonym for a remote object to build a publication item, then you are required to provide the JDBC connection information to the remote database where the remote object resides.

After you create the publication items in the project, then you can associate multiple publication items with a publication, which is then associated with an application. See Section 5.6.1, "Associate Publication Item With 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 and refresh type, as follows:
   - Publication item name: This name can be any valid Java identifier. The name cannot contain any spaces. Publication item names are limited to twenty-six characters and must be unique across all publications. For example, your publication item name may be something like `MY_PUBLICATION_ITEM`.
   - Refresh type: The refresh mode of the publication item is specified during creation to be either fast or complete refresh. See the Section 3.8, "Understanding Your Refresh Options" for more information.

     From the drop-down list choose one of the following refresh types:
     - Complete: All data is refreshed with current data. Everytime you synchronize, all data in the snapshot is retrieved. This can be performance intensive.
     - Fast: This is the recommended mode. Only incremental changes are synchronized. Thus, you are not downloading the complete data snapshot each time a synchronization is requested. The advantages of fast refresh are reduced overhead and increased speed when replicating data stores with large amounts of data where there are limited changes between synchronization sessions.
     - Queue-based: You can create your own queue. Mobile Server will upload and download changes from the user. You perform the activity of the MGP

and apply/compose the modifications to the back-end database. See the Section 3.16, "Customizing Replication With Your Own Queues" for more information.

Once you create the publication item with a particular refresh type, the only way to modify the publication item to have a different refresh type is to delete is and recreate it with the desired refresh type.

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. There are two tabs to enable you to structure your publication item, as follows:

   - Column Selection: Choose the columns from within the object that you will use to retrieve information for the application. To choose the appropriate columns, select the column name and click the left or right arrow buttons to move between the Available and Selected windows. To move all columns, use the double arrows.

     Note: The primary key defaults to being in the Selected window, as it is required if you are using Fast Refresh and Updateable option. Since most publication items use these options, MDW places the primary key as Selected. You can move it back to the Available window.

   - Structure: If you are not sure what columns you want, you can see the entire table structure by clicking this tab.

   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.

5. Modify the SQL statement for the publication item. From the columns that you selected in the previous screen, this simple SQL statement is available as a

template for you to modify. You can add any qualifiers and complexity to this base statement. To view the structure of the schema object, select the Structure tab.

- Perform Iterative Modifications

  See Section 5.2.1, "Create SQL Statement for Publication Item" for directions on how to edit and execute the SQL statement for this publication item.

- Apply/Compose Callbacks

  When creating publication items, the user can specify a customizable package to be called during the Apply and Compose phase of the MGP background process. Client data is accumulated in the in queue prior to being processed by the MGP. Once processed by the MGP, data is accumulated in the out queue before being pulled to the client by Mobile Sync. See Section 3.6.2, "Customize What Occurs Before and After Compose/Apply Phases for a Single Publication Item" for more information on how to create the callbacks.

  Provide the schema and package names for any apply/compose callbacks.

- Dependency Hint

  Click **Add** to add a dependency hint. All existing dependency hints are shown in the window. See Section 5.2.2, "Create a Dependency Hint" for more information.

6. If you specified a view, then you may need to define parent table and primary key hints. See Section 5.2.3, "Specify Parent Table and Primary Key Hints" for directions on how to define these hints.

7. Summary page provides an overview of the publication item that you just created. To view any dependency hints, click **Advanced**. If you have used a view as the base for your publication item and you created Parent Table or Primary Key hints, click Hint to view these hints. The Parent Table and Primary Key hints are only valid on views.

   To modify any part of the publication item, click **Back** to return to the previous screens.

   Click **Finish** if you are satisfied with the publication item. Click **Cancel** to eliminate all work in creating this publication item.

5.2.1 Create SQL Statement for Publication Item

You can compose your SQL statement through iterative steps to ensure that you are creating a valid statement. You have the following options:

- Modify the query by clicking **Edit**.
- Execute the statement against the schema in the back-end database by clicking **Run**. You will be notified directly if the statement fails or view the data from the schema object is retrieved for you to view.

  When you click **Run**, then the query of the publication item is validated by executing the query against the back-end database. A dialog is displayed with the returned snapshot that this publication item would generate. If there is more than one page, then click **Previous** and **Next** to move between the pages. Click **Close** to return to the publication item screen.

  If the publication item SQL statement requireds an input value, then a dialog appears for you to input the value for the SQL query. You can modify this value or the SQL query until you receive the results that you desire.

- To return the query to the original statement, click **Reset**.
- When finished, click **Next**.

5.2.2 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 3.14.2, "Creating or Removing a Dependency Hint".

5.2.3 Specify Parent Table and Primary Key Hints

When you use a view, you may need to specify a parent table and primary key hints. A view can be composed of one or more tables joined together. If you have specified fast refresh, then you must specify which table is the parent table and which column is the primary key.

- To create a parent table hint, select the base table from the Base Table(s) drop-down list and check the Parent Table Hint checkbox.

> **Note:** If you do not check the Parent Table Hint checkbox, then the hint is not created when you click **Next**.

- To create a primary key hint, click **Add**. Identify the primary key hint for this view, as follows:

  1. From the View Columns drop-down list, select the view column that you want to be the primary key. This column name may be an alias of the actual `table.column` name.
  2. From the Base Tables drop-down list, select the base table where the actual column exists that is to be the primary key.
  3. From the Primary Key Columns drop-down list, all primary key columns from the base table are shown, select the appropriate column for the primary key. If you have a composite primary key, iteratively add each column within the composite primary key.

  > **Note:** If you do not provide accurate details in regards to the view, base table, and primary key to which the view column maps, the publication item may save, but the execution of the publication item will fail. For example, if you choose a view column which does not map to the primary key column, MDW will allow the action, but any execution of the publication item will result in failure.

  Click **OK** to accept this primary key hint.

5.3 Create a Sequence

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

Create a sequence by clicking **File->New->Sequence**.

However, when you have Oracle Database Lite, you must consider how you allow the generation of sequence numbers. If you want the client only to use the sequence, you have a native sequence, which does not involve the server-side. However, if you want the server and client to share a sequence, you have to specify a server-side sequence and allow for both the client and server to be using numbers within the same range.

For example, if you have a single back-end database and a Mobile client, where both are allowed to add records to the application tables, then you have to check the checkbox to generate the server-side sequence and allow at least two as the increment value, since the client will use one and the server will use the other.

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

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

To create a sequence, provide the following information:

- Name: This name must be a valid Java identifier.
- Starts With: Enter the number with which you want this sequence to start.
- Increment: Specify the increment from the starting value for the next value in the sequence.

Note: If you have checked the **Generate server-side sequence** checkbox and set the increment value to 1, then this value is ignored and is set to 2. When you specify the server-side sequence, then both the client and the server use every other number in the sequence. Thus, you cannot increment by 1 on the client.

- Window Size: The range of numbers given to the client when the threshold is reached.
- Threshold: Defining the amount of sequence numbers necessary to be assigned in order for operations to continue smoothly.
- Description: A description of the sequence.
- Generate server-side sequence: If you want the client and the server-sides to share a sequence, where one side has all even numbers and the other has the odd numbers, check this box. If unchecked, then the sequence is created solely for the client.

Once you create a sequence in the project, you can associate it with a publication. See Section 5.6.2, "Associate a Sequence With the Publication" for details.

See the Section 3.3.1.7, "Creating Client-Side Sequences for the Downloaded Snapshot" for more information on sequences.

5.4 Test and Load a Script Into The Project

You can add a script to this project. Create the script on your file system and then upload it to MDW. Before you add the script to the project, you can use MDW to test the script. See the following sections for more information:

- Section 5.4.1, "Test SQL Scripts"
- Section 5.4.2, "Load the Script Into the Project"

5.4.1 Test SQL Scripts

You can test a SQL script that resides on your file system by selecting **Tool->SQL Window**. Through the SQL Wizard, perform the following:

1. Connect to the correct database—whether it is an Oracle database or a device Oracle Lite database.
2. Load the SQL script from your file system.
3. Execute the script. You can execute the current script or re-execute scripts that are on the history page. You can choose to have the results displayed on the screen or spooled to a file.

The following sections describe how to accomplish these tasks:

- Section 5.4.1.1, "Connect to the Database"
- Section 5.4.1.2, "Load and Execute SQL Scripts"

5.4.1.1 Connect to the Database

1. Select type of database—Select the radio button next to the type of database to which you are connecting—an Oracle database or the client Oracle Lite database. If you selected the client Oracle Lite database, you must also specify the Mobile client platform and protocol with the drop-down lists. Only currently installed platforms and protocols are displayed in these drop-down lists.
2. Specify database authentication and destination connection information—Part of the information necessary for completing the database connection is the authentication user name and password and the database destination information, which can include either the DSN of the Mobile client Oracle Lite database or the host, port, and SID for the Oracle database.

3. Review database connection values—The final screen displays a summary of all of the configured values for the database connection. Verify that the information is correct and then click **Finish**. You can return to any page to modify the information by clicking **Back**.

5.4.1.2 Load and Execute SQL Scripts

You can test any SQL scripts against the database defined in the previous portion of this wizard.

- Click **Load Script** to browse your file system for the script that you want to test. Define if you want this script to be spooled or auto-committed.
- Click **Execute** to run the script on the destination database. The results show up in the bottom results screen.

  If you want, you can spool the results to a file by checking the spool checkbox before you click **Execute**. When you check Spool, a dialog appears for you to define the location and name of the file to receive the output from the script. Check the Overwrite checkbox if you want this file overwritten each time that the script is executed.

  Check the Autocommit checkbox if you want the SQL committed automatically after the script completes.

The Results and History tabs show the current results and all past results respectively. Clear the Results screen by either clicking **Clear Results** or by checking the Auto Clear checkbox. Clear the historical information by clicking the **Clear History** button on the History page.

Note: Any SQL on the History page can be executed by selecting the corresponding row and clicking **Execute**.

5.4.2 Load the Script Into the Project

Define the script on your machine. Once defined, bring the script into the project by clicking **File->New->Script**. Provide a user-defined name to identify the script and browse for the script in your file system. Click **OK** to accept the definition and **Cancel** to return to the previous screen.

Once you include a script in the project, you can associate it with a publication. See Section 5.6.3, "Associate a Script With the Publication" for more information.

5.5 Load a Resource Into the Project

You can specify that Java class files and binaries archived in a JAR file is downloaded to the client on the first synchronization by specifying an existing resource. In addition, if this resource is modified, it will be sent down on the next synchronization.

To specify an existing resource, click **File->New->Resource**.

Provide the JAR file on your machine. Specify a user-defined name to identify the resource and browse for the JAR file in your file system. Click **OK** to accept the definition and **Cancel** to return to the previous screen.

Once you include a resource in the project, you can associate it with a publication. See Section 5.6.4, "Associate a Resource With the Publication" for more information.

5.6 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. Provide 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.

At this time, you can associate any existing objects with the publication by clicking on one of the following tabs: Publication Item, Sequence, Script, Resource. See the following sections for details on the information required for each of these screens. In addition, if you click **OK**, then you can associate the objects by selecting the publication name and then selecting the appropriate tab.

- Section 5.6.1, "Associate Publication Item With Publication"
- Section 5.6.2, "Associate a Sequence With the Publication"
- Section 5.6.3, "Associate a Script With the Publication"
- Section 5.6.4, "Associate a Resource With the Publication"

5.6.1 Associate Publication Item With Publication

Selecting the Publication Item tab from within the publication enables you to associate any existing publication item to this publication.

Manage Publication Items In This Publication

- To add an existing publication item to this publication, Click **Add**.
- To remove a publication item from this publication, select the desire publication item from the list and click **Remove**.
- To edit the details of the association for the publication item, select the desired publication item and click **Edit**.

To accept the current changes, click **OK**.

5.6.1.1 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.

- Read-only snapshots are used for querying purposes. Changes made to the master table are replicated to the snapshot by the Mobile client.
- Updatable snapshots provide updatable copies of a master table. You can define updatable snapshots to contain a full copy of a master table or a subset of rows in the master table that satisfy a value-based selection criteria. You can make changes to the snapshot which the Mobile Sync propagates back to the master table.

  A snapshot can only be updated when all the base tables that the snapshot is based on have a primary key. If the base tables do not have a primary key, a snapshot cannot be updated and becomes read-only.

Conflict Resolution

When adding a publication item to a publication, the user can specify winning rules to resolve synchronization conflicts in favor of either the client or the server. A Mobile Server synchronization conflict is detected under any of the following situations:

- The same row was updated on the client and on the server.
- Both the client and server created rows with equal primary keys.
- The client deleted a row and the server updated the same row.
- The client updated a row and the server deleted the same row. This is considered a synchronization error for compatibility with Oracle database advanced replication.
- For systems with delayed data processing, where a client's data is not directly applied to the base table (for instance in a three tier architecture) a situation could occur when first a client inserts a row and then updates the same row, while the row has not yet been inserted into the base table. In that case, if the `DEF_APPLY` parameter in `C$ALL_CONFIG` is set to `TRUE`, an `INSERT` operation is performed, instead of the `UPDATE`. It is up to the application developer to resolve the resulting primary key conflict. If, however, `DEF_APPLY` is not set, a "`NO DATA FOUND`" exception is thrown (see below for the synchronization error handling).
- All the other errors including nullity violations and foreign key constraint violations are synchronization errors.
- If synchronization errors are not automatically resolved, the corresponding transactions are rolled back and the transaction operations are moved into Mobile Server error queue in C$EQ, while the data is stored in CEQ$. Mobile Server database administrators can change these transaction operations and re-execute or purge transactions from the error queue.

Choose the type of conflict resolution you want for this publication item, as follows:

- Client wins—When the client wins, the Mobile Server automatically applies client changes to the server. And if you have a record that is set for `INSERT`, yet a record already exists, the Mobile Server automatically modifies it to be an `UPDATE`.
- Server wins—If the server wins, the client updates are not applied to the application tables. Instead, the Mobile Server automatically composes changes for the client. The client updates are placed into the error queue, just in case you still want these changes to be applied to the server—even though the winning rules state that the server wins.
- Custom—You have created your own callbacks to resolve the conflict resolution.

All synchronization errors are placed into the error queue. For each publication item created, a separate and corresponding error queue is created. The purpose of this queue is to store transactions that fail due to unresolved conflicts. The administrator can attempt to resolve the conflicts, either by modifying the error queue data or that of the server, and then attempt to re-apply the transaction.

See Section 3.11, "Resolving Conflict Resolution with Winning Rules" for more information.

DML Callback

A user can use Java to specify a customized PL/SQL procedure which is stored in the Mobile Server repository to be called in place of all DML operations for this publication item. There can be only one mobile DML procedure for each publication item. See Section 3.3.1.12, "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.

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 16.1.2, "Shared Maps" for more information.

Priority Condition

Provide a string that is to be added to the publication item query statement to limit what is returned based on priority. For example,

For example, if you have a snapshot with the following statement:

`select * from projects where prio_level in (1,2,3,4)`

The projects table has a column named `prio_level`, where the values can be 1 to 4. If you wanted to limit what data was returned, you could limit the statement specific to this publication item in this publication by adding a string that would be added to the SQL statement preceded by an AND.

For example, to restrict the snapshot from the projects table, you could define the priority condition string to be `prio_level = 1`. This would generate the following statement:

```
SELECT * FROM projects where prio_level in (1,2,3,4) AND prio_level = 1;
```

In this case, only projects with level =1 are replicated to the client. You can make this statement to be anything you wish to restrict what is returned to the client.

See the Section 16.1.3, "Priority-Based Replication" for more information.

MyCompose Class

Provide a string with the full path and classname of the location and name of the MyCompose Class. See Section 3.5, "Customize the Apply and Compose Phases Using the Consolidator Manager APIs" for more information on this class.

Weight

You can rate the order in which each publication item in this publication is executed by specifying the weight. This should be a number. Each publication item must have a unique number in ascending order. The first publication item executed is the one with the weight of one.

5.6.2 Associate a Sequence With the Publication

You can only associate an existing sequence with the publication on this screen. To add an existing sequence, click Add.

> **Note:** You can create a sequence through the **File->New->Sequence** screen.

Click on the drop-down list and select one of the existing sequences to add to the publication. Click **OK** to add the sequence; click **Cancel** to go back to the previous screen.

5.6.3 Associate a Script 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.

5.6.4 Associate a Resource With the Publication

You can only associate an existing resource with the publication on this screen. To add an existing resource, click Add.

> **Note:** You can import a resource through the **File->New->Resource** screen.

Click on the drop-down list and select one of the existing resources to add to the publication. Click **OK** to add the resource; click **Cancel** to go back to the previous screen.

5.7 Import Existing Publications and Objects from Repository

You can import existing publications, publication items, sequences, scripts or resources that already exist within the repository by choosing the Project->Add From Repository option, as described in the following sections:

- Section 5.7.1, "Import Existing Publication from Repository"
- Section 5.7.2, "Import Existing Publication Item From the Repository"
- Section 5.7.3, "Import Existing Sequence From the Repository"
- Section 5.7.4, "Import Existing Resource From the Repository"
- Section 5.7.5, "Import an Existing Script From the Repository"

5.7.1 Import Existing Publication from Repository

You can add an existing publication that already exists in the repository to this project by selecting **Project->Add From Repository->Publication**. All associated objects—publication items, sequences, scripts, resources—are also pulled into the project with the publication.

To view all publications in the repository, click **Search**. All publications are shown in the left-hand screen. To limit the displayed publications to only those with a certain string as part of the name, provide this string in the Filter and then click **Search**. Only those publications that match the filter are shown.

Note: In the Search Filter, you can use the same pattern matching characters in a valid `SQL WHERE` clause. The filter is case-sensitive; use upper-case characters.

Select the desired publications and either double-click or select the right arrow to move them to the right window. Once all desired publications are in the right window, click **OK** to move these publications into the project.

5.7.2 Import Existing Publication Item From the Repository

You can add an existing publication item that already exists in the repository to this project by selecting **Project->Add From Repository->Publication Item**.

To view all publication items in the repository, click **Search**. All publication items are shown in the left-hand screen. To limit the displayed publication items to only those with a certain string as part of the name, provide this string in the Filter and then click **Search**. Only those publication items that match the filter are shown.

Note: To search only for objects that match a condition, designate the condition in the Filter box and click **Search**. You can use the same pattern matching characters in a valid SQL WHERE clause. The filter is case-sensitive; use upper-case characters.

Select the desired publication items and either double-click or select the right arrow to move them to the right window. Once all desired publication items are in the right window, click **OK** to move these publication items into the project.

Once added into the project, you still must associate them with the publication if you want to test the synchronization of the publication item. See Section 5.6.1, "Associate Publication Item With Publication" for more information.

5.7.3 Import Existing Sequence From the Repository

You can add an existing sequence that already exists in the repository to this project by selecting **Project->Add From Repository->Sequence**.

To view all sequences in the repository, click **Search**. All sequences are shown in the left-hand screen. To limit the displayed sequences to only those with a certain string as part of the name, provide this string in the Filter and then click **Search**. Only those sequences that match the filter are shown.

Note: To search only for objects that match a condition, designate the condition in the Filter box and click **Search**. You can use the same pattern matching characters in a valid `SQL WHERE` clause. The filter is case-sensitive; use upper-case characters.

Select the desired sequences and either double-click or select the right arrow to move them to the right window. Once all desired sequences are in the right window, click **OK** to move these sequences into the project.

Once added into the project, you still must associate them with a publication if you want to test it with a synchronization. See Section 5.6.2, "Associate a Sequence With the Publication" for more information.

5.7.4 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->Resource**.

To view all resources in the repository, click **Search**. All resources are shown in the left-hand screen. To limit the displayed resources to only those with a certain string as part of the name, provide this string in the Filter and then click **Search**. Only those resources that match the filter are shown.

Note: To search only for objects that match a condition, designate the condition in the Filter box and click **Search**. You can use the same pattern matching characters in a valid SQL WHERE clause. The filter is case-sensitive; use upper-case characters.

Select the desired resources and either double-click or select the right arrow to move them to the right window. Once all desired resources are in the right window, click **OK** to move these resources into the project.

Once added into the project, you still must associate them with a publication if you want to test it with a synchronization. See Section 5.6.4, "Associate a Resource With the Publication" for more information.

5.7.5 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.

Once added into the project, you still must associate them with a publication if you want to test it with a synchronization. See Section 5.6.3, "Associate a Script With the Publication" for more information.

5.8 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:

1. Using the drop-down lists, choose the following:
   - Schema name
   - Object type: table, view or synonym type
   - Any string that exists within the object name, if desired
2. Click **Search**, which brings up a list of available objects.
3. From the object list, choose the appropriate table, view, or synonym. Once chosen, the available columns are listed.
4. Select the column(s) that you wish to be the primary key and click **OK**.

If you have a composite primary key, iteratively add each column within the composite primary key.

5.9 Test a Publication by Performing a Synchronization

You can create a test to perform a synchronization of the designated publication. Click Project->Test Publication. When you create the test, MDW automatically creates the subscription for the user.

1. Click **Create** to design the test and provide the following information:
   - Name: If the test is remote, then the user name is populated with the registered owner of the remote target device. If the test is local, then the user name should be a valid Mobile user in the repository.
   - Publication: From the drop-down list, select one of the available publications in this project for this test.
   - Client type: Designate if the client is local or remote. Default is local. If Active Sync is not installed, the remote option is not available.
   - Specify a user that is defined in Mobile Manager.

   Click **OK** to save the test; click **Cancel** to revert back to the previous screen.

Note: To remove any tests, select the test and click **Remove**.

---

2. Once created, click Synchronize to perform a synchronization for the designated publication. On the pop-up dialog, provide the password for the given username and the URL of the Mobile Server. The URL for the Mobile Server should be the hostname/webtogo.

   Click **Option** to specify priority of the publication items, as follows:

   - High Priority: Limits synchronization to server tables flagged as high priority, otherwise all tables are synchronized.
   - Push Only: Upload changes from the client to the server only, do not download. This is useful when data transfer is one way, client to server.
   - Complete Refresh: All data is refreshed from the server to the client.
   - Debug: Turn on debugging when synchronizing.
   - Selective Synchronization: Determine which publication and publication items are allowed to synchronize. When you click this option, move the publication items that you want to synchronize from the left window to the right window using the arrow buttons.

   Click **OK** to save the synchronization options or **Cancel** to return to the previous screen.

5.10 Deploy the Publications in the Project to the Repository

You can deploy one or more of the publications in the current project from the development/test Mobile Server repository to a target production Mobile Server repository by clicking File->Deploy. You should adequately test all publications before deploying to the production Mobile Server repository.

All available publications are displayed in the project publications section. To limit the displayed publications to only those with a certain string as part of the name, provide this string in the Filter and then click **Search**. Only those publications that match the filter are shown.

Note: In the Search Filter, you can use the same pattern matching characters in a valid `SQL WHERE` clause. The filter is case-sensitive; use upper-case characters.

Select the desired publications and click **OK** to deploy these publications into the repository. A dialog appears where you specify the remote database connection information, as follows:

- User name and password for database connection authentication.
- JDBC Driver type: Based on the type of the JDBC driver, different information is required. At this time, you can only use the JDBC Thin driver. Provide the host name, port, and SID for the remote database.

Click **OK** to accept the input values for the remote database; click **Cancel** to return to the previous screen.

6 Using the Packaging Wizard

The following sections enable you to package and publish your Mobile application definitions using the Packaging Wizard.

- Section 6.1, "Using the Packaging Wizard"
- Section 6.2, "Packaging Wizard Synchronization Support"

6.1 Using the Packaging Wizard

After you have completed the code implementation for your application, you need to define the SQL commands that retrieve the data for the user snapshot—also known as a publication. MDW (as described in Chapter 5, "Using Mobile Database Workbench to Create Publications") is a graphical tool that enables you to define the publications for your application. Then, use the Packagine Wizard to package the application with the subscription and publish the final application product to the Mobile Server.

- 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 Development Workbench (MDW) tool (as described in Chapter 5, "Using Mobile Database Workbench to Create Publications" for performing an iterative approach to defining your publications.
- Edit an existing Mobile application definition—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—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 GUI tool:

- Section 6.1.1, "Starting the Packaging Wizard"
- Section 6.1.2, "Specifying New Application Definition Details"
- Section 6.1.3, "Listing Application Files"
- Section 6.1.4, "Adding Servlets (For Web Applications Only)"
- Section 6.1.5, "Entering Database Information"
- Section 6.1.6, "Defining Application Roles"

- Section 6.1.7, "Defining Snapshots for Replication"
- Section 6.1.8, "Defining Sequences for Replication"
- Section 6.1.9, "Defining Application DDLs"
- Section 6.1.10, "Editing Application Definition"
- Section 6.1.11, "Troubleshooting"

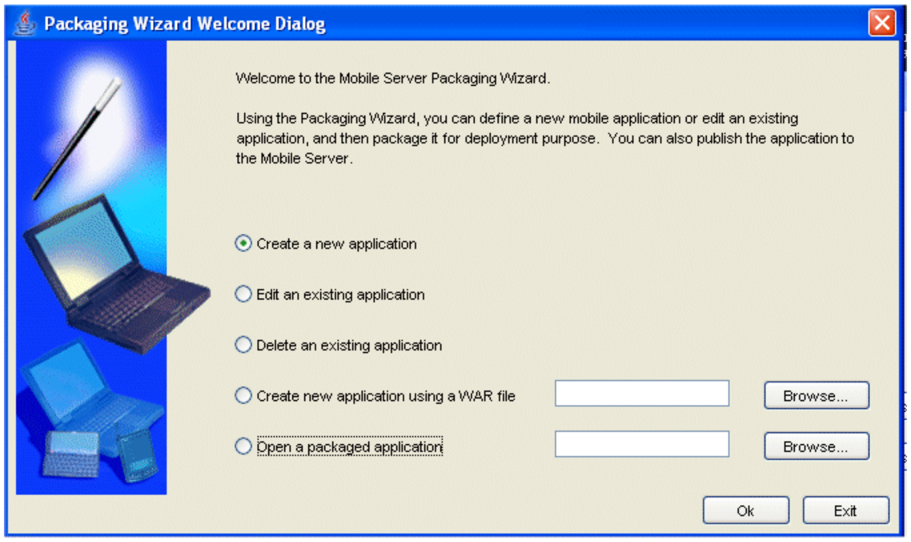
6.1.1 Starting the Packaging Wizard

To launch the Packaging Wizard, enter the following using a Command Prompt window.

```
wtgpack
```

Figure 6–1 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 6–1.

Figure 6–1 Packaging Wizard - Make A Selection Dialog



Packaging Wizard Welcome Dialog

Welcome to the Mobile Server Packaging Wizard.

Using the Packaging Wizard, you can define a new mobile application or edit an existing application, and then package it for deployment purpose. You can also publish the application to the Mobile Server.

[✓ Create a new application]
Edit an existing application
Delete an existing application
Create new application using a WAR file
Open a packaged application

Table 6–1 Make a Selection Dialog

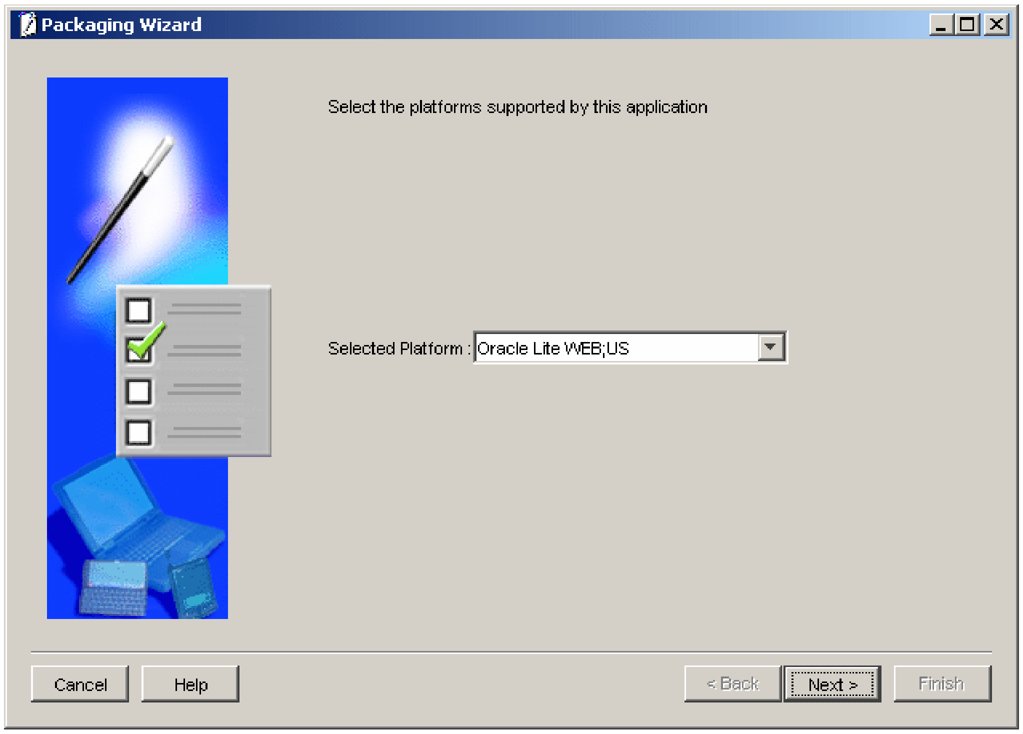
Feature	Description
Create a new application definition	Define a new Mobile application definition with the application implementation, publication items, and so on.
Edit an existing application definition	Edit an existing Mobile application definition. When selected, all existing application definitions 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.

Table 6–1 (Cont.) Make a Selection Dialog

Feature	**Description**
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 6–2 displays, this dialog enables you to specify a platform. If you are packaging a WAR file, this dialog only displays Web based platforms.

Figure 6–2 Select a Platform Dialog



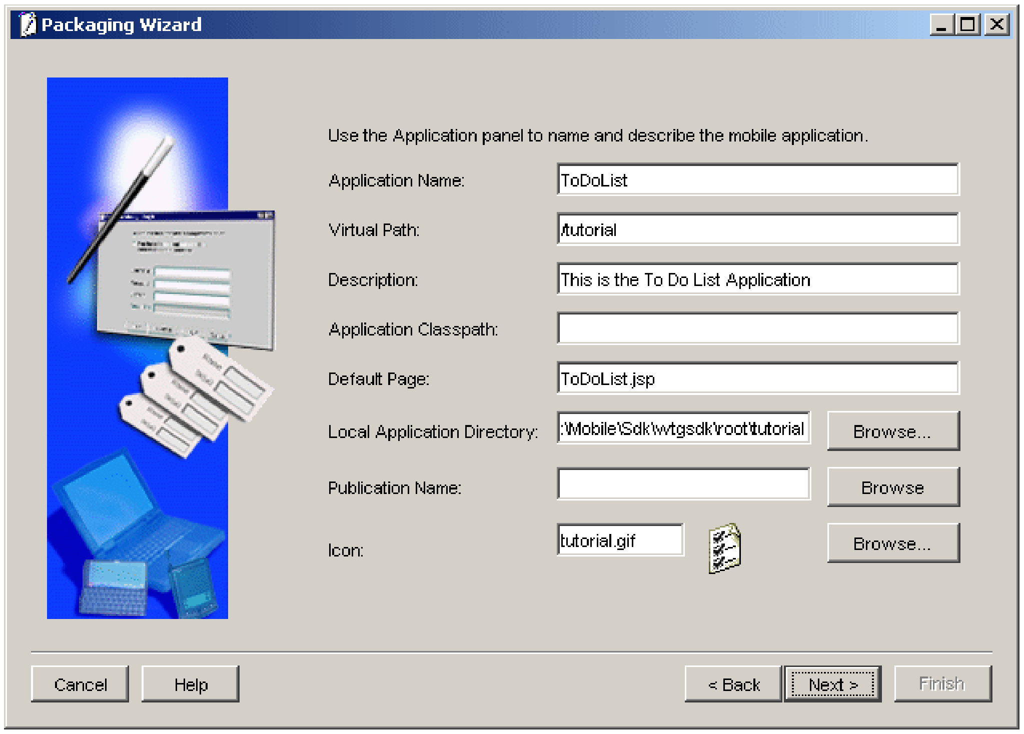
Select the platforms supported by this application

Selected Platform : Oracle Lite WEB;US

6.1.2 Specifying New Application Definition Details

Using the Application dialog, you can name a new Web-to-Go application and specify its storage location on the Mobile Server. As Figure 6–3 displays, the Application dialog includes the following fields.

Figure 6–3 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 6–2 describes the Application dialog.

Table 6–2 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 6–2 (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 6–3 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 6–2 (Cont.) Application Dialog Description

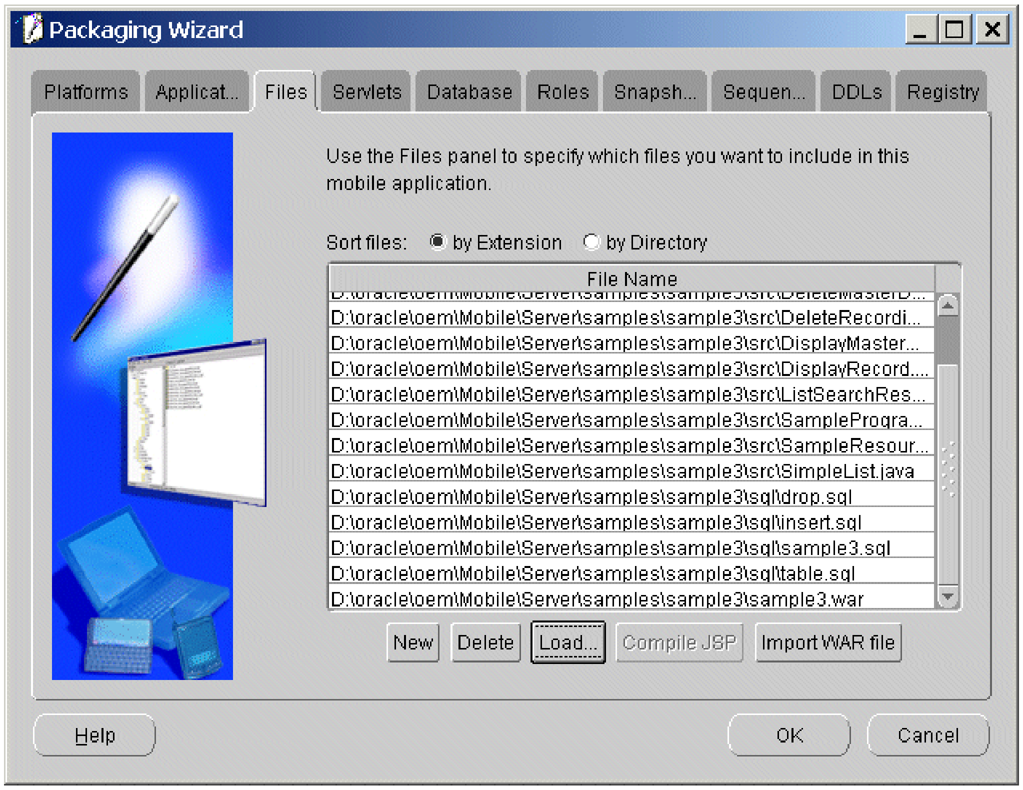
Field Name	Description	Required
Application Classpath [Web Applications Only]	The application classpath specifies where the classes (servlets, beans) for the application are located. The default application classpath is always the application root directory. To specify additional locations that the Mobile 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

6.1.3 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 6–3 describes, the Files tab contains the following field.

Figure 6–4 displays the Files tab.

Figure 6–4 Files Tab



Use the Files panel to specify which files you want to include in this mobile application.

Sort files: [✓ by Extension] by Directory

```
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\SamplePrograi...
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 6–3 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.

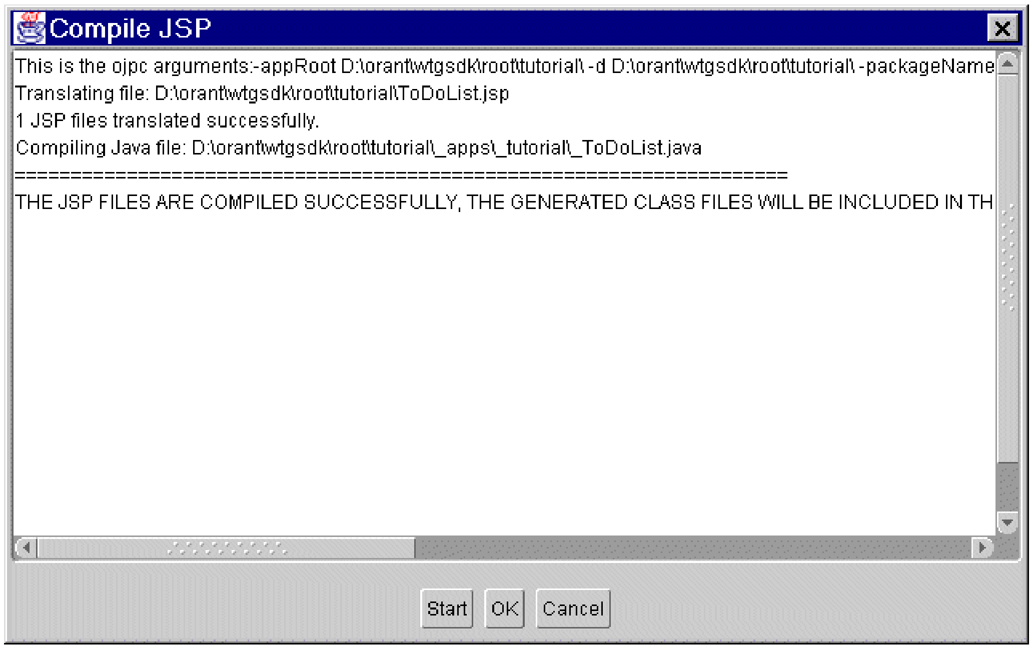
If you are importing a WAR file into an existing application, click the **Import WAR File** button on the 'Files' tab. Once you have specified the location of the WAR file, the 'Files' tab displays content of the WAR file.

6.1.3.1 Compile JSP (For Web 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 6–5 displays the Compile JSP Dialog.

Figure 6–5 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.

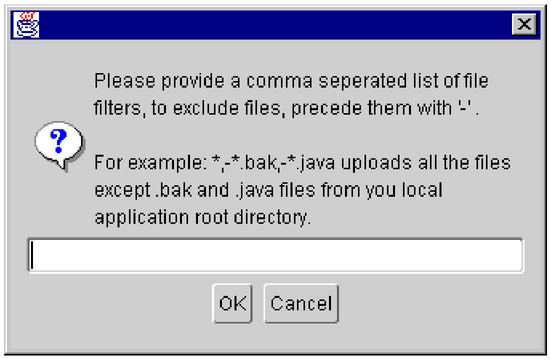
6.1.3.2 Filters

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 6–6 displays the Input dialog.

Figure 6–6 Input Dialog

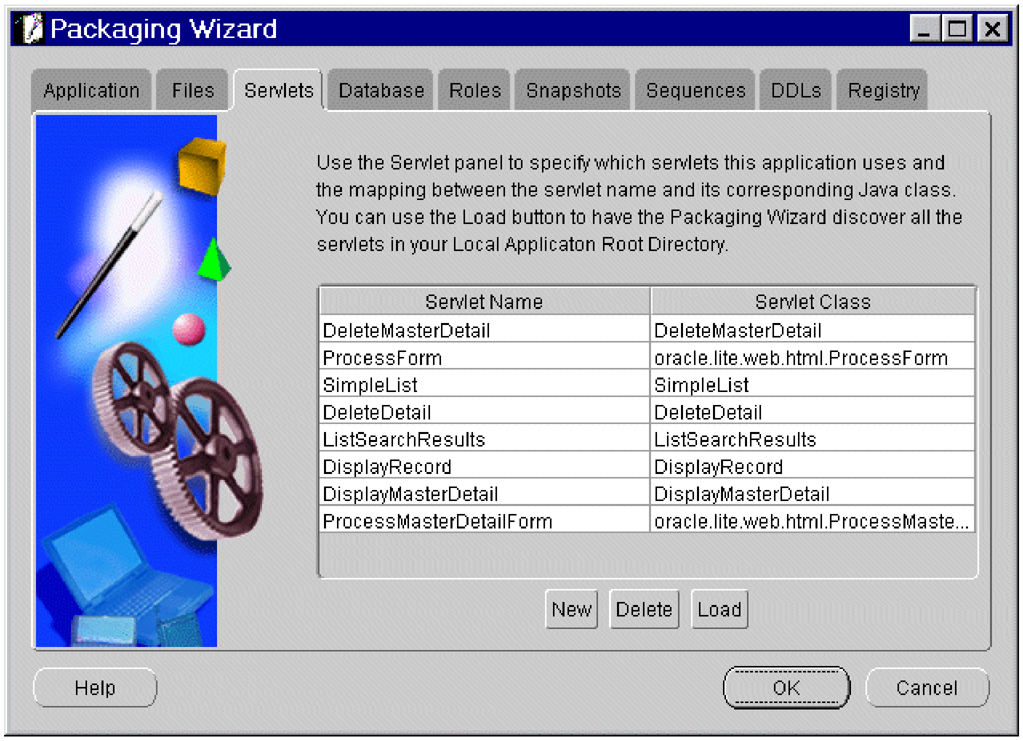

Please provide a comma seperated list of file filters, to exclude files, precede them with '-' .

For example: *,-*.bak,-*.java uploads all the files except .bak and .java files from you local application root directory.

6.1.4 Adding Servlets (For Web Applications Only)

The Packaging Wizard analyzes servlets in the File tab and defines them on the Mobile Server. As displayed in Figure 6–7, you can view your application's servlets in the Servlets tab.

Figure 6–7 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 6–4, the 'Servlets' tab includes the following fields.

Table 6–4 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

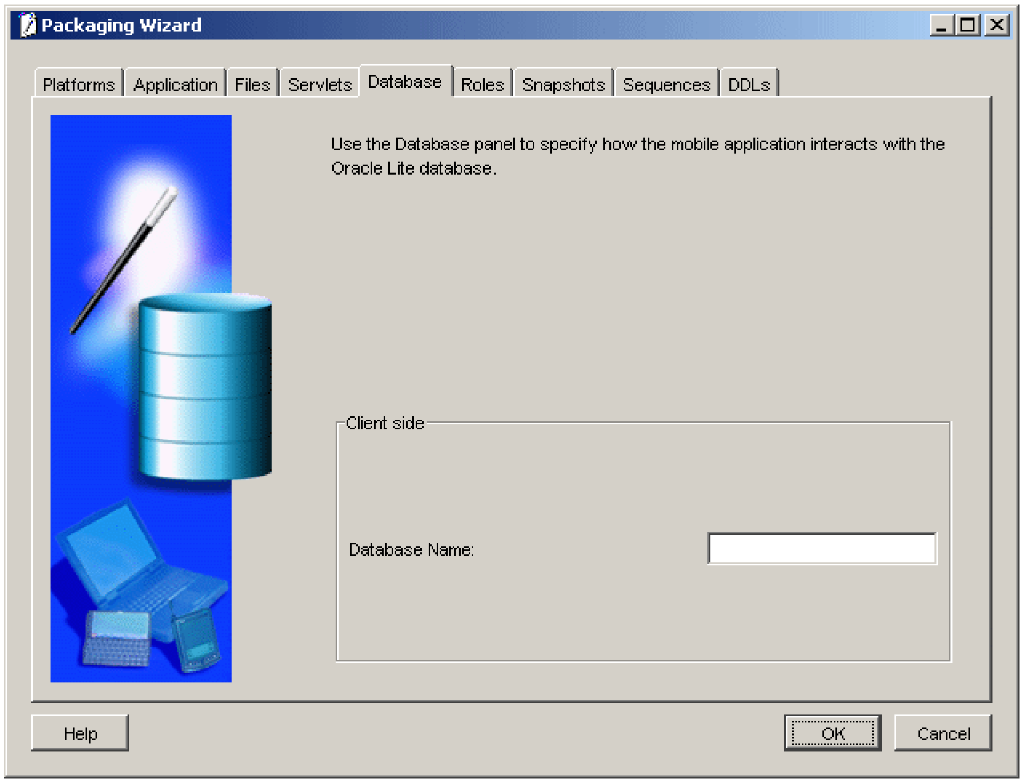
Using the 'Servlets' tab, you can add, remove, or load any servlets that are listed under the 'Servlets' tab. If you are creating a new application, the Packaging Wizard automatically lists all 'Servlets' based on files that are listed in the 'Files' tab. If you are editing an existing application, use the 'Load' button to locate and load individual servlets.

6.1.5 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 6–8 displays the Database tab.

Figure 6–8 Database Tab



Use the Database panel to specify how the mobile application interacts with the Oracle Lite database.

Database Name:

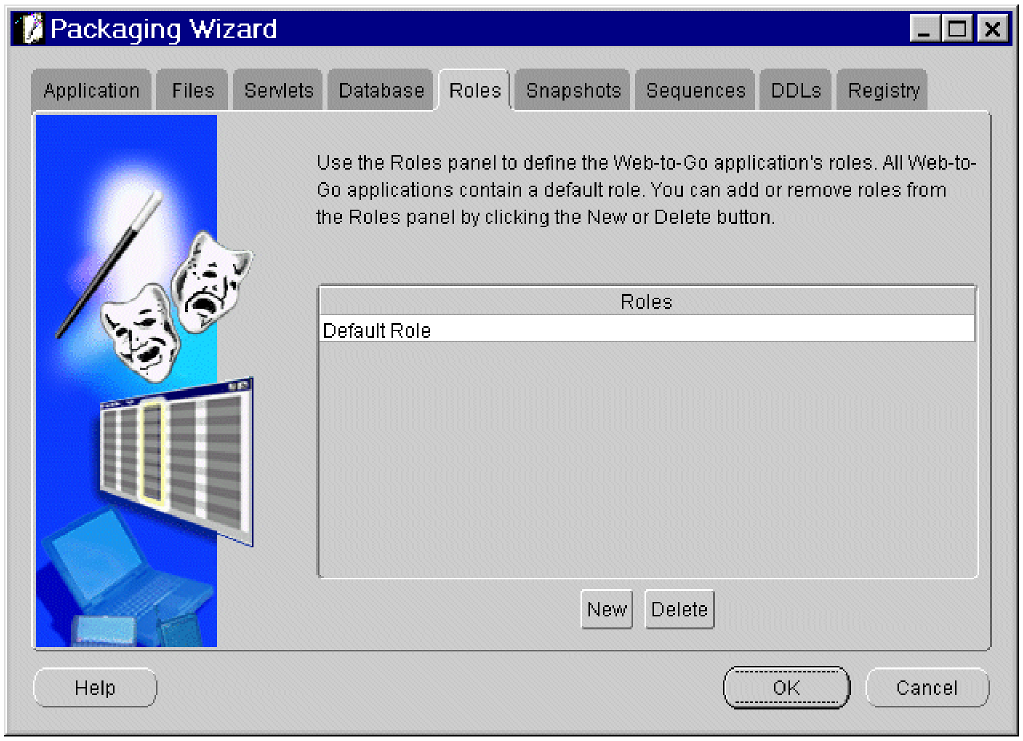
Enter the database name that you want to create on the client side. For example, a native Windows 32 application accesses the client database with this name. However, this is not required for Web applications.

6.1.6 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 6–9 displays the Roles tab.

Figure 6–9 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.

As described in Table 6–5, the Roles tab includes the following field.

Table 6–5 Roles Tab Description

Field	**Description**
Roles	Assigns roles to the Web-to-Go/Mobile Server application.

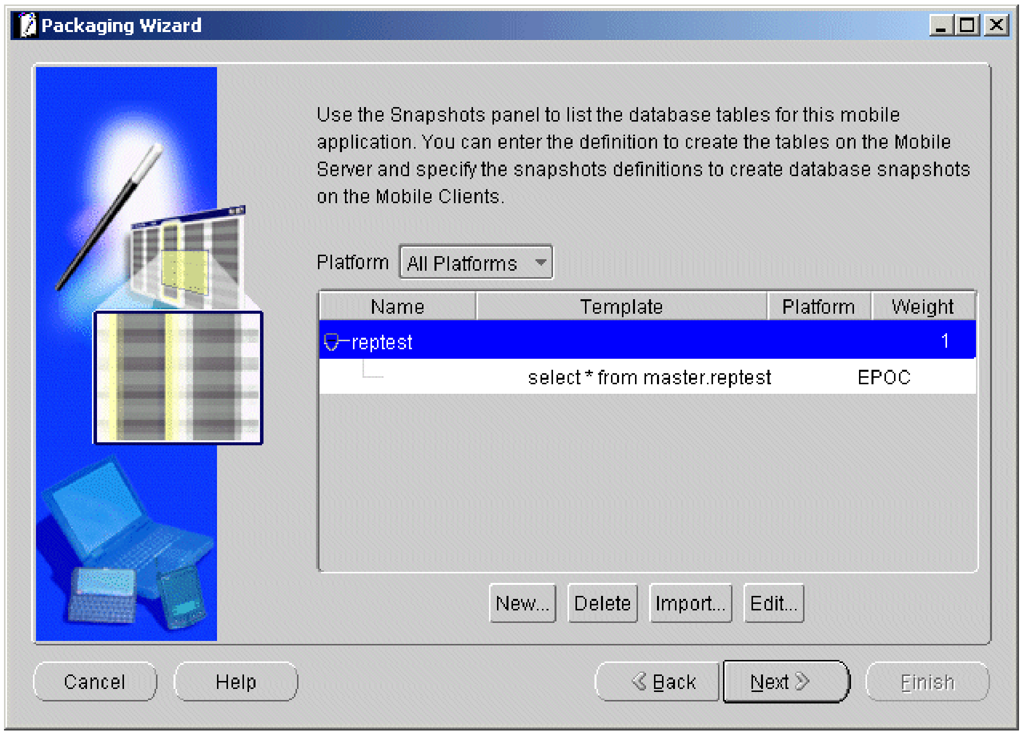
All Web-to-Go/Mobile Server applications contain a default role. You can add or remove roles from the Roles dialog using the 'New' or 'Delete' button.

6.1.7 Defining Snapshots for Replication

If you did not use MDW to create a subscription, then you can use the Snapshots tab to create replication snapshots for your application. A snapshot must have the same name as the database object such as a table or view. It must be unique across all applications. However, you must ensure that you use unique names when creating database objects. The Packaging Wizard enables you to create snapshots for the chosen platform. When you specify a view as the base object type, the Packaging Wizard enables you to specify the Parent Hint, Virtual Primary Hint, and the Primary Key Hint. For Web-to-Go, use the Windows 32 platform.

Figure 6–10 displays the Snapshots tab.

Figure 6–10 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 `wtgpack.exe` again.

Table 6–6 describes the Snapshots tab.

Table 6–6 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

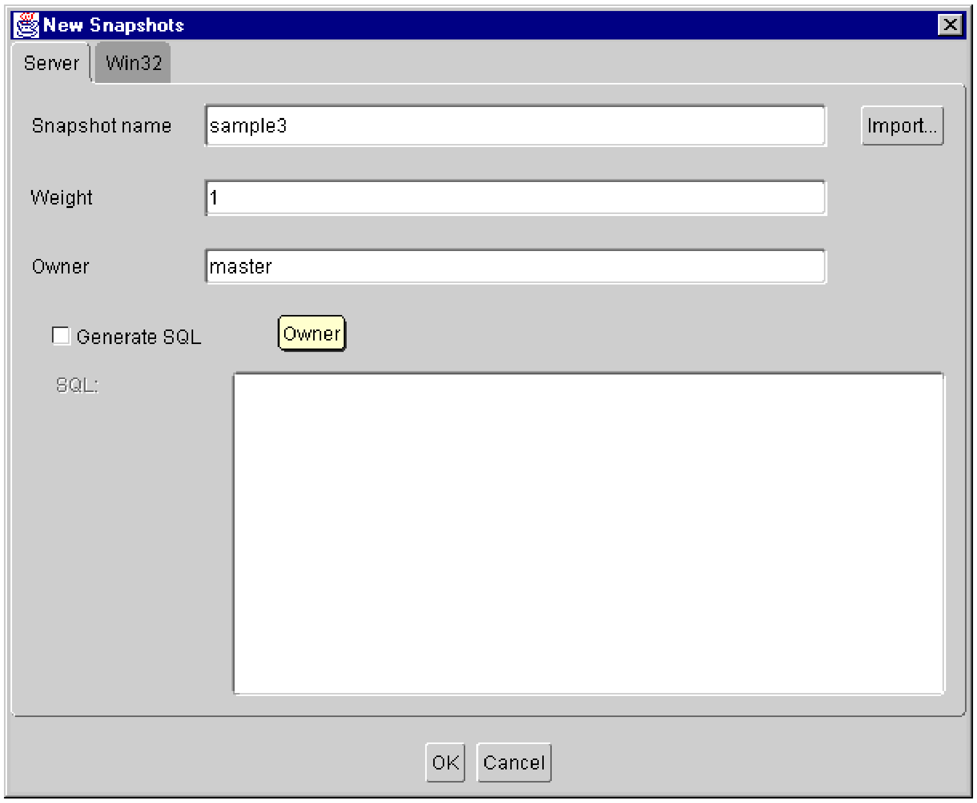
You can add or remove snapshots from the Snapshots tab using the 'New' or 'Delete' button. You can also import or edit snapshots using the 'Import' or 'Edit' button.

Note: You can import multiple snapshots from the Snapshots tab or import one when you create a new table from the 'New Table Dialog'.

6.1.7.1 Creating New Snapshots

To create new snapshots, click 'New'. The 'New Snapshots' dialog appears. As Figure 6–11 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 6–11 New Snapshots Dialog - Server Tab



Snapshot name: sample3

Weight: 1

Owner: master

☐ Generate SQL

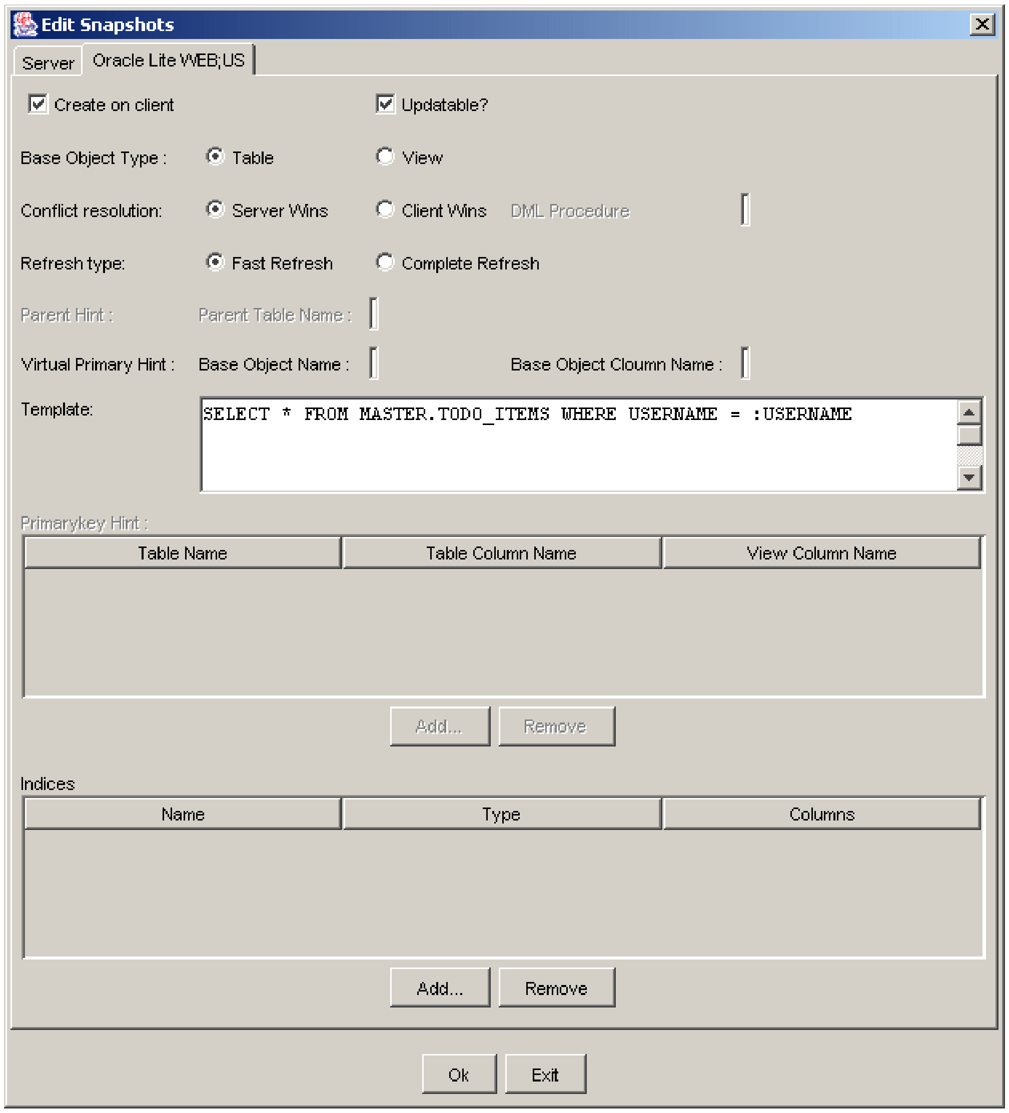
SQL:

For a description of Weight, see Section 6.1.7, "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 6–12 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 6–7 describes, the New Snapshots dialog displays the following information.

Table 6–7 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 6.1.7, "Defining Snapshots for Replication".

6.1.7.2 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 6–8 describes, enter values in the Win32 tab of the Edit Snapshots dialog.

Table 6–8 Win32 Tab - Edit Snapshots Dialog

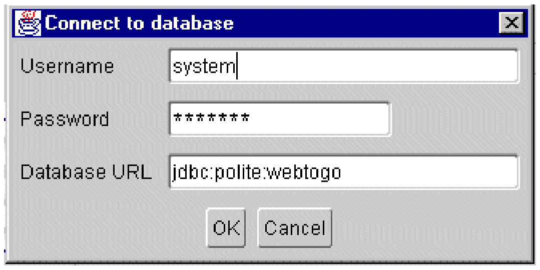
Field	Description
Create on Client	If selected, creates the snapshot on the client machine.
Updatable	If selected, creates an updatable snapshot of the specified table or view.
Base Object Type	Select **Table** to include a table as the base object type. or Select **View** to include a view as the base 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 6.1.7, "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.

6.1.7.3 Importing Snapshots

To import snapshots from an Oracle database or from Oracle Database Lite, click the 'Import' button. As Figure 6–13 describes, the database connection window appears if you have not specified a connection.

Figure 6–13 Connect to Database Dialog



Connect to database

Username: system

Password: *******

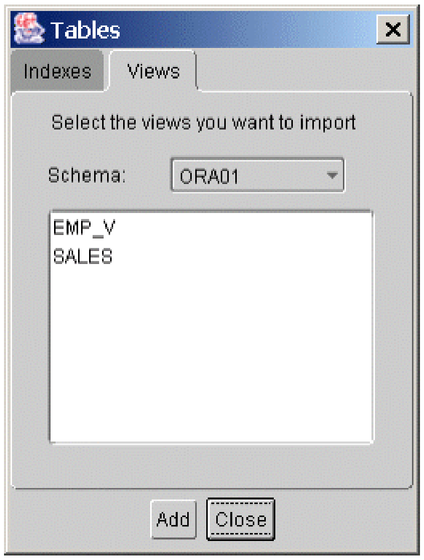
Database URL: jdbc:polite:webtogo

Enter the user name, password, and database URL for the Oracle database, or Oracle Database Lite from which you are importing your snapshot(s). The Tables window appears.

Note: Use the following format when entering the database URL for an Oracle database: `jdbc:oracle:thin:@<MOBILESERVER_JDBC_URL>`. For Oracle Database Lite, use `jdbc:polite:webtogo`.

Figure 6–14 displays the Tables dialog.

Figure 6–14 Tables Dialog



Tables

Select the views you want to import

Schema: ORA01

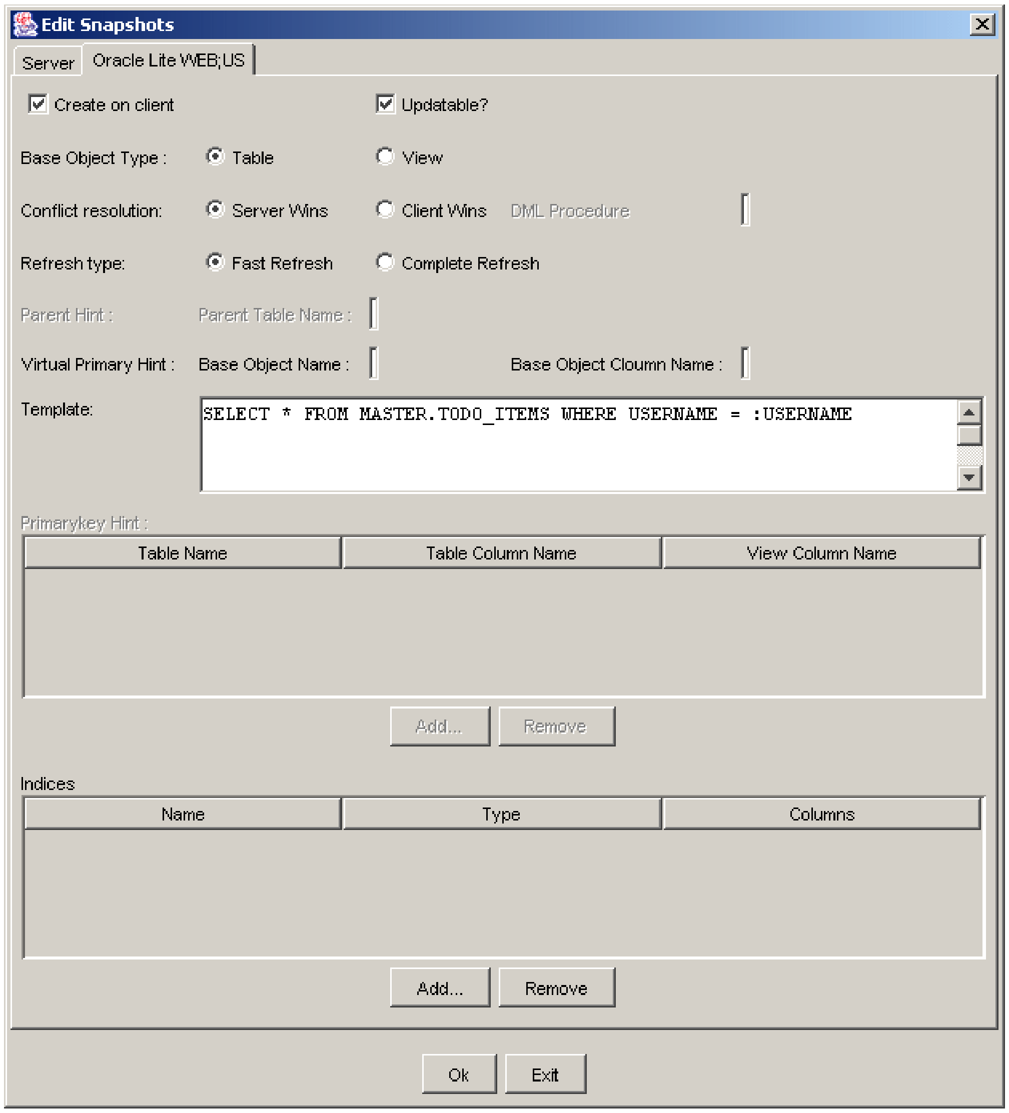
EMP_V
SALES

Click the Schema list and choose the required schema from the list displayed. The Tables dialog displays views associated with the chosen schema. Select the view that you need to import. Click Add and click Close.

6.1.7.4 Editing Snapshots

To edit a snapshot, select the snapshot from the Snapshots dialog and click Edit. As displayed in Figure 6–15, the Edit Snapshots dialog appears.

Figure 6–15 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 6–9, edit the snapshot by modifying the following features of the Edit Table window:

Table 6–9 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.

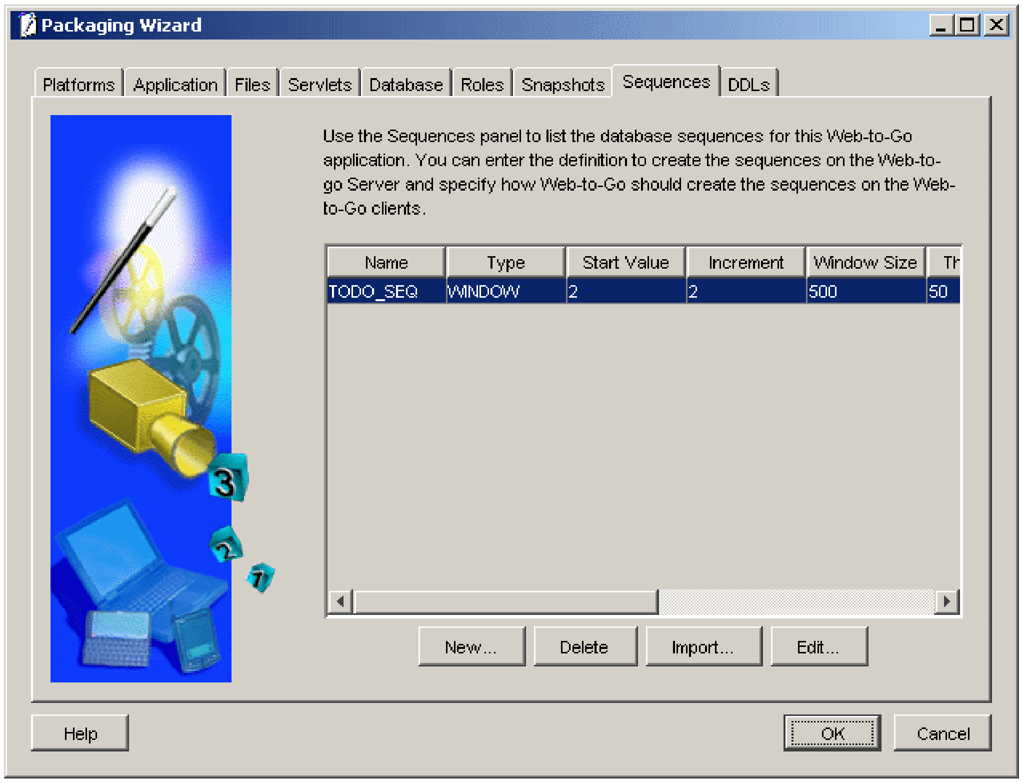
6.1.8 Defining Sequences for Replication

Use the Sequences dialog to define offline sequence support for the Web-to-Go application. Web-to-Go uses sequences to assign unique primary key values to an application before it disconnects and is in offline mode. These unique primary key values are used for replication when the client goes back online. Sequences are important because they eliminate replication conflicts by preventing duplicate primary key values across disconnected applications. All sequences must have a unique name. You can accomplish this by modifying your sequence names by preceding them with your application name.

Note: If you edit an existing application in the packaging wizard, which uses sequences, then you can not modify the increment value of the existing sequence.

Figure 6–16 displays the Sequences tab.

Figure 6–16 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 6–10, the Sequences dialog includes the following fields.

Table 6–10 Sequences Dialog Description

Field	Description	Required
Name	The name of the sequence used by the Web-to-Go application in disconnected mode.	Yes

Table 6–10 (Cont.) Sequences Dialog Description

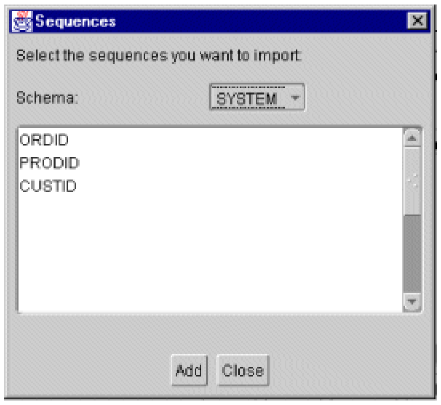
Field	Description	Required
Type	The type of sequence used by the Web-to-Go application in disconnected mode. Window. The window sequence assigns a unique range of values to each client. Window sequences are unique to each client and never overlap with those of other clients. When a client uses all the values in its sequence range, Web-to-Go recreates the sequence with a new, unique range of values the next time the client goes offline.	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
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 goes offline.	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.

6.1.8.1 Importing Sequences

To import sequences from an Oracle database, click the Import button. As Figure 6–17 displays, the Sequences dialog appears.

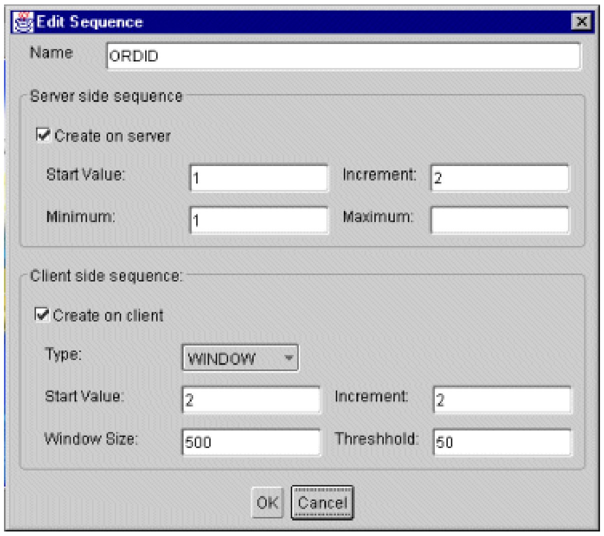
Figure 6–17   Sequences Dialog



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 6–18 displays, the Edit Sequences dialog appears.

Figure 6–18   Edit Sequences Dialog



Name: ORDID

Create on server: ✓
Start Value: 1
Increment: 2
Minimum: 1
Maximum:

Create on client: ✓
Type: WINDOW
Start Value: 2
Increment: 2
Window Size: 500
Threshhold: 50

As Table 6–11 describes, edit the sequence by modifying the following features of the Edit Sequences dialog.

Table 6–11   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.

Table 6–11 (Cont.) Edit Sequences Dialog Description

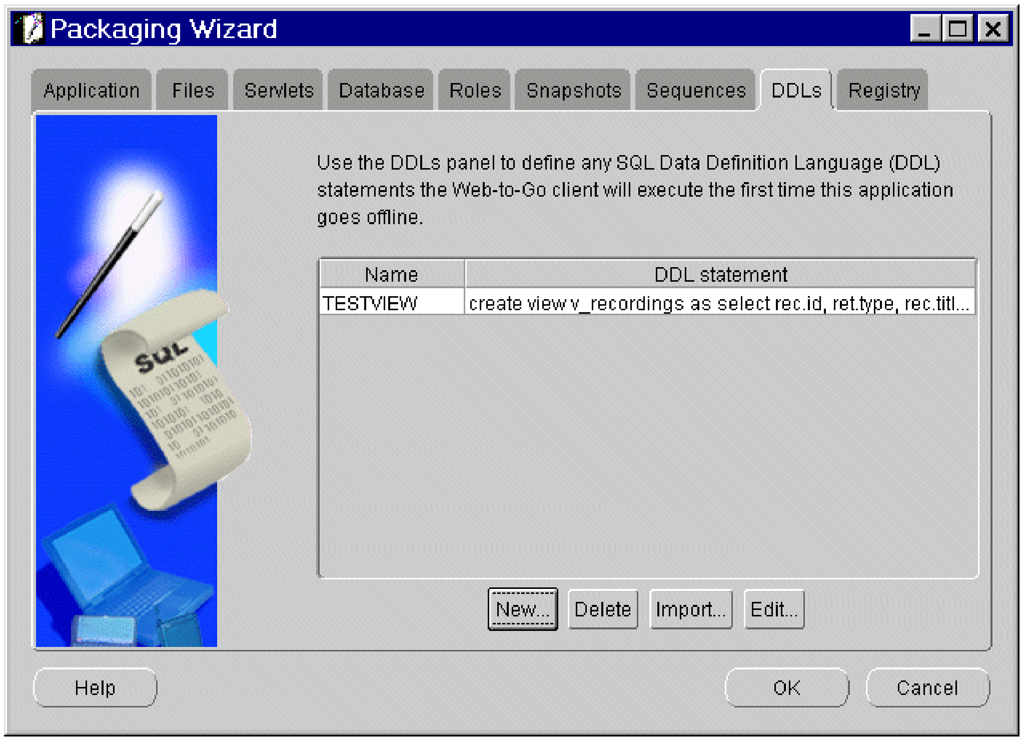
Feature	Description
Minimum	The minimum start value for an ascending sequence on the Oracle database. For example, an ascending sequence could start at 1 and continue in ascending order.
Maximum	The maximum start value for a descending sequence on the Oracle database. For example, a descending sequence 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.
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 goes offline. This information is not used by the leapfrog sequence.

6.1.9 Defining Application DDLs

Use the DDLs dialog to define any DDL (Data Definition Language) statements that the Web-to-Go application can execute the first time it goes offline. 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 6–19 displays the DDLs dialog.

Figure 6–19 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...

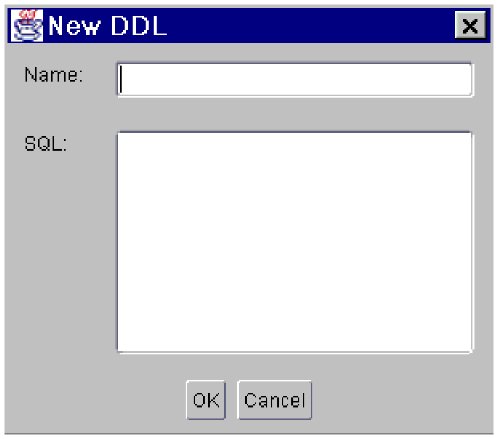
As described in Table 6–12, the DDLs dialog includes the following fields.

Table 6–12 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.
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 6–20.

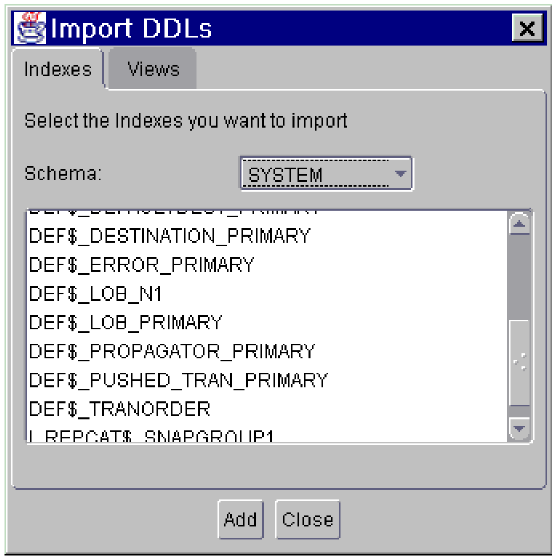
Figure 6–20 New DDL Dialog



6.1.9.1 Importing Views and Index Definitions

To import views and index definitions from an Oracle database, click the Import button. As displayed in Table 6–21, the Import DDLs dialog appears.

Figure 6–21 Import DDLs Dialog



DEF$_DESTINATION_PRIMARY
DEF$_ERROR_PRIMARY
DEF$_LOB_N1
DEF$_LOB_PRIMARY
DEF$_PROPAGATOR_PRIMARY
DEF$_PUSHED_TRAN_PRIMARY
DEF$_TRANORDER

To import an index definition, click the Indexes tab and then click the schema from which you want to import an index. Select the index you want to import, click Add, and then click Close.

To import a view definition, click the Views tab and then click the schema from which you want to import a view. Select the view you want to import, click Add, and then click Close.

6.1.10 Editing Application Definition

You can edit application definitions by launching the Packaging Wizard and selecting "Edit an existing application definition."

6.1.11 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.

```
wtgpack -d
```

6.2 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 6–13 describes basic features.

Table 6–13 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
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

Table 6–13 (Cont.) Packaging Wizard Synchronization Support

Function	Packaging Wizard	Mobile Manager	API
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 6–14 describes these features.

Table 6–14 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

7

Developing Mobile Web Applications

This document describes how to develop and test Web applications. Topics include:

- Section 7.1, "Setting up the Mobile Client"
- Section 7.2, "Developing and Testing the Application"

7.1 Setting up the Mobile Client

To install and set up the Mobile Client, see Section 19.6.1, "Step 1: Installing the Mobile Client for Web-to-Go".

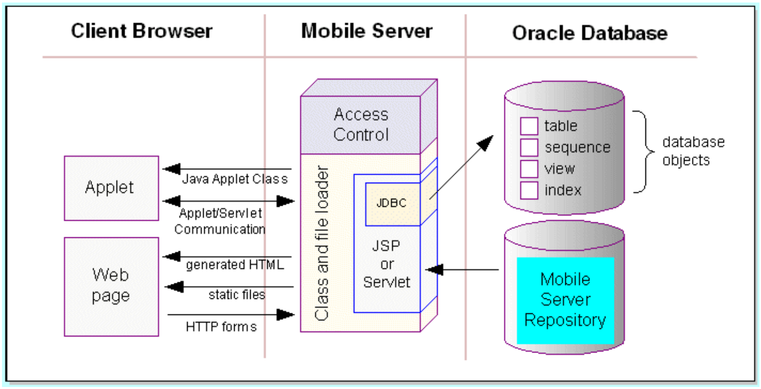
7.2 Developing and Testing the Application

Web-to-Go provides a high level Java API that provides easy-to-use functionality to developers of Mobile applications. Using this API, developers no longer need to write code for such functions as replication of database tables, online and offline database connections, security, directory locations, or deployment of applications to client devices.

In addition, the Mobile Development Kit allows developers to develop and debug Web-to-Go applications that contain Java applets, Java servlets, and JavaServer Pages (JSP).

Figure 7–1 displays the development architecture of the Mobile Server and the Oracle database.

Figure 7–1 Development Architecture



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

- Section 7.2.1, "Building Web-to-Go Applications"
- Section 7.2.2, "Application Roles"
- Section 7.2.3, "Developing JavaServer Pages"
- Section 7.2.4, "Developing Java Servlets for Web-to-Go"
- Section 7.2.5, "Using Web-to-Go Applets"
- Section 7.2.6, "Developing Applet JDBC Communication"
- Section 7.2.7, "Developing Applet Servlet Communication"
- Section 7.2.8, "Debugging Web-to-Go Applications"
- Section 7.2.9, "Customizing the Workspace Application"
- Section 7.2.10, "Using the Mobile Server Admin API"

7.2.1 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 7–1 provides examples of each database component type.

Table 7–1 Database Component Types

Component Type	Example
static	HTML files, image files (such as GIF and JPG), HTML templates
dynamic	Java servlets, Java applets and JavaServer pages
database	tables, snapshots, and sequences

7.2.1.1 Static Components

Static components are HTML files that do not change, such as graphical elements (GIF files and JPG files), and textual elements (HTML files and templates).

7.2.1.2 Dynamic Components

Java Applets, Java Servlets, and JavaServer Pages (JSP) are dynamic components that create dynamic Web pages. Java applets, create a rich graphical user interface, while Java servlets and JSPs extend server side functionality.

7.2.1.3 Database Components

Snapshots and sequences are the two database components that Web-to-Go supports. On the Mobile Server, the snapshot definition incorporates information about the table whose snapshot was taken. Web-to-Go also executes custom DDLs (Data Definition Language) statements, enabling the creation of such database objects as views and indexes.

Note: DDLs are only supported on Windows32 and WindowsCE platforms.

7.2.1.4 Database Connections

Database connections are both application based and session based. For a given session, Web-to-Go maintains a separate connection for each application. If an application runs multiple servlets simultaneously, they use the same connection object. This may occur if the application uses multiple frames or if a user accesses the application with two separate browser windows.

7.2.2 Application Roles

It is common for applications to display different functionality depending on the type of user who is running the application. For example, an application may show different menu items depending on whether manufacturing managers or shipping clerks are running the application.

You can accomplish this in Web-to-Go by defining application roles. The application behavior then changes depending on whether or not a user has a specific role.

In the above example, you can define the application role `MANAGER`. In your application code, where you generate the menu, you must check if the user has the role `MANAGER`, and display the correct menu items.

You will use the Packaging Wizard to define application roles in Web-to-Go. You can assign roles to users and groups through the Mobile Manager. However, it is up to the application developer to determine and implement application behavior, if the user has a specific role.

You can query the Web-to-Go user context to retrieve a list of roles that are created for users.

7.2.3 Developing JavaServer Pages

Web-to-Go handles HTTP requests for JavaServer Pages (JSP) using the Mobile Client Web Server, Mobile Server, and Mobile Client for Web-to-Go.

7.2.3.1 Mobile Server or Mobile Development Kit Web Server

After the Mobile Server receives an HTTP request for a JSP, it checks if the JSP source file and corresponding class file exist. If the class file exists and is newer than the JSP source file, the Mobile Server loads the Java class and executes the servlet.

If the class file does not exist, or is older than the JSP source file, the Mobile Server automatically converts the JSP source file into a Java source file and compiles it into a Java class under the `APP_HOME/_pages`. After the JSP has been converted and compiled, the Mobile Server (or the Mobile Development Kit Web Server) loads the Java class and executes the servlet.

7.2.3.2 Mobile Client for Web-to-Go

After the Mobile Client for Web-to-Go receives the HTTP request for a JavaServer page, the corresponding Java class is loaded from the `APP_HOME/_pages` directory and is executed. Since the Mobile Client for Web-to-Go assumes that the corresponding class file exists, you must convert the JSP source file into a class file. While deploying the application using the Packaging Wizard, you must include both the JSP source file and the corresponding class file. You can create the class files using the Packaging Wizard tool or manually, using the Oracle JSP (OJSP) command line translator.

List your JSP files in the Files panel of the Packaging Wizard and click **Compile** under the Files tab. The Packaging Wizard automatically locates all the JSP files that you have listed and automatically compiles all of them. The Packaging Wizard adds the `compile` class to the application package.

7.2.4 Developing Java Servlets for Web-to-Go

You develop Web-to-Go Java servlets with the Mobile Development Kit. The Mobile Development Kit for Web-to-Go simplifies the process of writing Mobile Server servlets. Before using the Mobile Development Kit for Web-to-Go, you must first install it on the development client. The Mobile Development Kit for Web-to-Go contains a Web server called the Mobile Client Web Server that executes Java servlets. You can use the Mobile Client Web Server to run and debug Java servlets.

- Section 7.2.4.1, "Limitations"
- Section 7.2.4.2, "Accessing Applications on the Mobile Development Kit for Web-to-Go"
- Section 7.2.4.3, "Creating a Servlet"
- Section 7.2.4.4, "Running a Servlet"
- Section 7.2.4.5, "Debugging a Servlet"
- Section 7.2.4.6, "Accessing the Schema Directly in Oracle Database Lite"

7.2.4.1 Limitations

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

- It contains no application repository. As a result, the Mobile Development Kit for Web-to-Go Web server loads all files and classes directly from the file system.
- Security and access control are disabled.
- Clients that connect to the Mobile Development Kit for Web-to-Go Web server cannot go off-line.
- It provides connection management only to Oracle Database Lite. It connects the user to the schema `SYSTEM` in the Oracle Database Lite named `webtogo`.

7.2.4.2 Accessing Applications on the Mobile Development Kit for Web-to-Go

You can access applications on the Mobile Development Kit for Web-to-Go Web server by performing the following steps.

1. To launch the Mobile Development Kit for Web-to-Go Web server, start the Command Prompt and enter the following.

   ```
   cd <ORACLE_HOME>\mobile\sdk\bin
   wtgdebug.exe
   ```

2. Use your browser to connect to the Mobile Development Kit for Web-to-Go Web server using the following URL.

   ```
   http://machine_name:7070/
   ```

   The Mobile Development Kit for Web-to-Go page displays icons that represent an application in the Mobile Client Web Server. Note that port 7070 is the default port for debugging Web-to-Go. For more information, see the file `webtogo.ora` under the following location.

```
<ORACLE_HOME>\mobile\sdk\bin\webtogo.ora
```

3. Click the icon of the application that you want to access.

7.2.4.3 Creating a Servlet

Web-to-Go uses servlets to handle HTTP client requests. Servlets handle HTTP client requests by performing one of the following tasks.

- Creating dynamic HTML content and returning it to the browser.
- Processing and submitting HTML forms using an `HTTP POST` request.

Servlets must extend the `HttpServlet` abstract class defined in the Java Servlet API. The following is a servlet example.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HelloWorld extends HttpServlet
{
   /**
   * Process the HTTP POST method
   */

   public void doPost (HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
   {
     writeOutput("doPost", request, response);
   }

   /**
   * Process the HTTP GET method
   */
   public void doGet (HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
   {
      writeOutput("doGet", request, response);
   }

   /**
   * Write the actual output
   */

   public void writeOutput (String method, HttpServletRequest  request,
                            HttpServletResponse response)
   throws ServletException, IOException
   {
      PrintWriter out;

      // set content type
      response.setContentType("text/html");

      // Write the response
      out = response.getWriter();

      out.println("<HTML><HEAD><TITLE>");
      out.println("Hello World");
```

```
        out.println("</TITLE></HEAD><BODY>");
        out.println("<P>This is output from HelloWorld "+method+"().");
        out.println("</BODY></HTML>");
        out.close();
    }
}
```

7.2.4.3.1 Packages Web-to-Go provides the following Java package:

```
oracle.lite.web.applet
```

This package contains the classes to be used with Web-to-Go applets. It contains the `AppletProxy` class which is used as a proxy for Web-to-Go applets requiring JDBC connections or communicating with a servlet on the Mobile Server. It also contains a few more classes which are used by the `AppletProxy` class to communicate with the Mobile Server. For more information, see the `oracle.lite.web.applet` package as documented in the *Oracle Database Lite API Specification*.

7.2.4.3.2 Web-to-Go User Context Web-to-Go creates a user context (or user profile) for every user who logs in to Web-to-Go. Web 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
- Mode the user is running in (online or offline)
- 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");
    .
    .
    .
}
```

7.2.4.3.3 Database Connectivity in Java Code Servlets can obtain a connection to the Oracle database, using the following statement.

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

7.2.4.3.4 Accessing the Mobile Server Repository Servlets can open or create a new file in the application repository. Access to the Mobile Server Repository is provided through the servlet context, which can be obtained by calling the `getServletContext()` from within the servlet. For example:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
   // Retrieve the servlet context
   ServletContext ctxt = getServletContext();

   // Open an input stream to the file input.html in the Mobile Server Repository
   // All file names are relative to the application's repository directory
   InputStream in = ctx.getResourceAsStream("input.html");

  // Open an output stream to the file output.html in the Mobile Server Repository
  // All file names are relative to the application's repository directory
  URL            url = ctxt.getResource ("output.html");
  URLConnection  conn = url.openConnection();
  OutputStream   out  = conn.getOutputStream();
  ...
}
```

7.2.4.4 Running a Servlet

Before you can execute the servlet, perform the following:

- Section 7.2.4.4.1, "Registering Servlets Using wtgpack.exe"
- Section 7.2.4.4.2, "The webtogo.ora File"
- Section 7.2.4.4.3, "Using wtgdebug.exe"
- Section 7.2.4.4.4, "Using WebtoGoServer.class"
- Section 7.2.4.4.5, "Controlling Web Server Properties"
- Section 7.2.4.4.6, "Registering MIME Types"

7.2.4.4.1 Registering Servlets Using wtgpack.exe Before you can access servlets from the browser, you need to register them with the Mobile Client Web Server. To register servlets, you must first register the application and then add the servlets to it. As Web-to-Go enables you to register multiple applications, it displays a list of all registered applications.

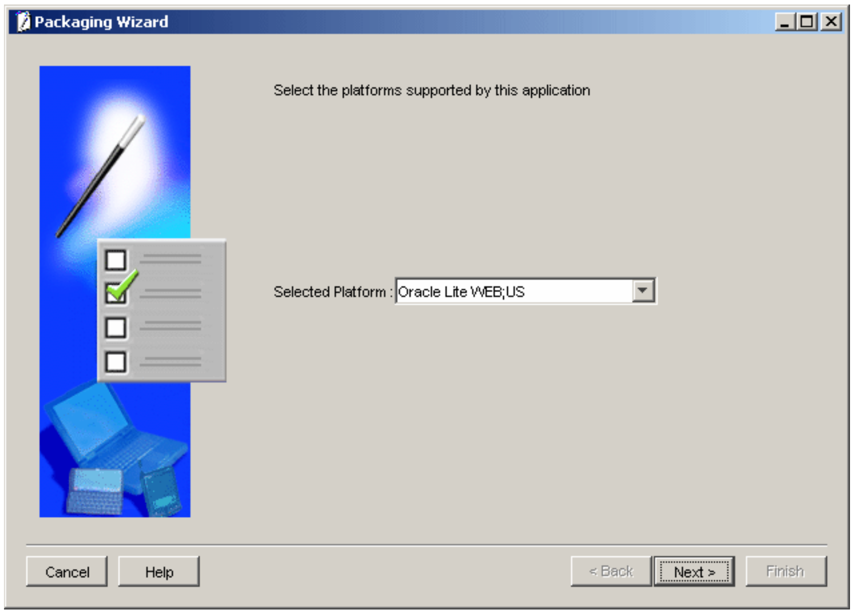
The Mobile Development Kit for Web-to-Go includes the Packaging Wizard, a tool for registering applications and servlets. You can invoke the Packaging Wizard by entering the following at the command line.

```
C:\> wtgpack -d
```

Initially, you select whether to create a new application or to continue work on an existing application.

Figure 7–2 displays the Make a Selection dialog.

Figure 7–2 Make a Selection Panel



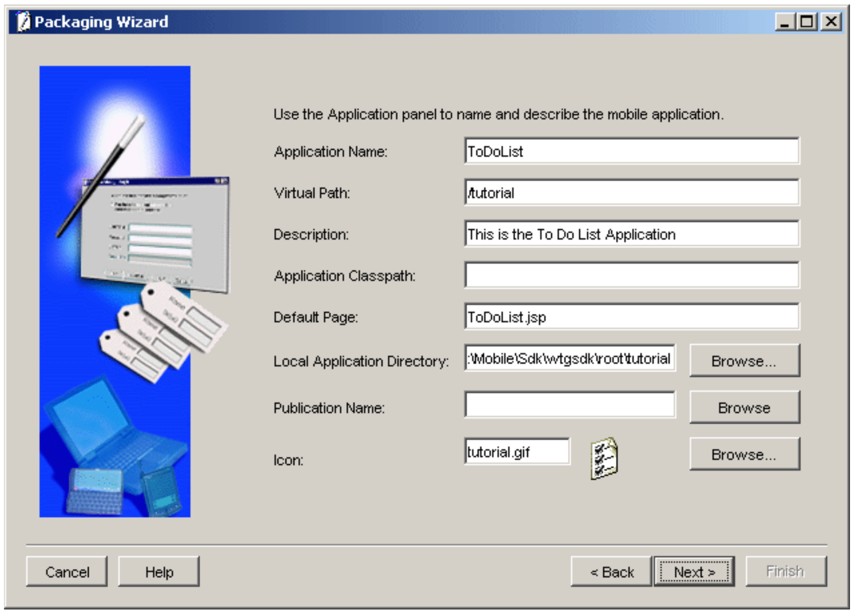
Select the platforms supported by this application

Selected Platform : Oracle Lite WEB;US

After you make your selection and click **OK**, the Applications dialog appears.

Figure 7–3 displays the Applications dialog.

Figure 7–3 Applications 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

For detailed instructions on how to use the Packaging Wizard, see Chapter 19, "Building Mobile Web Applications: A Tutorial".

7.2.4.4.2 The webtogo.ora File The configuration information for the Web server and the Packaging Wizard is stored in the `webtogo.ora` file.

Table 7–2 describes `webtogo.ora` parameters.

Table 7–2 Webtogo.ora Parameters

Parameter Name	Description
`ROOT_DIR`	The Mobile Server expands all file paths that are relative to its root directory. You can change the root directory by modifying the value of the parameter named `ROOT_DIR` in the `webtogo.ora` file. The default parameter value is given below. `<`*`ORACLE_HOME`*`>\mobile\sdk\wtgsdk\root`
`PORT`	The port on which the Web server listens. The default value is `80`. The default value for the Mobile Client Web Server is `7070`.
`XMLFILE`	The XML file that contains the application information. The Packaging Wizard creates and maintains the XML file. You can modify the XML file using the Packaging Wizard.

For more information, refer the discussion of initialization parameters in the *Oracle Database Lite Administration and Deployment Guide*.

7.2.4.4.3 Using wtgdebug.exe **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.

7.2.4.4.4 Using WebtoGoServer.class Because the Mobile Client Web Server is written in Java, you can run it inside a Java Virtual Machine (JVM), instead of running the `wtgdebug.exe`. Running the Mobile Client Web Server in the JVM enables you to debug Web-to-Go applications by running the Mobile Client Web Server inside a Java debugger. You can use the class `oracle.lite.web.server.WebToGoServer` to start the Java version of the Mobile Client Web Server.

Before you can use the Java version of the Mobile Client Web Server, you need to add the following jar files to your `CLASSPATH`.

```
<ORACLE_HOME>\mobile\sdk\bin\webtogo.jar

<OLITE_HOME>\bin\olite40.jar

<ORACLE_HOME>\mobile\classes\xmlparser.jar

<ORACLE_HOME>\mobile\classes\classgen.jar

<ORACLE_HOME>\mobile\classes\ojsp.jar

<ORACLE_HOME>\mobile\classes\jssl-1_2.jar

<ORACLE_HOME>\mobile\classes\javax-ssl-1_2.jar
```

```
<ORACLE_HOME>\mobile\classes\consolidator.jar
```

You must add the location of your application classes, such as `<ORACLE_HOME>\mobile\sdk\wtgsdk\root`, to the `CLASSPATH`. For more information, refer Section 7.2.5, "Using Web-to-Go Applets".

To control and start the Mobile Client Web Server, the file `RunWebServer.java` demonstrates how to use the class `oracle.lite.web.server.WebToGoServer`. This file is located in the following directory.

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

To start the Mobile Client Web Server, perform the following steps.

1. Using the following command, compile the Java file.

   ```
   javac RunWebServer.java
   ```

2. Run the Mobile Client Web Server using the following command.

   ```
   java RunWebServer
   ```

7.2.4.4.5 Controlling Web Server Properties

You can set various properties of the Mobile Client Web Server dynamically using the method `WebToGoServer.setProperty()`. These values override the values in the file `webtogo.ora`. The following table lists properties that can be controlled.

Table 7–3 describes Mobile Client Web Server properties.

Table 7–3 Property Controls

Property	Definition
config_file	The configuration file to use. For more information, refer Section 7.2.4.4.2, "The webtogo.ora File".
port	The port on which the Mobile Development Kit Web Server listens.
debug	Enables debugging. Set the value as "0" if you want to view debug messages.
log_file	The debug log file. If specified, debug messages are sent to this file, otherwise the messages are displayed to the screen.
root_dir	The root directory. Overrides `ROOT_DIR` in `webtogo.ora`. For more information, refer Section 7.2.4.4.2, "The webtogo.ora File".

For example,

```
WebToGoServer.setProperty ("config_file",
                           "d:\\orant\\mobile\\server\\bin\\webtogo.ora");
WebToGoServer.setProperty ("debug", "0");
WebToGoServer.setProperty ("port", "80");
```

7.2.4.4.6 Registering MIME Types

You can create your own servlet that handles all HTTP requests for files with a particular file extension. For example, you can have a servlet named `ASPHandler` which handles all requests that end in `'asp'`.

You can register this handler with the Mobile Client Web Server using the method `WebToGoServer.addMIMEHandler()`. For example,

```
WebToGoServer.addMIMEHandler("text/asp", "asp", "ASPHandler")
```

7.2.4.5 Debugging a Servlet

In software development, debuggers are often used to examine code and fix bugs. With Web-to-Go, you can use a debugger to test applications containing Java servlets. By running these servlets inside a Java debugger, you can set breakpoints in the Java code, view the code, examine threads, and evaluate objects. You can debug Web-to-Go applications using the `WebToGoServer` class inside a debugger. For more information, see Section 7.2.8.1, "Running Sample 1 Using Oracle9i JDeveloper".

7.2.4.6 Accessing the Schema Directly in Oracle Database Lite

The Mobile Development Kit for Web-to-Go automatically creates a database connection to Oracle Database Lite. This database connection connects to the database schema `SYSTEM`. Within your servlet code, you can obtain this connection from the HTTP request. You can also connect to Oracle Database Lite directly using ODBC. Connecting to Oracle Database Lite directly by using ODBC is helpful for performing the following tasks.

- Creating schema objects such as tables, view and sequences
- Manually checking the contents table

To connect to Oracle Database Lite, launch `msql` using the Command Prompt.

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

7.2.5 Using Web-to-Go Applets

Web-to-Go supports Java applets. For security reasons, Web-to-Go applets must communicate with the Mobile Server or the Oracle database by using a proxy class. The `AppletProxy` class acts as a proxy for Web-to-Go applets and provides the applet with the required methods for communicating with the Web-to-Go servlet or for making a JDBC connection. An instance of the `AppletProxy` should be created while instantiating the applet. Once the instance of the `AppletProxy` class is created, the `AppletProxy` object communicates with the Mobile Server and derives all the requisite information to connect to the server or to make a JDBC connection to the Oracle database.

7.2.5.1 Creating the Web-to-Go Applet

The Web-to-Go applet extends the `java.applet.Applet`. When the `init()` method initializes the Web-to-Go applet, it creates an instance of the `AppletProxy` class by passing the Applet reference as the parameter. Once you create an instance of the `AppletProxy` class, you can use different methods of the `AppletProxy` class for communicating with the servlet or for establishing a JDBC connection with the Oracle database. For example,

```
import oracle.lite.web.applet.*;
public class AppApplet extends Applet
{

   public void init()
   {
      ..
      ..
      // Create Instance and pass Reference of applet as parameter
      proxy = new AppletProxy(this);
   }
   AppletProxy proxy;
}
```

The applet can use the following methods to communicate with the servlet. Each method requires an instance of the `AppletProxy` class.

- `getResultObject()`
- `setSessionId()`
- `showDocument()`

The applet can use the `getConnection()` method to establish a JDBC connection with the database.

7.2.5.2 Creating the HTML Page for the Applet

The Web-to-Go applet is launched from an HTML page that contains the following tags.

```
<html>
<body>
<applet ARCHIVE="/webtogo/wtgapplet.jar" CODE="MyApplet.class" WIDTH=200
HEIGHT=100>
<PARAM NAME="ORACLE_LITE_WEB_SESSION_ID"  VALUE="123">
</applet>
</body>
</html>
```

The `AppletProxy` class uses the value of the `ORACLE_LITE_WEB_SESSION_ID` parameter to obtain the `SessionID` from the Mobile Server. The `SessionID` is subsequently added to every request an applet makes to a servlet. You can write the HTML code in a static HTML page or you can generate it from a servlet.

7.2.5.2.1 Static HTML Page Web-to-Go can automatically add the parameter to any static page containing the `APPLET` tag. For this option, you must change the HTML page's extension to `.ahtml` as demonstrated in the following syntax.

page_name.ahtml

When the client accesses the HTML page, a Web-to-Go system servlet adds the required `<PARAM>` tag for the `ORACLE_LITE_WEB_SESSION_ID` parameter, to the HTML output. For example,

```
<PARAM NAME="ORACLE_LITE_WEB_SESSION_ID" VALUE="123">
```

The Web-to-Go system servlet sets the `VALUE` attribute to your Web-to-Go `SessionID`.

7.2.5.2.2 HTML Page Generated from a Servlet You can also dynamically generate the HTML page that contains the `<APPLET>` tag. When you generate the HTML page dynamically, you must add the `SessionID` parameter manually. You can retrieve the `SessionID` information from the `oraUserProfile` as follows.

```
import oracle.lite.web.html.*;
import oracle.lite.web.servlet.*;

public class AppServlet extends HttpServlet
{

   public void doGet(HttpServletRequest req, HttpServletResponse resp)
   {
```

```
            PrintWriter out = new PrintWriter(resp.getOutputStream());
            out.println("<HTML>");
            out.println("<BODY>");
            out.println("<APPLET ARCHIVE="/webtogo/wtgapplet.jar"
                        CODE='MyApplet.class' WIDTH=200 HEIGHT=100>");
            // Add these lines to add one more PARAM tag in html page
            // This code should be added in-between  <APPLET> and  </APPLET> tag
            OraHttpServletRequest ora_request    = (OraHttpServletRequest) req;
            OraUserProfile        oraUserProfile = ora_request.getUserProfile();
            out.println(" <PARAM NAME=\"ORACLE_LITE_WEB_SESSION_ID\" VALUE=\""
                        +oraUserProfile.getAppletSessionId(req)+"\"> ");
            out.println("</APPLET>");
            out.println("</BODY>");
            out.println("</HTML>");
            out.close();
         }
      }
```

7.2.6 Developing Applet JDBC Communication

You can develop Java applets that access the database using a JDBC connection. Once you create an instance of the `AppletProxy` class, you must use the `getConnection()method` of the `AppletProxy` class to obtain a JDBC connection. The `getConnection() method` returns the `JDBCConnection` object.

> **Note:** The `AppletProxy` class is described in Section 7.2.5.1, "Creating the Web-to-Go Applet".

7.2.6.1 getConnection()

You can use the `getConnection()` method to obtain a `JDBCConnection`. The `getConnection()` method determines whether the connection mode is online or offline and provides the correct database connection (Oracle database for online mode and Oracle Database Lite for offline mode) 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;
}
```

7.2.6.2 Design Issue

The Web-to-Go applet holds the database connection even after the user exits Web-to-Go. The applet maintains the connection even if the user types a new URL in the browser or clicks the **Back** button. Web-to-Go application designers must ensure that their applications explicitly close the database connection when the user exits Web-to-Go.

Example

You can close the connection by calling the following statement.

```
dBConnection.close()
```

7.2.7 Developing Applet Servlet Communication

You can develop Java applets that communicate with Java servlets in the Web-to-Go environment. When a client first connects to the Mobile Server, the server generates a `SessionID` and sends it back to the client. Each subsequent client request to the server contains this `SessionID`. The Mobile Server authenticates the `SessionID` before executing the client's request. When applets communicate with Web-to-Go servlets, each applet request must also contain this `SessionID`. The `setSessionId` method in the `AppletProxy` class can be used to add the `SessionID` to each applet request. The `AppletProxy` class also contains other methods that provide communication between applets and servlets.

Note: The `getResultObject()` and `showDocument()` methods can be used to communicate with the Java servlet. Use the `setSessionID` method if you want to create your own URL connection object.

7.2.7.1 Creating the Web-to-Go Servlet

Servlets must extend the `HttpServlet` abstract class defined in the Java Servlet API. The following example creates a servlet called `HelloWorld` that extends the `HttpServlet` class. The servlet sends the `Hello World` string to the applet that calls it as an object.

Example

```
public class HelloWorld extends HttpServlet
{

   public void doGet (HttpServletRequest request, HttpServletResponse response)
   {
      ObjectOutputStream out = new ObjectOutputStream (resp.getOutputStream());
      Object obj = (Object) "Hello World" ;
      out.writeObject(obj);
      out.close();
   }
}
```

7.2.7.1.1 getResultObject() The Web-to-Go applet uses the `getResultObject()` method to communicate with the Web-to-Go servlet by passing the servlet URL and the `ServletParameter` object as parameters. The servlet responds to the applet request with a text string. The `ServletParameter` object can be either an object that can be serialized or a string containing `name/value` pairs. If the servlet accepts parameters, you can call the `getResultObject` method and pass the servlet parameters as one of the arguments.

Example

```
public Object getResult()
{

   java.net.URL url = new URL("http://www.foo.com/EmpServlet");
   String ServletParameter = "empname=John";
   Object resultObject = proxy.getResultObject(url, ServletParameter);
   return resultObject;
}
```

7.2.7.1.2 setSessionID() You can use the `setSessionID` method for adding a `SessionID` to an existing `URLConnection` object. When you write the applet-servlet communication mechanism, call `setSessionID` (URLConnection) at the end of the method. The method adds a `SessionID` to the passed `URLConnection` object and then returns the `URLConnection` object.

Example

```
public void YourMethod()
{

   java.net.URL url = new URL("http://www.foo.com/MyServlet");
   java.net.URLConnection con = url.URLConnection();
   ..
   ..
   ..
   // pass the URLConnection to the method setSessionId
   con = proxy.setSessionID(con);
   // Do whatever you want to do with this URLConnection object
   ObjectOutputStream out = new ObjectOutputStream(con.getOutputStream());
   out.writeObject(obj);
   out.flush();
   out.close();
}
```

7.2.7.1.3 showDocument() The `showDocument` method displays any static document including those with a suffix of `.html`, `.doc`, `.xls`, or any other one defined by the user. The `showDocument` method retrieves these documents from the Mobile Server and displays them in the client browser. To display documents, a user must have access permissions for the document and must have the correct `MIME` type set in the Mobile Server. The `showDocument` (String `relativeDocUrl`, String `winName`) method displays the document in a different browser window identified by a window name that is passed in the `winName` parameter. The following method launches the help file from the server in a browser window named `'helpwin'`.

Example

```
public void showHelp()
{

   String relativeDocUrl = "Help/HelpIndex.html";
   proxy.showDocument (url, helpWin);
}
```

To show the document in the same browser window as your applet, use call `showDocument(url)` as given below.

```
public void showHelp()
{
   String relativeDocUrl = "Help/HelpIndex.html";
   proxy.showDocument (url);
}
```

7.2.8 Debugging Web-to-Go Applications

You can run Web-to-Go applications inside a Java debugger if you have already installed the Mobile Development Kit for Web-to-Go and a Java debugger, such as the Oracle9*i* JDeveloper, Borland's JBuilder, or Visual J++. The example in this section assumes you are using Oracle9*i* JDeveloper. However, most of the information provided is also relevant to other debuggers.

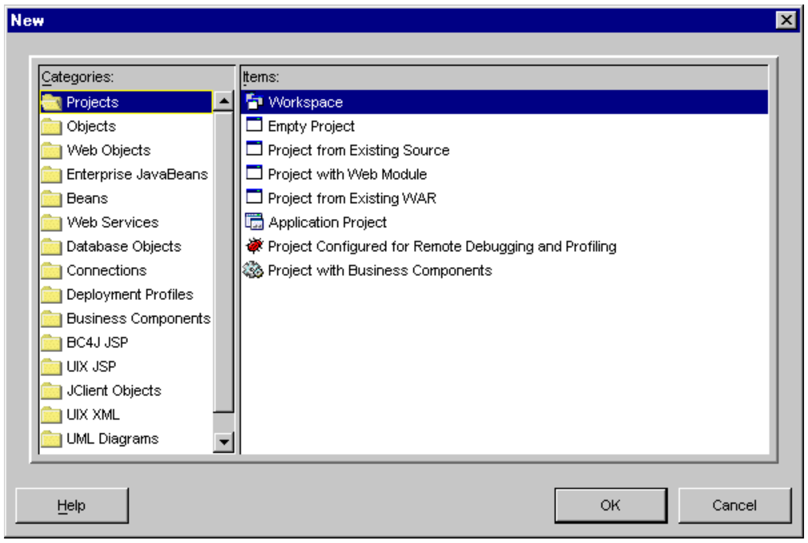
7.2.8.1 Running Sample 1 Using Oracle9*i* JDeveloper

This section discusses how to configure the Oracle9*i* JDeveloper to run the Sample 1 application that is bundled with the Mobile Development Kit for Web-to-Go. For detailed information and full documentation on how to use Oracle9*i* JDeveloper, consult the online help in Oracle9*i* JDeveloper and Oracle9*i* JDeveloper's documentation.

7.2.8.1.1 Creating a Debug Project To create a new debug project in Oracle9*i* JDeveloper, perform the following steps.

1. Start JDeveloper.
2. To create a new project in JDeveloper, click **File**, then click **New** (assuming you have defined a workspace in JDeveloper).
3. From the **Directories** menu in the left panel, select **Projects**, as displayed in Figure 7–4, then select **Empty Project**.

Figure 7–4 Creating a New Project

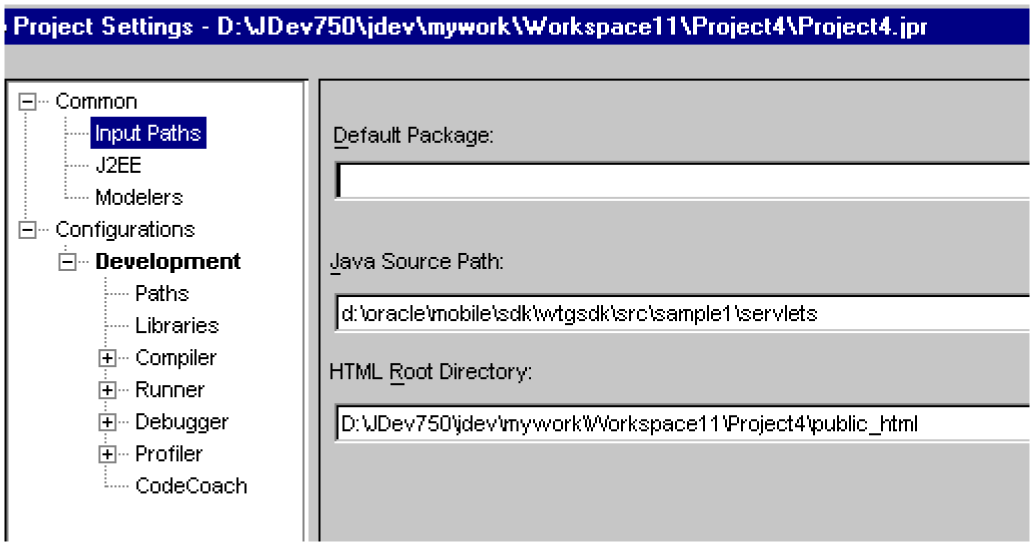


4. Set the Project Settings for your new project. Right click on **Project** to retrieve Project Settings. In the Project Settings dialog, expand Common in the left panel and select Input Paths. In the right panel, enter the following information in the Java Source Path field, as displayed in Figure 7–5.

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

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

Figure 7–5 Project Settings - Input Paths


Project Settings - D:\JDev750\jdev\mywork\Workspace11\Project4\Project4.jpr

Default Package:

Java Source Path: d:\oracle\mobile\sdk\wtgsdk\src\sample1\servlets

HTML Root Directory: D:\JDev750\jdev\mywork\Workspace11\Project4\public_html

5. Expand **Configurations** and then **Development** in the left panel. Select **Paths**, which appears below Development in the left panel. In the **Output Directory** field, in the right panel, enter the following information.

   `<`*`ORACLE_HOME`*`>\mobile\sdk\wtgsdk\root\sample1\servlets`

7.2.8.1.2 Creating a Library Oracle9*i* JDeveloper makes it easier to manage sets of `.jar` files by using libraries instead of `CLASSPATH` settings.

Files for the WTGSDK Library

Create a WTGSDK library with the following `.jar` files and add this library to your project.

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

`<`*`OLITE_HOME`*`>\bin\olite40.jar`

`<`*`ORACLE_HOME`*`>\mobile\sdk\bin\webtogo.jar`

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

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

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

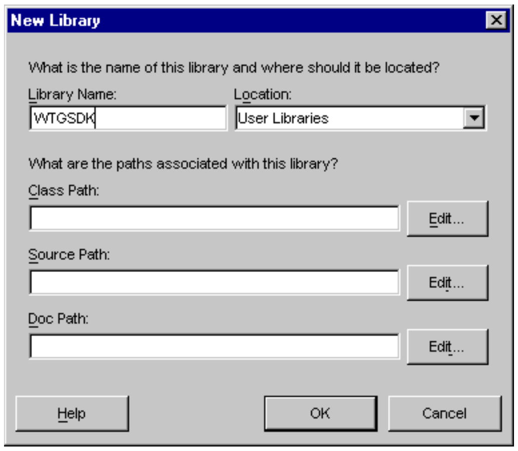
`<`*`ORACLE_HOME`*`>\mobile\classes\wtgpack.jar`

Creating a WTGSDK Library

Perform the following steps to create a WTGSDK library.

1. Select **Libraries** in the left panel, then click **New** in the right panel.
2. The **New Library** dialog appears, as illustrated in Figure 7–6. In the **Library Name** field, enter WTGSDK.

Figure 7–6 The New Library Dialog


New Library

What is the name of this library and where should it be located?

Library Name: WTGSDK

Location: User Libraries

What are the paths associated with this library?

Class Path:

Source Path:

Doc Path:

3. Click **Edit...** next to the Class Path field. The **File** dialog appears.
4. From the appropriate directory, select the six `.jar` files that are listed above.
5. To add the files, click **OK**.

7.2.8.1.3 Adding Files to the Project To add the `Sample1` files to your project, perform the following steps.

1. Click the **green plus-sign** in the Oracle9*i* JDeveloper System-Navigator to add the Java sources to the project. The **File** dialog appears.
2. Select the Java source file `Helloworld.java` in the directory <*ORACLE_HOME*>`\mobile\sdk\wtgsdk\src\sample1\servlets`, and click **Open**.
3. Also, add the file `RunWebServer.java`, which is located in the directory <*ORACLE_HOME*>`\mobile\sdk\wtgsdk\src`, to the project.
4. A dialog appears prompting you to update the project source path. Click **No**.

7.2.8.1.4 Running and Debugging Set one or more breakpoints in your code by right-clicking at the statement where you want to break. Select **Toggle breakpoint**. The background of the statement becomes red, indicating the breakpoint.

1. Select the file `RunWebServer.java` in the **System-Navigator** window.
2. Choose **Debug** by right clicking on the file that you selected to start the Mobile Server inside the debugger.

The Mobile Server is now ready for use. You can access it through your Web browser, by accessing the following URL.

```
http://<machine_name>
```

Where `<machine_name>` is the host name of the computer on which you are running Oracle9*i* JDeveloper.

7.2.8.1.5 Troubleshooting This section describes troubleshooting options that you can implement.

Improving Performance

When you run the Mobile Server inside the Java debugger and access it using a Web browser, performance may decrease. To improve performance, perform the following tasks.

1. Run the Web browser on a different machine.
2. Using the **Task Manager**, set the priority of the Web browser process to `LOW` after you start the Web browser.

7.2.9 Customizing the Workspace Application

The Mobile Development Kit for Web-to-Go includes a set of APIs that contain a basic Web-to-Go workspace application. Developers can use these APIs to replace the standard Web-to-Go workspace application—on the Mobile client only—with a customized version. 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 *Web-to-Go API Specification*, which is located in the following directory.

<*ORACLE_HOME*>`\mobile\doc\javadoc\wtg`

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 7.2.9.1, "Web-to-Go Parameters".
4. Load the `webtogo.odb` file, which is created by the Mobile Client for Web-to-Go, the `webtogo.ora` file for the Mobile Client for Web-to-Go, and the Web-to-Go workspace into the Mobile Server Repository. For more information, refer to the file `crserver.bat`, which is included in the sample Web-to-Go workspace application.
5. Instruct the Mobile Server to use the new Web-to-Go workspace application by modifying the `webtogo.ora` file on the server. For the correct parameter settings in the `webtogo.ora`, refer the section Section 7.2.9.1, "Web-to-Go Parameters".

7.2.9.1 Web-to-Go Parameters

To instruct Web-to-Go to use a customized Web-to-Go workspace application, you must set the following parameters in the `[WEBTOGO]` section of the `webtogo.ora` file.

Table 7–4 describes `webtogo.ora` parameter settings.

Table 7–4 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.

7.2.9.2 Sample Workspace

The Mobile Development Kit for Web-to-Go includes a sample Web-to-Go workspace application that illustrates how to use the Web-to-Go workspace API. Developers can use this sample application as a starting point when developing their Web-to-Go workspace applications. The sample Web-to-Go workspace application is written using JavaServer Pages (JSP) and `.html` files. The JSP files are located in the `myworkspace/out` directory in the Mobile Development Kit for Web-to-Go. These files are compiled into class files that are copied into `myworkspace/out` directory. This directory also contains all `.html` files and image files that are used by the sample Web-to-Go workspace application.

The Mobile Development Kit for Web-to-Go includes the following scripts that compile the JSP files, create the Oracle Database Lite named `webtogo` for the Mobile Client for Web-to-Go, and load all necessary files into the Mobile Server Repository.

Table 7–5 describes scripts available for JSP compilation.

Table 7–5 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.

7.2.10 Using the Mobile Server Admin API

The Mobile Server Admin API enables an administrator to manage the application resources programmatically. Using the Mobile Server Admin API set, administrators can potentially create their own customized Mobile Manager application to perform the following functions.

- Creating and modifying users and user groups
- Including users and excluding users from group level access to applications
- Assigning snapshot variables to the user

- Suspending and resuming applications
- Publishing a pre-packaged Web-to-Go application
- Customizing an application's underlying database connections

For more information on using the API to build the Mobile Manager, see the Web-to-Go API Specification, which is located in the following directory.

`<ORACLE_HOME>\mobile\doc\javadoc\wtg`

Note: Administrators cannot use the open API set to change the basic properties of an application, such as snapshot definitions or servlets. This can only be done through the Packaging Wizard. For more information, see the *Oracle Database Lite Administration and Deployment Guide*.

8 Native Application Development

This document discusses Mobile application development for native platforms. The discussion covers the following topics:

- Section 8.1, "Supported Platforms for Native Applications"
- Section 8.2, "Supported APIs for Oracle Database Lite"
- Section 8.3, "Data Source Name"

8.1 Supported Platforms for Native Applications

For developing native applications on the Oracle Database Lite platform, the following operating system platforms are supported:

- Microsoft Windows NT/2000/XP
- Windows CE

  The following Windows CE chipsets are supported:

  - .NET 4.2 (ARMv4, Emulator)
  - Pocket PC 2003 (ARM, xScale, Emulator)
  - Pocket PC (ARM, Emulator)
- Palm OS
- Linux Redhat 3.0

8.2 Supported APIs for Oracle Database Lite

The following lists the supported APIs for Oracle Database Lite:

Table 8–1 Supported Native APIs

Native API	Description
COM Interface	Can use ODBC to access database. Use Oracle-specific APIs for programmatic synchronization. See Section 4.1, "Synchronization API For the COM Interface" for more information.
C, C++, C#	Can use ODBC to access database. Use Oracle-specific APIs for programmatic synchronization. See Section 4.2, "Synchronization APIs For C or C++ Applications" for more information.

In addition, you can use the following APIs for accessing database.

Table 8–2 Supported APIs

Native API	Description
JDBC	Use JDBC to access the database. See Oracle Database JDBC manuals and Chapter 10, "JDBC Programming" for instructions on how to use this API.
ODBC	Use ODBC to access the database. See Microsoft ODBC manuals for instructions on how to use this API.
.NET environment	Use the ADO.NET API. You can use Oracle-specific APIs for connecting to the database, programmatic synchronization, and other functions. See Section 14.1, "Discussion of the Classes That Support the ADO.NET Provider" for more information.
SODA	See Chapter 12, "Using Simple Object Data Access (SODA) for PalmOS and PocketPC Platforms" for more information.
Visual Basic	Use ODBC to access database.
.NET environment	Use the ADO.NET API. You can use Oracle-specific APIs for connecting to the database, programmatic synchronization, and other functions. See Section 14.1, "Discussion of the Classes That Support the ADO.NET Provider" for more information.
Java	There are several specifications for Java applications. See Chapter 9, "Java Application Development" for the Java application support.

8.3 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 and Palm, the data source name is simply the database name; that is, `<dbname>`.

It is helpful to create a data source name to contain all of the properties of your connection to the database.

9 Java Application Development

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

- Section 9.1, "Java Support for Applications"
- Section 9.2, "Oracle Database Lite Java Development Environment"
- Section 9.3, "Java Development Tools"

9.1 Java Support for Applications

Table 9–1 lists the Java support provided for each platform in Oracle Database Lite.

Table 9–1 Java Support

Category	Windows 32 Web	Windows 32 Native	Windows CE	Linux (1)	For More Information...
JDBC	Yes Oracle Database Lite offer three JDBC drivers. Refer to Section 9.1.1, "JDBC Drivers".	Yes	Yes	Yes On Linux, only JDBC and ODBC access is supported.	Chapter 10, "JDBC Programming"
Java Stored Procedures /Triggers	Yes Java Stored Procedures/Triggers are not supported in the Web-to-Go application model. However Java Stored Procedures can be replicated using the Consolidator Manager API.	Yes	N/A	Yes	Chapter 11, "Java Stored Procedures and Triggers"
Java Server Pages	1.1	N/A	N/A	N/A	Section 7.2.3, "Developing JavaServer Pages"

Table 9–1 (Cont.) Java Support

Category	Windows 32 Web	Windows 32 Native	Windows CE	Linux (1)	For More Information...
Java Servlet	2.2	N/A	N/A	N/A	Section 7.2.4, "Developing Java Servlets for Web-to-Go" and Section 7.2.7, "Developing Applet Servlet Communication"
BC4J	Yes Latest version of Oracle JDeveloper 10*g*.	N/A	N/A	N/A	
Struts	Yes	N/A	N/A	N/A	

In addition, for programmatically synchronizing from a Java application, see Chapter 3, "Synchronization".

9.1.1 JDBC Drivers

The Oracle Database Lite JDBC driver is JDBC 1.2 compliant. Oracle Lite provides a limited number of extensions specified by JDBC 2.0. These extensions are compatible with the Oracle Database JDBC implementation.

Oracle Database Lite offers the following JDBC drivers:

- Type 2 driver: There are two types of type 2 driver: one provides an embedded, direct connection. This driver allows Java applications to communicate directly with the Oracle Lite database. The other type 2 driver provides a remote connection and requires Multi-User service support.
- Type 4 driver : 100% Java implementation. Requires the multi-user database version.

9.2 Oracle Database Lite Java Development Environment

To develop Java applications, you need to set up your development environment to create Oracle Database Lite applications, as follows:

- You must have the Sun Microsystems Java Development Kit (JDK), version 1.4.2 (or higher).
- To enable Oracle Database Lite to work with the JDK, set your `PATH` and `CLASSPATH` environment variables after you install Oracle Database Lite. See Section 9.2.1, "Setting Variables for JDK 1.4.2" for full details.

Note: If your environment includes a `CLASSPATH` user variable before you install Oracle Database Lite and the user variable does not include the `CLASSPATH` system variable (is not specified as `CLASSPATH=...;%CLASSPATH%`), then you must modify the `CLASSPATH` user variable to include the `olite40.jar` file in the *`OLITE_HOME`*`\bin` directory.

Note: All command prompt windows must be closed and reopened to reflect changes made to your `CLASSPATH`.

9.2.1 Setting Variables for JDK 1.4.2

The directory with the JDK 1.4.2 Java compiler (`javac.exe`) should be in the `PATH` variable before any other directories that contain other Java compilers.

Add the directory that contains the Classic Java Virtual Machine (JVM) shared library, `jvm.dll`, to the PATH. `jvm.dll` should be in your `JDK_Home\jre\bin\classic` directory.

For example,

```
set PATH=C:\JDK_Home\bin;c:\JDK_Home\jre\bin\classic
set CLASSPATH=c:\JDK_Home\jrc\lib\rt.jar;c:\OLITE_HOME\bin\olite40.jar
```

As an alternative to using the Classic JVM, you can use the HotSpot JVM. HotSpot is a JDK add on module provided by Sun Microsystems. HotSpot is available from the Sun Microsystems Web site.

After installing HotSpot, set your `PATH` as given below.

```
set PATH=c:\jdk\bin;c:\jdk\jre\bin\hotspot;%PATH%
```

In the example above, your installation of the JDK and HotSpot is on Drive `C:\`. Verify the location of your installation before amending your `PATH` statement. To test whether your system is set up correctly, run the Java examples in the <*ORACLE_HOME*>`\Mobile\Sdk\Samples\JDBC` directory.

9.3 Java Development Tools

To write and debug Java programs, you can use any Java development tool. However, you must ensure that you set the `CLASSPATH` and `PATH` correctly.

10 JDBC Programming

This chapter discusses the Oracle Database Lite support for JDBC programming. It includes the following topics:

10.1 JDBC Compliance

Oracle Database Lite provides a native JDBC driver that allows Java applications to communicate directly with the Oracle Database Lite object relational database engine. The Oracle Database Lite implementation of JDBC complies with JDBC 1.2. In addition, Oracle Database Lite provides certain extensions specified by JDBC 2.0, which are compatible with the Oracle database JDBC implementation. For a complete JDBC reference, see the Sun Microsystems Web site.

10.2 JDBC Environment Setup

For your Java applications using the client/server model, include the `olite40.jar`, which is located in *`OLITE_HOME`*`/bin`, in the system `CLASSPATH` on the server machine and in the user `CLASSPATH` on the client machine.

When using the Oracle Lite JDBC driver in your OC4J application, use the default classloader instead of a per-application classloader, which many J2EE containers use. Ensure that the `olite40.jar` is in the OC4J `CLASSPATH` when OC4J initiates and not in the `/lib` subdirectory of your application WAR file.

Note: For more information on how to start the Multiuser Oracle Database Lite Database Service, see Section 2.5, "Oracle Database Lite Multi-User Service".

10.3 JDBC Drivers to Use When Connecting to Oracle Database Lite

Note: JDK 1.4.2 or higher is required to connect to Oracle Database Lite.

Oracle Database Lite supports Type 2 and Type 4 drivers.

- The Type 2 driver uses native code on the client side through which it interfaces with the Oracle Database Lite ODBC driver.
- The Type 4 JDBC driver is a pure Java driver and uses the Oracle Database Lite network protocol to communicate with the Oracle Database Lite service. Before using this driver, ensure that you start Oracle Database Lite. Any Java applet can use the Type 4 JDBC driver.

The supported Type 2 and Type 4 drivers are described in the following sections:

- Section 10.3.1, "Type 2 Driver"
- Section 10.3.2, "Type4 (Pure Java) Driver Connection URL Syntax"

10.3.1 Type 2 Driver

For most applications, use the type 2 driver for connecting to the database. You can use the type 2 driver to connect either to the local Oracle Lite database or to the server where a Multi-User Service is managing the Oracle Lite databases.

- To connect to the local Oracle Lite database, use the following URL syntax:

  ```
  jdbc:polite[:uid / pwd]:localDSN[;key=value]*
  ```

  where the localDSN is the DSN name for the local Oracle Lite database (the ODB file on the local machine) and the optional key=value pairs are listed in Table 10–1.

  The following example retrieves a connection to the local Oracle Lite database, where the DSN name is `polite`:

  ```
  DriverManager.getConnection("jdbc:polite:polite","system","admin");
  ```

- To access the Oracle Lite database on a remote host where a Multi-User Service or Branch Office is located, use the following URL syntax:

  ```
  jdbc:polite[:uid / pwd]@[host]:[port]:serverDSN [;key=value]*
  ```

  where the host, port, and serverDSN identify the host, port and DSN of the remote host where the Oracle Lite database (the ODB file on the local machine) and the multi-user service is located. The optional key=value pairs are listed in Table 10–1.

  For more information on how to install and start the Multiuser Oracle Database Lite Service, refer to Section 2.5, "Oracle Database Lite Multi-User Service".

You can provide additional configuration information in the JDBC driver URL within key-value pairs, as specified in Table 10–1, each of which are separated by a semi-colon. The information specified within the key-value pairs always overrides the information that is specified in the URL.

Table 10–1 Key/Value Pairs for JDBC Connect URL

Argument	Description
`jdbc`	Identifies the protocol as JDBC.

Table 10–1 (Cont.) Key/Value Pairs for JDBC Connect URL

Argument	Description
`polite`	Identifies the subprotocol as `polite`.
uid / pwd	The optional user ID and password for Oracle Database Lite. If specified, this overrides the specification of a username and password defined in the UID and PWD arguments. If the database is encrypted, you must include the password in the key-value pair.
host	The name of the machine that hosts the Multi-User Service or Branch Office and on which the Oracle Database Lite service `olsv2040.exe` runs. This host name is optional. If omitted, it defaults to the local machine on which the JDBC application runs.
port	The port number at which the Multi-User or Branch Office service listens. The port number is optional and if omitted defaults to port "1531".
dsn	Identifies the data source name (DSN) entry in the `odbc.ini` file. This entry contains all the necessary information to complete the connection to the server. **Note**: For a JDBC program, you need not create a DSN if you have supplied all the necessary values for the data directory and database through key=value pairs. On the windows platform, you can use the ODBC administrator to create a DSN. For more information, see Section 2.5.1.6, "Creating DSNs".
DataDirectory=	Directory in which the `.odb` file resides.
Database=	Name of database as given during its creation.
IsolationLevel=	Transaction isolation level: READ COMMITTED, REPEATABLE READ, SERIALIZABLE or SINGLE USER. For more information on isolation levels, see Section 15.2, "What Are the Transaction Isolation Levels?".
Autocommit=	Commit behavior, either `ON` or `OFF`.
CursorType=	Cursor behavior: DYNAMIC, FORWARD ONLY, KEYSET DRIVEN or STATIC. For more information on cursor types, see Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types".
UID=	User name
PWD=	Password

Example of Using JDBC Type 2 Connection for Local Oracle Lite Database

```
String ConnectMe=("jdbc:polite:SCOTT/tiger:polite;
DataDirectory=<ORACLE_HOME>;Database=polite;IsolationLevel=SINGLE USER;
Autocommit=ON;CursorType=DYNAMIC")

try
   {Class.forName("oracle.lite.poljdbc.POLJDBCDriver")
   Connection conn = DriverManager.getConnection(ConnectMe)
   }
catch (SQLException e)
{
...
}
```

Example of Using Type 2 in Multi-User Service Situation

An example of this type of connection is given below.

```
try {
Connection conn = DriverManager.getConnection(
   "jdbc:polite@yourhostname
        ;DataDirectory=<ORACLE_HOME>
        ;Database=polite
        ;IsolationLevel=SINGLE USER
        ;Autocommit=ON
        ;CursorType=DYNAMIC", "Scott", "tiger")
}
catch (SQLException e)
{
}
```

10.3.2 Type4 (Pure Java) Driver Connection URL Syntax

Use the JDBC Type 4 driver for any pure Java application that uses the Multi-User or Branch office services. The URL syntax for the type 4 driver as follows:

```
jdbc:polite4[:uid/pwd]@[host]:[port]:serverDSN[;key=value]*
```

The parameter `polite4` indicates that the JDBC type 4 driver is being used. For the rest of the parameters, see the definitions of those parameters for the type 2 driver, as described in Table 10–1.

Note: The URL works with the Oracle Database Lite service only.

10.4 DataSource Connection

In JDBC 2.0, the `DataSource` object is an alternative to the `DriverManager` facility. The `DataSource` object is the preferred method for retrieving a connection and is typically registered with a naming service based on the JNDI API. A driver that is accessed through a `DataSource` object does not register itself with the `DriverManager`.

The Oracle Database Lite JDBC driver contains the basic `DataSource` implementation to produce a standard `Connection` object. The retrieved connection is identical to a connection obtained through the `DriverManager`.

Oracle Database Lite implements `javax.sql.DataSource` interface with the `POLJDBCDataSource` class in the `oracle.lite.poljdbc` package.

As with any class that implements the `DataSource` interface, the `POLJDBCDataSource` object defines properties for connecting to a specific database. In addition to the standard DataSource properties, the `POLJDBCDataSource` class has one additional property, which is a `String` to define the URL of the database connection string, as described in Section 10.3.1, "Type 2 Driver" and Section 10.3.2, "Type4 (Pure Java) Driver Connection URL Syntax".

- The URL can include username and password, which then overrides any previous individual property settings. The following URL specifies a username and password of system/manager:

  ```
  jdbc:polite4:system/manager@::polite
  ```

- You must have a username and password defined either in the URL or in the getConnection method. If defined in both places, then the username and password in the getConnection takes precedence over the URL definitions.

See the JDBC example—JDBCEXJSR169—on the CD for how to use the `POLJDBCDataSource` object.

10.5 Java Datatypes and JDBC Extensions

The Oracle Database Lite JDBC driver supports JDBC 1.2 and provides extensions that support certain features defined in JDBC 2.0. The extensions include support for BLOB (large binary object) and CLOB (large character object) datatypes and scrollable result sets. The Oracle Database Lite JDBC extensions are compatible with the Oracle database JDBC implementation. However, Oracle Database Lite does not support the following Oracle database JDBC datatype extensions: `Array`, `Struct`, or `REF`.

The following sections list and describe the Oracle Database Lite datatypes and data access extensions. For details regarding function syntax and call parameters, see the Sun Microsystems Java 2 specification at the Sun Javasoft website.

- Section 10.5.1, "Java Datatypes"
- Section 10.5.2, "Datatype Extensions"
- Section 10.5.3, "Data Access Extensions"

10.5.1 Java Datatypes

Oracle Database Lite performs type conversions between Java and Oracle datatypes as indicated by the following table. Table 10–2 lists the Java datatypes and the corresponding SQL datatypes that result from the type conversion.

Table 10–2 Datatype Conversions

Java Datatype	SQL Datatype
`byte[], byte[][], Byte[]`	`BINARY, RAW, VARBINARY, BLOB`
`boolean, Boolean`	`BIT`
`String, String[]`	`CHAR, VARCHAR, VARCHAR2, CLOB`
`short, short[], short[][], Short, Short[]`	`SMALLINT`
`int,int[], int[][], Integer, Integer[]`	`INT`
`float, float[], float[][], Float, Float[]`	`REAL`
`double, double[], double[][], Double, Double[]`	`DOUBLE, NUMBER` (without precision)
`BigDecimal, BigDecimal[]`	`NUMBER(n)`
`java.sql.Date, java.sql.Date[]`	`DATE`
`java.sql.Time, java.sql.Time[]`	`TIME`
`java.sql.Timestamp, java.sql.Timestamp[]`	`TIMESTAMP`
`java.sql.Connection`	Default JDBC connection to database

10.5.2 Datatype Extensions

BLOBs and CLOBs store data items that are too large to store directly in a database table. Rather than storing the data, the database table stores a locator that points to the location of the actual data. BLOBs contain a large amount of unstructured binary data items and CLOBs contain a large amount of fixed-width character data items (characters that require a fixed number of bytes per character).

You can select a BLOB or CLOB locator from the database using a standard SELECT statement. When you select a BLOB or CLOB locator using SELECT, you acquire only the locator for the large object, not the data itself. Once you have the locator, however, you can read data from or write data to the large object using access functions.

Table 10–3 lists the methods included in the Oracle Database Lite BLOB class and their descriptions:

Table 10–3 Methods in the Oracle Database Lite BLOB Class

Function	Description
`length`	Returns the length of a BLOB in bytes.
`getBinaryOutputStream`	Returns BLOB data.
`getBinaryStream`	Returns a BLOB instance as a stream of bytes.
getBytes	Reads BLOB data, starting at a specified point, into a buffer.
`getConnection`	Returns the current connection.
`isConvertibleTo`	Determines if a BLOB can be converted to a particular class.
`putBytes`	Writes bytes to a specified point in the BLOB data.
`makeJdbcArray`	Returns the JDBC array representation of a BLOB.
`toJdbc`	Converts a BLOB to a JDBC class.
`trim`	Trims to length.

Table 10–4 lists the methods included in the Oracle Database Lite CLOB class and their descriptions.

Table 10–4 Methods in the Oracle Database Lite CLOB Class

Function	Description
`length`	Returns the length of a CLOB in bytes.
`getSubString`	Retrieves a substring from a specified point in the CLOB data.
`getCharacterStream`	Returns CLOB data as a stream of Unicode characters.
`getAsciiStream`	Returns a CLOB instance as an ASCII stream.
`getChars`	Retrieves characters from a specified point in the CLOB data into a character array.
`getCharacterOutputStream`	Writes CLOB data from a Unicode stream.
`getAsciiOutputStream`	Writes CLOB data from an ASCII stream.
`getConnection`	Returns the current connection.
`putChars`	Writes characters from a character array to a specified point in the CLOB data.
`putString`	Writes a string to a specified point in the CLOB data.
`toJdbc`	Converts a CLOB to a JDBC class.
`isConvertibleTo`	Determines if a CLOB can be converted to a particular class.
`makeJdbcArray`	Returns a JDBC array representation of a CLOB.
`trim`	Trims to length.

10.5.3 Data Access Extensions

Oracle Database Lite provides access functions to set and return values of the CLOB and BLOB datatypes. In addition, stream classes provide functions that enable stream-format access to large objects.

The large object access functions are located in the `OraclePreparedStatement`, the `OracleCallableStatement`, and the `OracleResultSet` class.

Table 10–5 lists the data access functions included in the `OracleResultSet` class.

Table 10–5 Data Access Functions in the OracleResultSet Class

Function	Description
`getBLOB`	Returns a locator to BLOB data.
`getCLOB`	Returns a locator to CLOB data.

The stream format access classes are `POLLobInputStream`, `POLLobOutputStream`, `POLClobReader`, and `POLClobWriter`.

The `POLLobInputStream` class includes the following data access function.

Function	Description
`read`	Reads from a large object into an array.

The `POLLobOutputStream` class includes this data access function.

Function	Description
`write`	Writes from an output stream into a large object.

The `POLClobReader` class extends the class `java.io.reader`. It includes these data access functions.

Function	Description
`read`	Reads characters from a CLOB into a portion of an array.
`ready`	Indicates whether a stream is ready to read.
`close`	Closes a stream.
`markSupported`	Indicates whether the stream supports the `mark` operation.
`mark`	Marks the current position in the stream. Subsequent calls to the `reset` function reposition the stream to the marked location.
`reset`	Resets the current position in the stream to the marked location. If the stream has not been marked, this function attempts to reset the stream in a way appropriate to the particular stream, such as by repositioning it at its starting point.
`skip`	Skips characters in the stream.

The `POLClobWriter` class extends the class `java.io.writer`. It includes these data access functions:

Function	Description
`write`	Writes an array of characters to the output stream.
`flush`	Writes any characters in a buffer to their intended destination.
`close`	Flushes and closes the stream.

10.5.3.1 Reading from a BLOB Sample Program

The following sample uses the `getBinaryStream` method to read BLOB data into a byte stream. It then reads the byte stream into a byte array, and returns the number of bytes read.

```
// Read BLOB data from BLOB locator.
InputStream byte_stream = my_blob.getBinaryStream();
byte [] byte_array = new byte [10];
int bytes_read = byte_stream.read(byte_array);
...
```

10.5.3.2 Writing to a CLOB Sample Program

The following sample reads data into a character array, then uses the `getCharacterOutputStream` method to write the array of characters to a CLOB.

```
java.io.Writer writer;
char[] data = {'0','1','2','3','4','5','6','7','8','9'};

// write the array of character data to a CLOB
writer = ((CLOB)my_clob).getCharacterOutputStream();
writer.write(data);
writer.flush();
writer.close();
...
```

10.6 Limitations

If data truncation occurs during a write, a SQL data truncation exception is thrown. A SQL data truncation warning results if data truncation occurs during a read.

The Oracle Database Lite JDBC classes and the JDBC 2.0 classes use the same name for certain datatypes (for example, `oracle.sql.Blob` and `java.sql.Blob`). If your program imports both `oracle.sql.*` and `java.sql.*`, attempts to access the overlapping classes without fully qualifying their names may result in compiler errors. To avoid this problem, use one of the following steps:

1. Use fully qualified names for BLOB, CLOB, and data classes.
2. Import the class explicitly (for example, import `oracle.sql.Blob`).

Class files always contain fully qualified class names, so the overlapping datatype names do not cause conflicts at runtime.

10.7 New JDBC 2.0 Features

This section describes JDBC 2.0 methods or interfaces that are supported by the Oracle Database Lite JDBC driver. Topics include:

- Section 10.7.1, "Interface Connection"
- Section 10.7.2, "Interface Statement"

- Section 10.7.3, "Interface ResultSet"
- Section 10.7.4, "Interface Database MetaData"
- Section 10.7.5, "Interface ResultMetaData"
- Section 10.7.6, "Interface PreparedStatement"

10.7.1 Interface Connection

This section describes the JDBC 2.0 Interface methods that are implemented by the Oracle Database Lite JDBC driver.

10.7.1.1 Methods

Statement

`createStatement(int resultSetType, int resultSetConcurrency)`

Creates a statement object that generates ResultSet objects with the given type and concurrency.

Map

`getTypeMap()`

Gets the TypeMap object associated with this connection.

CallableStatement

```
prepareCall(String sql, int resultSetType, int
resultSetConcurrency)
```

Creates a CallableStatement object that generates ResultSet objects with the given type and concurrency.

PreparedStatement

```
prepareStatement(String sql, int resultSetType, int
resultSetConcurrency)
```

Creates a PreparedStatement object that generates ResultSet objects with the given type and concurrency.

void

`setTypeMap(Map map)`

Installs the given type map as the type map for this connection.

10.7.2 Interface Statement

This section describes the JDBC 2.0 Interface Statement methods that are implemented by the Oracle Database Lite JDBC driver.

Connection

`getConnection()`

Returns the Connection object that produced this Statement object.

int

`getFetchDirection()`

Retrieves the direction for fetching rows from database tables that is the default for result sets generated from this Statement object. Only FETCH_FORWARD is supported for now.

int

`getFetchSize()`

Retrieves the number of result set rows that is the default fetch size for result sets generated from this Statement object. Only fetch size = 1 is supported for now.

int

`getResultSetConcurrency()`

Retrieves the result set concurrency. Only CONCUR_READ_ONLY is supported for now.

int

`getResultSetType()`

Determine the result set type. Only TYPE_FORWARD_ONLY and TYPE_SCROLL_INSENSITIVE are supported for now.

void

`setFetchDirection(int direction)`

Gives the driver a hint as to the direction in which the rows in a result set will be processed.

void

`setFetchSize(int rows)`

Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed.

10.7.3 Interface ResultSet

This section describes the JDBC 2.0 Interface ResultSet methods that are implemented by the Oracle Database Lite JDBC driver.

10.7.3.1 Fields

The following fields can be used to implement the Interface ResultSet feature.

static int

`CONCUR_READ_ONLY`

The concurrency mode for a ResultSet object that may NOT be updated.

static int

`CONCUR_UPDATABLE`

The concurrency mode for a ResultSet object that may be updated. Not supported for now.

static int

`FETCH_FORWARD`

The rows in a result set will be processed in a forward direction; first-to-last.

static int

`FETCH_REVERSE`

The rows in a result set will be processed in a reverse direction; last-to-first. Not supported for now.

static int

`FETCH_UNKNOWN`

The order in which rows in a result set will be processed is unknown.

static int

`TYPE_FORWARD_ONLY`

The type for a ResultSet object whose cursor may move only forward.

static int

`TYPE_SCROLL_INSENSITIVE`

The type for a ResultSet object that is scrollable but generally not sensitive to changes made by others.

static int

`TYPE_SCROLL_SENSITIVE`

The type for a ResultSet object that is scrollable and generally sensitive to changes made by others. Not supported for now.

10.7.3.2 Methods

This section describes the JDBC 2.0 ResultSet method implemented by the Oracle Database Lite JDBC driver.

boolean

`absolute(int row)`

Moves the cursor to the given row number in the result set.

void

`afterLast()`

Moves the cursor to the end of the result set, just after the last row.

void

`beforeFirst()`

Moves the cursor to the front of the result set, just before the first row.

boolean

`first()`

Moves the cursor to the first row in the result set.

Array
`getArray(String colName)`

Gets an SQL ARRAY value in the current row of this ResultSet object.

BigDecimal
`getBigDecimal(int columnIndex)`

Gets the value of a column in the current row as a java.math.BigDecimal object with full precision.

BigDecimal
`getBigDecimal(String columnName)`

Gets the value of a column in the current row as a java.math.BigDecimal object with full precision.

int
`getConcurrency()`

Returns the concurrency mode of this result set.

Date
`getDate(int columnIndex, Calendar cal)`

Gets the value of a column in the current row as a java.sql.Date object.

int
`getFetchDirection()`

Returns the fetch direction for this result set.

int
`getFetchSize()`

Returns the fetch size for this result set.

int
`getRow()`

Retrieves the current row number.

Statement
`getStatement()`

Returns the Statement that produced this ResultSet object.

int
`getType()`

Returns the type of this result set.

boolean
`isAfterLast()`

boolean

`isBeforeFirst()`

boolean

`isFirst()`

boolean

`isLast()`

boolean

`last()`

Moves the cursor to the last row in the result set.

boolean

`previous()`

Moves the cursor to the previous row in the result set.

void

`refreshRow()`

Refreshes the current row with its most recent value in the database. Currently does nothing.

boolean

`relative(int rows)`

Moves the cursor a relative number of rows, either positive or negative.

10.7.3.3 Methods that Return False

The following three methods always return false because this release does not support deletes, inserts, or updates.

boolean

`rowDeleted()`

Indicates whether a row has been deleted.

boolean

`rowInserted()`

Indicates whether the current row has had an insertion.

boolean

`rowUpdated()`

Indicates whether the current row has been updated.

void

`setFetchDirection(int direction)`

Gives a hint as to the direction in which the rows in this result set will be processed.

void

`setFetchSize(int rows)`

Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed for this result set.

10.7.4 Interface Database MetaData

This section describes the JDBC 2.0 Interface Database MetaData methods that are implemented by the Oracle Database Lite JDBC driver.

10.7.4.1 Methods

The following methods can be used to implement the Interface Database MetaData feature.

Connection

`getConnection()`

Retrieves the connection that produced this metadata object.

boolean

`supportsResultSetConcurrecny(int type, int concurrency)`

Supports the concurrency type in combination with the given result set type.

boolean

`supportsResultSetType(int Type)`

Supports the given result set type.

10.7.4.2 Methods that Return False

The following methods return false, because this release does not support deletes or updates.

boolean

`deletesAreDetected(int Type)`

Indicates whether or not a visible row delete can be detected by calling ResultSet.rowDeleted().

boolean

`insertsAreDetected(int Type)`

Indicates whether or not a visible row insert can be detected by calling ResultSet.rowInserted().

boolean

`othersDeletesAreVisible(int Type)`

Indicates whether deletes made by others are visible.

boolean

`othersInsertsAreVisible(int Type)`

Indicates whether inserts made by others are visible.

boolean

`othersUpdatesAreVisible(int Type)`

Indicates whether updates made by others are visible.

boolean

`ownDeletesAreVisible(int Type)`

Indicates whether a result set's own deletes are visible.

boolean

`ownInsertsAreVisible(int Type)`

Indicates whether a result set's own inserts are visible.

boolean

`ownUpdatesAreVisisble(int Type)`

Indicates whether a result set's own updates are visible.

boolean

`updatesAreDetected(int Type)`

Indicates whether or not a visible row update can be detected by calling the method ResultSet.rowUpdated.

10.7.5 Interface ResultMetaData

This section lists methods that can be implemented using the Interface ResultMetaData feature.

10.7.5.1 Methods

The following method can be used to implement the Interface ResultMetaData feature.

String

`getColumnClassName(int column)`

Returns the fully-qualified name of the Java class whose instances are manufactured if the method ResultSet.getObject is called to retrieve a value from the column.

10.7.6 Interface PreparedStatement

This section describes methods that can be implemented using the Interface PreparedStatement feature.

10.7.6.1 Methods

The following methods can be used to implement the Interface PreparedStatement feature.

Result

`SetMetaDatagetMetaData()`

Gets the number, types and properties of a ResultSet's columns.

void

`setDate(int parameter Index, Date x, Calendar cal)`

Sets the designated parameter to a java.sql.Date value, using the given Calendar object.

void

`setTime(int parameterIndex, Time x, Calendar cal)`

Sets the designated parameter to a java.sql.Time value, using the given Calendar object.

void

`setTimestamp(int parameter Index, Timestamp x, Calendar cal)`

Sets the designated parameter to a java.sql.Timestamp value, using the given Calendar object.

10.7.6.1.1 Limitation currently, the option `setQueryTimeOut` is not supported.

11

Java Stored Procedures and Triggers

This chapter describes how to use Java stored procedures and triggers within the Oracle Database Lite relational model. Topics include:

11.1 Java Stored Procedure Features in Oracle Database Lite

Oracle Database Lite supports the Oracle database server development model for stored procedures. In this model (referred to as the "load and publish" development model), instead of attaching classes to tables, you load the Java class into the Oracle Database Lite database using the `loadjava` command-line utility or the SQL statement `CREATE JAVA`. After loading the class into the database, you use a call specification to publish the methods in the class that you want to call from SQL. You use either the `CREATE FUNCTION` or `CREATE PROCEDURE` command to create a call specification. For more information, see Section 11.3.1, "Model 1: Using the Load and Publish Stored Procedure Development Model".

Oracle Database Lite still supports the traditional model of creating stored procedures. In the traditional model, you attach the Java class to a table. The static methods in the class become the table-level stored procedures of the table, and the non-static (instance) methods become the row-level stored procedures.

Oracle Database Lite now includes the `loadjava` utility, which automates the task of loading Java classes into the database. Using `loadjava`, you can load Java class, source, and resource files, individually or in archives. For more information, see Section 11.3.1.1.1, "loadjava".

11.2 Stored Procedures and Triggers Overview

A Java stored procedure is a Java method that is stored in Oracle Database Lite. The procedure can be invoked by applications that access the database. A trigger is a stored procedure that executes, or "fires", when a specific event occurs, such as a row update, insertion, or deletion. An update of a specific column can also fire a trigger.

A trigger can operate at the statement-level or row-level. A statement-level trigger fires once per triggering statement, no matter how many rows are affected. A row-level trigger fires once for every row affected by the triggering statement. Java stored procedures can return a single value, a row, or multiple rows. Triggers, however, cannot return a value.

The first step to creating a stored procedure is to create the class that you want to store in Oracle Database Lite. You can use any Java IDE to write the procedure, or you can simply reuse an existing procedure that meets your needs.

When creating the class, consider the following restrictions on calling Java stored procedures from SQL DML statements:

- When called from an `INSERT`, `UPDATE`, or `DELETE` statement, the method cannot query or modify any database tables modified by that statement.
- When called from a `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement, the method cannot execute SQL transaction control statements, such as `COMMIT` or `ROLLBACK`.

Any SQL statement in a stored procedure that violates a restriction produces an error at run time.

You must provide your class with a unique name for its deployment environment, since only one Java Virtual Machine is loaded for each Oracle Database Lite application. If the application executes methods from multiple databases, then the Java classes from these databases are loaded into the same Java Virtual Machine. By prefixing the Java class name with the database name, you ensure that the Java class names are unique across multiple databases.

If a Java stored procedure takes an argument of type `java.sql.Connection`, then Oracle Database Lite supplies the appropriate argument value from the current transaction or row as the first argument to the method. The application executing the method does not need to provide a value for this parameter. In this case, DMLs executed inside the procedure are executed in the invoker's transaction context.

11.3 Using Stored Procedures

Oracle Database Lite supports several development models for creating stored procedures. In the load and publish model, you load the Java class into Oracle Database Lite, then create a call specification (call spec) for the static methods in the class that you want to call from SQL. This model is also supported by Oracle database, which enables you to utilize skills and resources you have already developed in implementing Oracle database enterprise applications and data.

This model consists of the following steps:

1. Develop a Java class that contains the methods you want to store.
2. Use the `loadjava` utility or the `SQL CREATE JAVA` command to load the class into the Oracle Database Lite.
3. Publish the methods that you want to make accessible to SQL by creating call specs for those methods. By publishing a method, you associate a SQL name to the method. SQL applications use this name to invoke the method.

You do not need to publish every procedure that you store in Oracle Database Lite, only those that should be callable from SQL. Many stored procedures are only called by other stored procedures, and do not need to be published. For more information on using this model for developing stored procedures, see Section 11.3.1, "Model 1: Using the Load and Publish Stored Procedure Development Model". The load and publish model only supports static methods.

In the second model, you attach the class to a table and invoke methods in the class by name. This is the traditional Oracle Database Lite model for developing stored procedures. Using this model, you can store both class-level (static) methods and object-level (non-static) methods.

For this model, follow these steps:

1. Develop a Java class with the methods you want to store.
2. Attach the class to a table using the `SQL ALTER TABLE` command.

After attaching the class, you can invoke methods in the class directly from SQL. You identify the method with the following syntax:

```
table_name.method_name
```

For more information on attaching Java classes to tables, see Section 11.3.2, "Model 2: Using the Attached Stored Procedure Development Model".

Oracle Database Lite provides tools and SQL commands for dropping stored procedures. You should use caution when dropping procedures from the database, since Oracle Database Lite does not keep track of dependencies between classes. You must ensure that the stored procedure you drop is not referenced by other stored procedures. Dropping a class invalidates classes that depend on it directly or indirectly.

11.3.1 Model 1: Using the Load and Publish Stored Procedure Development Model

This section describes how to create stored procedures using the load and publish development model. The first step in creating a stored procedure is to write the class. Make sure that the class compiles and executes without errors. Next, load the class into Oracle Database Lite. Finally, publish the methods that you want to call from SQL. In Oracle Database Lite, you cannot publish a method that is mapped to a `main` method. Oracle database, on the other hand, permits call specs that publish `main` methods.

Note: The load and publish development model only supports Java static methods. To store static and non-static (instance) methods, you must attach the class to database tables, as described in Section 11.3.2, "Model 2: Using the Attached Stored Procedure Development Model".

11.3.1.1 Loading Classes

To load Java classes into the Oracle Database Lite database, you can use either:

- `loadjava`
- the SQL statement `CREATE JAVA`

The `loadjava` command-line utility automates the task of loading Java classes into Oracle Database Lite and Oracle databases. To load Java classes manually, use the SQL statement `CREATE JAVA`.

11.3.1.1.1 loadjava

`loadjava` creates schema objects from files and loads them into the database. Schema objects can be created from Java source files, class files, and resource files. Resource files may be image files, resources, or anything else a procedure may need to access as data. You can pass files to `loadjava` individually, or as ZIP or JAR archive files.

Oracle Database Lite does not keep track of class dependencies. Make sure that you load into the database, or place in the `CLASSPATH`, all supporting classes and resource files required by a stored procedure. To query the classes that are loaded in the database, you can query the `okJavaObj` meta class.

Note: The table name and column names are case sensitive.

Syntax

`loadjava` uses the following syntax:

```
loadjava {-user | -u} username/password[@database]
   [-option_name -option_name ...] filename filename ...
```

Arguments

This section discusses the `loadjava` arguments in detail.

User

The `user` argument specifies a username, password, and database directory in the following format:

```
<user>/<password>[@<database>]
```

For example:

```
scott/tiger@ ORACLE_HOME\Mobile\Sdk\OLDB40\Polite.odb
```

Options

Oracle Database Lite supports the following options that are listed and described in Table 11–1.

Table 11–1 Options

Option	Description
`-force | -f`	Forces files to be loaded, even if a schema object with the same name already exists in the database.
`-verbose | -v`	Directs `loadjava` to display detailed status messages while running.
`-meta | -m`	Creates the meta information in the database but does not load the classes. This is useful when the classes are in a .jar file and are not loaded into the database.

When specifying multiple options, you must separate the options with spaces. For example:

```
-force -verbose
```

Oracle database supports additional options, as described in the "Developing Stored Procedures" chapter in the *Oracle Database Java Developer's Guide*. If used with Oracle Database Lite, the additional options are recognized but not supported. Using them does not result in an error.

To view the options supported by Oracle database, see the `loadjava` help information using the following syntax.

```
loadjava {-help | -h}
```

Filenames

On the command line, you can specify as many class, source, JAR, ZIP, and resource files as you like, in any order. You must separate multiple file names with spaces, not commas. If passed a source file, `loadjava` invokes the Java compiler to compile the file before loading it into the database. If passed a JAR or ZIP file, `loadjava` processes each file in the JAR or ZIP. It does not create a schema object for the JAR or ZIP archive. `loadjava` does not process a JAR or ZIP archive within another JAR or ZIP archive.

The best way to load files is to place them in a JAR or ZIP and then load the archive. Loading archives avoids the complications associated with resource schema object names. If you have a JAR or ZIP that works with the JDK, then you can be sure that loading it with `loadjava` also works, and you can avoid the complications associated with resource schema object naming.

As it loads files into the database, `loadjava` must create a name for the schema objects it creates for the files. The names of schema objects differ slightly from filenames, and different schema objects have different naming conventions. Class files are self-identifying, so `loadjava` can map their filenames to the names of schema objects automatically. Likewise, JAR and ZIP archives include the names of the files they contain.

However, resource files are not self-identifying; `loadjava` derives the names of Java resource schema objects from the literal names you enter on the command-line (or the literal names in a JAR or ZIP archive). Because classes use resource schema objects while executing, it is important that you specify the correct resource file names on the command line.

The best way to load individual resource files is to run `loadjava` from the top of the package tree, specifying resource file names relative to that directory. If you decide not to load resource files from the top of the package tree, consider the following information concerning resource file naming.

When you load a resource file, `loadjava` derives the name of the resource schema object from the file name that you enter on the command line. Suppose you type the following relative and absolute pathnames on the command line:

```
cd \scott\javastuff
loadjava options alpha\beta\x.properties
loadjava options  \scott\javastuff\alpha\beta\x.properties
```

Although you have specified the same file with a relative and an absolute pathname, `loadjava` creates *two* schema objects:

- `alpha\beta\x.properties`
- `\scott\javastuff\alpha\beta\x.properties`.

`loadjava` generates the resource schema object's name from the file names *you entered.*

Classes can refer to resource files relatively (for example, `b.properties`) or absolutely (for example, `\a\b.properties`). To ensure that `loadjava` and the class loader use the same name for a schema object, pass `loadjava` *the name of the resource that the class passes to the* `java.lang.Object.getResource` *or* `java.lang.Class.`*`getResourceAsStream`* *method.*

Instead of remembering whether classes use relative or absolute resource names and changing directories so that you can enter the correct name on the command line, you can load resource files into a JAR file, as follows:

```
cd \scott\javastuff
jar -cf alpharesources.jar alpha\*.properties
loadjava options alpharesources.jar
```

Or, to simplify further, put both the class and resource files in a JAR, which makes the following invocations equivalent:

```
loadjava options alpha.jar
loadjava options \scott\javastuff\alpha.jar
```

Example

The following loads a class and resource file into Oracle Database Lite. It uses the `force` option; if the database already contains objects with the specified names, `loadjava` replaces them.

```
c:\> loadjava -u scott/tiger@c:\Olite\Mobile\Sdk\OLDB40\Polite.odb -f Agent.class\
images.dat
```

11.3.1.1.2 Using CREATE JAVA

To load Java classes manually, use the following syntax:

```
CREATE [OR REPLACE] [AND RESOLVE] [NOFORCE]
   JAVA {CLASS [SCHEMA <schema_name>] |
   RESOURCE NAMED [<schema_name>.]<primary_name>}
   [<invoker_rights_clause>]
   RESOLVER <resolver_spec>]
   USING BFILE ('<dir_path>', '<class_name>')
```

The following apply to the CREATE JAVA parameters:

- The OR REPLACE clause, if specified, recreates the function or procedure if one with the same name already exists in the database.
- For compatibility with the Oracle database, Oracle Database Lite recognizes but ignores the `<resolver_spec>` clause. Unlike the Oracle database, Oracle Database Lite does not resolve class dependencies. When loading classes manually, be sure to load all dependent classes.
- Oracle Database Lite recognizes but ignores `<invoker_rights_clause>`.

Example

The following demonstrates a CREATE JAVA statement. It loads a class named `Employee` into the database.

```
CREATE JAVA CLASS USING BFILE ('c:\myprojects\java',
   'Employee.class');
```

11.3.1.2 Publishing Stored Procedures to SQL

After loading the Java class into the Oracle Database Lite database using `loadjava` or CREATE JAVA, you publish any static method in the class that you want to call from SQL. To publish the method, create a call specification (call spec) for it. The call spec maps the Java method's name, parameter types, and return types to SQL counterparts.

You do not need to publish every stored procedure, only those that serve as entry points for your application. In a typical implementation, many stored procedures are called only by other stored procedures, not by SQL users.

To create a call spec, use the SQL commands CREATE FUNCTION or CREATE PROCEDURE. Use CREATE FUNCTION for methods that return a value, and CREATE PROCEDURE for methods that do not return a value. The CREATE FUNCTION and CREATE PROCEDURE statements have the following syntax.

```
CREATE [OR REPLACE]
   { PROCEDURE [<schema_name>.]<proc_name> [([<sql_parms>])] |
   FUNCTION [<schema_name>.]<func_name> [([<sql_parms>])]
   RETURN <sql_type> }
   <invoker_rights_clause>
   { IS | AS } LANGUAGE JAVA NAME
   '<java_fullname>  ([<java_parms>])
   [return <java_type_fullname>]';
   /
```

The following apply to this statement's keywords and parameters:

- <sql_parms> has the following format:

  ```
  <arg_name> [IN | OUT | IN OUT]
           <datatype>
  ```

- <java_parms> is the fully qualified name of the Java datatype.
- For compatibility with the Oracle database, Oracle Database Lite recognizes but ignores the `<invoker_rights_clause>` clause.
- <java_fullname> is the fully qualified name of a static Java method.
- `IS` and `AS` are synonymous.

For example, assume the following class has been loaded into the database:

```
import java.sql.*;
import java.io.*;

public class GenericDrop {
   public static void dropIt (Connection conn, String object_type,
                      String object_name) throws SQLException {
      // Build SQL statement
      String sql = "DROP " + object_type + " " + object_name;
      try {
         Statement stmt = conn.createStatement();
         stmt.executeUpdate(sql);
         stmt.close();
      } catch (SQLException e) {
         System.err.println(e.getMessage());}
   }  // dropIt
}  // GenericDrop
```

Class `GenericDrop` has one method named `dropIt`, which drops any kind of schema object. For example, if you pass the arguments "table" and "emp" to `dropIt`, the method drops the database table EMP from your schema.

The following call specification publishes the method to SQL:

```
CREATE OR REPLACE PROCEDURE drop_it (
      obj_type VARCHAR2,
      obj_name VARCHAR2)
   AS LANGUAGE JAVA
```

```
   NAME 'GenericDrop.dropIt(java.sql.Connection,
      java.lang.String, java.lang.String)';
   /
```

Notice that you must fully qualify the Java datatype parameters.

Given that you have a table named TEMP defined in your schema, you can execute the drop_it procedure from SQL Plus as follows.

```
Select drop_it('TABLE', 'TEMP') from dual;
```

You can also execute the drop_it procedure from within a ODBC application using ODBC CALL statement. For more information, refer Section 11.3.3, "Calling Java Stored Procedures from ODBC".

11.3.1.3 Calling Published Stored Procedures

After publishing the stored procedure to SQL, you call it by using a SQL DML statement. For example, assume that this class is stored in the database:

```
public class Formatter {
   public static String formatEmp (String empName, String jobTitle) {
      empName = empName.substring(0,1).toUpperCase() +
         empName.substring(1).toLowerCase();
      jobTitle = jobTitle.trim().toLowerCase();
      if (jobTitle.equals("analyst"))
         return (new String(empName + " is an exempt analyst"));
      else
      return (new String(empName + " is a non-exempt " + jobTitle));
  }
}
```

Class `Formatter` has one method named `formatEmp`, which returns a formatted string containing an employee's name and job status. Create a call spec for `Formatter` as follows:

```
CREATE OR REPLACE FUNCTION format_emp (ename VARCHAR2, job VARCHAR2)
   RETURN VARCHAR2
   AS LANGUAGE JAVA
   NAME 'Formatter.formatEmp (java.lang.String, java.lang.String)
   return java.lang.String';
   /
```

The call spec publishes the method `formatEmp` as `format_emp`. Invoke it as follows:

```
SELECT FORMAT_EMP(ENAME, JOB) AS "Employees" FROM EMP
   WHERE JOB NOT IN ('MANAGER', 'PRESIDENT') ORDER BY ENAME;
```

This statement produces the following output:

```
Employees
-------------------------------------------
Adams is a non-exempt clerk
Allen is a non-exempt salesman
Ford is an exempt analyst
James is a non-exempt clerk
Martin is a non-exempt salesman
Miller is a non-exempt clerk
Scott is an exempt analyst
Smith is a non-exempt clerk
Turner is a non-exempt salesman
Ward is a non-exempt salesman
```

Note: Oracle Database Lite does not support the Oracle database SQL CALL statement for invoking stored procedures.

For information on calling stored procedures from C and C++ applications, see Section 11.3.3, "Calling Java Stored Procedures from ODBC".

11.3.1.4 Dropping Published Stored Procedures

To remove classes from Oracle Database Lite, use either of the following:

- the `dropjava` utility
- the `SQL DROP JAVA` statement

To drop call specifications, use either `DROP FUNCTION` or `DROP PROCEDURE`.

11.3.1.4.1 Using dropjava

`dropjava` is a command-line utility that automates the task of dropping Java classes from Oracle Database Lite and Oracle databases. `dropjava` converts file names into the names of schema objects and drops the schema objects. Use the following syntax to invoke `dropjava`:

```
dropjava {-user | -u} username/password[@database]
  [-option] filename filename ...
```

Arguments

This section describes the arguments to `dropjava`.

User

The `user` argument specifies a username, password, and absolute path to the database file in the following format:

```
<user>/<password>[@<database>]
```

For example:

```
scott/tiger@c:\Olite\Mobile\Sdk\OLDB40\Polite.odb
```

Option

By specifying the verbose option (`-verbose | -v`), you can direct `dropjava` to produce detailed status messages while running.

Oracle database supports additional options for `dropjava`, as described in the *Oracle Database Java Developer's Guide*. If used with Oracle Database Lite, the additional options are recognized but not supported. Using them does not result in an error.

For a complete list of supported and recognized options, from the command prompt type:

```
dropjava -help
```

Filename

For the `filename` argument, you can specify any number of Java class, source, JAR, ZIP, and resource files, in any order. JAR and ZIP files must be uncompressed. `dropjava` interprets most file names the same way `loadjava` does:

- For class files, `dropjava` finds the class name in the file and drops the corresponding schema object.

- For source files, `dropjava` finds the first class name in the file and drops the corresponding schema object.
- For JAR and ZIP files, `dropjava` processes the archived file names as if they had been entered on the command line.

If a file name has an extension other than **.java**, **.class**, **.jar**, or **.zip**, or has no extension, then `dropjava` assumes that the file name is the name of a schema object, then drops all source, class, and resource schema objects with that name. If `dropjava` encounters a file name that does not match the name of any schema object, it displays an error message and then processes the remaining file names.

11.3.1.4.2 Using SQL Commands To drop a Java class from Oracle Database Lite manually, use the DROP JAVA statement. DROP JAVA has the following syntax:

```
DROP JAVA { CLASS | RESOURCE } [<schema-name> .]<object_name>
```

To drop a call specification, use the DROP FUNCTION or DROP PROCEDURE statement:

```
DROP { FUNCTION | PROCEDURE } [<schema-name>.]<object_name>
```

The schema name, if specified, is recognized but not supported by Oracle Database Lite.

11.3.1.5 Example

The following example creates a Java stored procedure using the load and publish model.

In this example, you store the Java method `paySalary` in the Oracle Database Lite. `paySalary` computes the take-home salary for an employee.

This example covers the following steps.

- Step 1: Create the Java Class
- Step 2: Load the Java Class into the Database
- Step 3: Publish the Function
- Step 4: Execute the Function

More examples of Java stored procedures are located in the *<ORACLE_HOME>*`\Mobile\SDK\samples\jdbc` directory.

Step 1: Create the Java Class

Create the Java class `Employee` in the file **Employee.java**. The `Employee` class implements the `paySalary` method:

```
import java.sql.*;
public class Employee {
   public static String paySalary(float sal, float fica, float sttax,
                            float ss_pct, float espp_pct)  {
      float deduct_pct;
      float net_sal;
        // compute take-home salary
      deduct_pct = fica + sttax + ss_pct + espp_pct;
      net_sal = sal * deduct_pct;
      String returnstmt = "Net salary is " + net_sal;
      return returnstmt;
   } // paySalary
}
```

Note: The keyword "public class" should not be used in a comment before the first public class statement.

Step 2: Load the Java Class into the Database

From mSQL, load the Java class using CREATE JAVA, as follows:

```
CREATE JAVA CLASS USING BFILE ('c:\myprojects\doc',
'Employee.class');
```

This command loads the Java class located in c:\myprojects\doc into the Oracle Database Lite.

Step 3: Publish the Function

Create a call spec for the `paySalary` method. The following call spec publishes the Java method `paySalary` as function `pay_salary`:

```
CREATE FUNCTION pay_salary (
sal float, fica float, sttax float, ss_pct float, espp_pct float)
RETURN VARCHAR2
AS LANGUAGE JAVA NAME
'Employee.paySalary(float, float, float, float, float)
return java.lang.String';
/
```

Step 4: Execute the Function

To execute `pay_salary` in mSQL:

```
SELECT pay_salary(6000.00, 0.2, 0.0565, 0.0606, 0.1)
FROM DUAL;
```

To execute `pay_salary` in ODBC:

```
SQLExecDirect(hstm,
   "SELECT pay_salary(6000.00,0.2,0.0565,0.0606,0.1)
   FROM DUAL);
```

Because the arguments to `pay_salary` are constants, the FROM clause specifies the dummy table DUAL. This SELECT statement produces the following output:

```
Net salary is 2502.6
```

11.3.2 Model 2: Using the Attached Stored Procedure Development Model

This section describes how to create stored procedures by attaching classes to tables. This information is specific to Oracle Database Lite; you cannot attach classes to Oracle database tables as described here. The load and publish model for developing stored procedures, described in Section 11.3.1, "Model 1: Using the Load and Publish Stored Procedure Development Model", only supports class (static) methods. By attaching classes to tables, however, you can store and call Java class and instance methods.

To create attached stored procedures, develop the class that you want to attach. Make sure that the class compiles and executes without errors. Then attach the class to an Oracle Database Lite table. Once the class is attached, the methods in the class become the table-level and row-level stored procedures of the table.

11.3.2.1 Attaching a Java Class to a Table

To attach a Java class to a table, use the SQL command ALTER TABLE. The ALTER TABLE command has the following syntax:

```
ALTER TABLE [schema.]table
   ATTACH JAVA {CLASS|SOURCE} "cls_or_src_name "
   IN {DATABASE|'cls_or_src_path '}
   [WITH CONSTRUCTOR ARGS (col_name_list )]
```

You can attach either a source file or a class file. Source files are compiled by the Java compiler found in the system path.

`cls_or_src_name` specifies a fully qualified name of a class or source file. This includes the package name followed by class name, such as `Oracle.lite.Customer`. Do not include the file extension in the class or source file name. The name is case-sensitive. If you use lowercase letters, enclose the name in double quotes (" "). Make sure that the source or class is in the package specified by `cls_or_src_name`. (The source file of the example class `Customer` should contain the line "`package Oracle.lite;`".) The class file is stored in the database in the same package. Oracle Database Lite creates the package if it does not already exist.

If you have already attached the Java class to another table in the database, you can use the IN DATABASE clause. If the class has not yet been attached, specify the directory location of the class or source file in `cls_or_src_path`.

Prior to executing a row-level stored procedure, Oracle Database Lite creates a Java object for the row, if one does not already exist. If the ALTER TABLE statement includes a WITH CONSTRUCTOR clause, Oracle Database Lite creates the object using the class constructor that is the best match given the datatypes of the columns included in `col_name_list`. If the ALTER TABLE statement does not include a WITH CONSTRUCTOR clause, Oracle Database Lite uses the default constructor.

You can use the ODBC functions `SQLProcedures` and `SQLProcedureColumns` to retrieve information about methods defined in a table.

11.3.2.2 Table-Level Stored Procedures

Table-level stored procedures are the static methods of the attached Java class. Therefore, when executing the method, Oracle Database Lite does not instantiate the class to which it belongs. In a call statement, you refer to table-level stored procedures as *table_name.method_name*.

Statement-level triggers and BEFORE INSERT and AFTER DELETE row-level triggers (see section "Section 11.5.1, "Statement-Level vs. Row-Level Triggers"") must be table-level stored procedures.

11.3.2.3 Row-Level Stored Procedures

Row-level stored procedures are the non-static methods in the attached Java class. To execute a row-level stored procedure, Oracle Database Lite instantiates the class to which the procedure belongs. The arguments to the class constructor determine which column values the constructor uses as parameters to create the class instances. In a call statement, you refer to row-level stored procedures as method_name (without the table qualifier). Row-level triggers can indirectly execute row-level stored procedures.

11.3.2.4 Calling Attached Stored Procedures

After attaching the class to a table using the ALTER TABLE statement, you can call it with a SELECT statement. Refer to table-level stored procedures as *table_name.method_name* and row-level procedures as *method_name*.

For example, to execute a table-level stored procedure:

```
SELECT table_name.proc_name[arg_list]
   FROM {DUAL|[schema.]table WHERE condition};
```

The `proc_name` is the name of the table-level stored procedure. Each argument in `arg_list` is either a constant or a reference to a column in the table. If all the arguments of `arg_list` are constants, the FROM clause should reference the dummy table DUAL.

Execute a row-level stored procedure as follows:

```
SELECT [schema.]proc_name[arg_list]
   FROM [schema.]table
   WHERE condition;
```

If you call a procedure in the form *table_name.method_name*, and a table or method with that name does not exist, Oracle Database Lite assumes that table_name refers to a schema name and *method_name* refers to a procedure name. If you reference method_name only, Oracle Database Lite assumes that the referenced method is a row-level procedure. If there is no such procedure defined, however, Oracle Database Lite assumes that method_name refers to a procedure in the current schema.

Note: Oracle Database Lite does not support the Oracle8*i* SQL CALL statement for invoking stored procedures.

You can use a callable statement to execute a procedure from ODBC or JDBC applications. For more information, see Chapter 10, "JDBC Programming". For additional information, see Section 11.3.3, "Calling Java Stored Procedures from ODBC".

11.3.2.5 Dropping Attached Stored Procedures

You use the ALTER TABLE command to drop stored procedures. ALTER TABLE has the following syntax:

```
ALTER TABLE [schema.]table
   DETACH [AND DELETE] JAVA CLASS "class_name"
```

Note: You must enclose the class name in double quotes (" ") if it contains lowercase letters.

Detaching the Java class does not delete it from the database. To delete the Java class file from the database, use the DETACH AND DELETE statement.

If you delete a Java class from the database after invoking it as a stored procedure or trigger, the class remains in the Java Virtual Machine attached to the application. To unload the class from the Java Virtual Machine, commit changes to the database, if necessary, and close all applications connected to the database. To replace a Java class, you must close all connections to the database and reload the class.

11.3.2.6 Example

The following example shows how to create a Java stored procedure in Oracle Database Lite. In this example, you attach the Java method `paySalary` to the table EMP. `paySalary` computes the take-home salary for an employee.

This example covers the following steps:

- Step 1: Create the Table
- Step 2: Create the Java Class
- Step 3: Attach the Java Class to the Table
- Step 4: Execute the Method

Step 1: Create the Table

Create the table using the following SQL command:

```
CREATE TABLE EMP(Col1 char(10));
```

Step 2: Create the Java Class

Create the Java class `Employee` in the file **Employee.java**. The `Employee` class implements the `paySalary` method:

```
import java.sql.*;
public class Employee {
   public static String paySalary(float sal, float fica, float sttax,
                             float ss_pct, float espp_pct)  {
      float deduct_pct;
      float net_sal;
        // compute take-home salary
      deduct_pct = fica + sttax + ss_pct + espp_pct;
      net_sal = sal * deduct_pct;
      String returnstmt = "Net salary is " + net_sal;
      return returnstmt;
   } // paySalary
}
```

Step 3: Attach the Java Class to the Table

From mSQL, attach the Java class using the ALTER TABLE command:

```
ALTER TABLE EMP ATTACH JAVA SOURCE "Employee" IN 'C:\tmp';
```

This command attaches the Java source file for the `Employee` class, which resides in the directory `C:\tmp`, to the EMP table.

Step 4: Execute the Method

To execute the `paySalary` method in mSQL, type the following statement:

```
SELECT EMP."paySalary"(6000.00,0.2,0.0565,0.0606,0.1)
   FROM DUAL;
```

To execute `paySalary` from ODBC, invoke `SQLExecDirect`:

```
SQLExecDirect(hstm,
   "SELECT EMP.\"paySalary\"(6000.00,0.2,0.0565,0.0606,0.1)
   FROM DUAL);
```

This statement produces the following result:

```
Net salary is 2502.6
```

11.3.3 Calling Java Stored Procedures from ODBC

When invoking a Java stored procedure from a multithreaded C or C++ application, you should load `jvm.dll` from the application's `main` function. This resolves a problem that occurs with the Java Virtual Machine's garbage collection when a C or

C++ application creates multiple threads that invoke a stored procedure directly or indirectly. The Java Virtual Machine runs out of memory because the threads do not detach from the Java Virtual Machine before exiting. Since Oracle Database Lite cannot determine whether the Java Virtual Machine or the user application created the thread, it does not attempt to detach them.

`main` should load the library before taking any other action, as follows:

```
int main (int argc, char** argv)
{
   LoadLibrary("jvm.dll");
   ...
}
```

The library loads the Java Virtual Machine into the application's main thread. It attempts to detach any thread from the Java Virtual Machine if the thread detaches from the process. The **jvm.dll** behaves correctly even if the thread is not attached to a Java Virtual Machine.

11.4 Java Datatypes

Oracle Database Lite performs type conversion between Java and SQL datatypes according to standard SQL rules. For example, if you pass an integer to a stored procedure that takes a string, Oracle Database Lite converts the integer to a string. For information about row-level triggers arguments, see Section 11.5.5, "Trigger Arguments". For a complete list of Java to SQL datatype mappings, see Section 10.5.1, "Java Datatypes".

Note: In Oracle database, DATE columns are created as TIMESTAMP. You must specify trigger methods accordingly.

Java does not allow a method to change the value of its arguments outside the scope of the method. However, Oracle Database Lite supports IN, OUT, and IN/OUT parameters.

Many Java datatypes are immutable or do not support NULL values. To pass NULL values and use IN/OUT parameters for those datatypes, a stored procedure can use an array of that type or use the equivalent object type. Table 11–2 shows the Java integer datatypes you can use to enable an integer to be an IN/OUT parameter or carry a NULL value.

Table 11–2 The Java Integer Datatypes

Java Argument	Can Be IN/OUT	Can Be NULL
`int`	No	No
`int[]`	Yes	Yes
`Integer`	No	Yes
`Integer[]`	Yes	Yes
`int[][]`	Yes	Yes

You can use mutable Java datatypes, such as `Date`, to pass a NULL or an IN/OUT parameter. However, use a `Date` array if a stored procedure needs to change the NULL status of its argument.

Note: Passing a NULL when the corresponding Java argument cannot be NULL causes an error.

11.4.1 Declaring Parameters

The return value of a Java method is the OUT parameter of the procedure. A primitive type or immutable reference type can be an IN parameter. A mutable reference type or array type can be an IN/OUT parameter. Table 11–3 shows the Java type to use to make the corresponding Oracle Database Lite parameter an IN/OUT parameter.

Table 11–3 Java Types for Oracle Database Lite IN/OUT Parameters

For IN/OUT parameters of type...	Use...
Number	`Integer[]` or `int[]`
Binary	`byte[]` or `byte[][]`
String	`string[]`

If the stored procedure takes a `java.sql.Connection`, Oracle Database Lite automatically supplies the argument using the value of the current transaction or row. This argument is the first argument passed to the procedure.

11.4.2 Using Stored Procedures to Return Multiple Rows

You can use stored procedures to return multiple rows. You can invoke stored procedures that return multiple rows only from JDBC or ODBC applications, however. For a stored procedure to return multiple rows, its corresponding Java method must return a `java.sql.ResultSet` object. By executing a SELECT statement, the Java method obtains a `ResultSet` object to return. The column names of the `ResultSet` are specified in the SELECT statement. If you need to address the result columns by different names than those used in the table, the SELECT statement should use aliases for the result columns. For example:

```
SELECT emp.name Name,  dept.Name Dept
   FROM  emp, dept
   WHERE emp.dept# = dept.dept#;
```

Because the return type of a stored procedure that returns multiple rows must be `java.sql.ResultSet`, the signature of that stored procedure cannot be used to obtain the column names or types of the result. Consequently, you should design additional tables to track the column names or result types for the stored procedures. For example, if you embed the preceding SELECT statement in a Java method, the method return type should be `java.sql.ResultType`, not `char Name` and `char Dept`.

Note: You can only create Java stored procedures that return multiple rows using the attached stored procedure development model, described in Section 11.3.2, "Model 2: Using the Attached Stored Procedure Development Model".

11.4.2.1 Returning Multiple Rows in ODBC

To execute a stored procedure that returns multiple rows in an OBDC application, use the following CALL statement, in which P is the name of the stored procedure and a_1 through a_n are arguments to the stored procedure.

```
{CALL P(a1,...,an)}
```

You use a marker (`?`) for any argument that should be bound to a value before the statement executes. When the statement executes, the procedure runs and the cursor on the result set is stored in the statement handle. Subsequent fetches using this statement handle return the rows from the procedure.

After you execute the `CALL` statement, use `SQLNumResultCols` to find the number of columns in each row of the result. Use the `SQLDescribeCol` function to return the column name and datatype.

11.4.2.2 Example

The following example shows how to use ODBC to execute a stored procedure that returns multiple rows. This example does not use the `SQLNumResultCols` or `SQLDescribeCol` functions. It assumes that you have created a stored procedure, which you have published to SQL as PROC. PROC takes an integer as an argument.

```
rc = SQLPrepare(StmtHdl, "{call PROC(?)}", SQL_NTS);
CHECK_STMT_ERR(StmtHdl, rc, "SQLPrepare");

rc = SQLBindParameter(StmtHdl, 1, SQL_PARAM_INPUT_OUTPUT,
   SQL_C_LONG,SQL_INTEGER, 0, 0, &InOutNum, 0, NULL);
CHECK_STMT_ERR(StmtHdl, rc, "SQLBindParameter");

rc = SQLExecute(StmtHdl);
CHECK_STMT_ERR(StmtHdl, rc, "SQLExecute");

/* you can use SQLNumResultCols and SQLDescribeCol here */

rc = SQLBindCol(StmtHdl, 1, SQL_C_CHAR, c1, 20, &pcbValue1);
CHECK_STMT_ERR(StmtHdl, rc, "SQLBindCol");

rc = SQLBindCol(StmtHdl, 2, SQL_C_CHAR, c2, 20, &pcbValue2);
CHECK_STMT_ERR(StmtHdl, rc, "SQLBindCol");

while ((rc = SQLFetch(StmtHdl)) != SQL_NO_DATA_FOUND) {
   CHECK_STMT_ERR(StmtHdl, rc, "SQLFetch");
   printf("%s, %s\n", c1, c2);
}
```

11.5 Using Triggers

Triggers are stored procedures that execute, or "fire", when a specific event occurs. A trigger can fire when a column is updated, or when a row is added or deleted. The trigger can fire before or after the event.

Triggers are commonly used to enforce a database's business rules. For example, a trigger can verify input values and reject an illegal insert. Similarly, a trigger can ensure that all tables depending on a particular row are brought to a consistent state before the row is deleted.

11.5.1 Statement-Level vs. Row-Level Triggers

There are two types of triggers: row-level and statement-level. A row-level trigger is fired once for each row affected by the change to the database. A statement-level trigger fires only once, even if multiple rows are affected by the change.

The BEFORE INSERT and AFTER DELETE triggers can only fire table-level stored procedures, since a row object cannot be instantiated to call the procedures. The AFTER INSERT, BEFORE DELETE, and UPDATE triggers may fire table-level or row-level stored procedures.

11.5.2 Creating Triggers

Use the CREATE TRIGGER statement to create a trigger. The CREATE TRIGGER statement has the following syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name {BEFORE | AFTER} [{INSERT | DELETE |
   UPDATE [OF column_list]} [OR ]] ON table_reference
   [FOR EACH ROW] procedure_ref
   (arg_list)
```

In the CREATE TRIGGER syntax:

- Use the OR clause to specify multiple triggering events.
- Use FOR EACH ROW to create a row-level trigger. For a table-level trigger, do not include this clause.
- Use `procedure_ref` to identify the stored procedure to execute.

You can create multiple triggers of the same kind for a table if each trigger has a unique name within a schema.

In the following example, assume that you have stored and published a procedure as PROCESS_NEW_HIRE. The trigger AIEMP fires every time a row is inserted into the EMP table.

```
CREATE TRIGGER AIEMP AFTER INSERT ON EMP FOR EACH ROW
   PROCESS_NEW_HIRE(ENO);
```

UPDATE triggers that use the same stored procedure for different columns of a table are fired only once when a subset of the columns is modified within a statement. For example, the following statement creates a BEFORE UPDATE trigger on table T, which has columns C1, C2, and C3:

```
CREATE TRIGGER T_TRIGGER BEFORE UPDATE OF C1,C2,C3 ON T
   FOR EACH ROW trigg(old.C1,new.C1,old.C2,new.C2,
   old.C3,new.C3);
```

This update statement fires `T_TRIGGER` only once:

```
UPDATE T SET C1 = 10, C2 = 10 WHERE ...
```

11.5.2.1 Enabling and Disabling Triggers

When you create a trigger, it is automatically enabled. To disable triggers, use the ALTER TABLE or ALTER TRIGGER statement.

To enable or disable individual triggers, use the ALTER TRIGGER statement, which has the following syntax:

```
ALTER TRIGGER <trigger_name> {ENABLE | DISABLE}
```

To enable or disable all triggers attached to a table, use ALTER TABLE:

```
ALTER TABLE <table_name> {ENABLE | DISABLE} ALL TRIGGERS
```

11.5.3 Dropping Triggers

To drop a trigger, use the DROP TRIGGER statement, which has the following syntax:

```
DROP TRIGGER [schema.]trigger
```

11.5.4 Trigger Example

This example creates a trigger. It follows the development model described in Section 11.3.2, "Model 2: Using the Attached Stored Procedure Development Model". For an example of creating triggers using the load and publish model, see Section 11.5.6, "Trigger Arguments Example". In the example, you first create a table and a Java class. Then you attach the class to the table. And finally, you create and fire the trigger.

The `SalaryTrigger` class contains the `check_sal_raise` method. The method prints a message if an employee gets a salary raise of more than ten percent. The trigger fires the method before updating a salary in the EMP table.

Since `check_sal_raise` writes a message to standard output, use mSQL to issue the mSQL commands in the example. To start mSQL, invoke the Command Prompt and enter the following.

```
msql username/password@connect_string
```

`connect_string` is JDBC URL syntax. For example, to connect to the default database as user SYSTEM, at the Command Prompt.

```
msql system/passwd@jdbc:polite:polite
```

At the mSQL command line, create and populate the EMP table as follows.

```
CREATE TABLE EMP(E# int, name char(10), salary real,
   Constraint E#_PK primary key (E#));

INSERT INTO EMP VALUES (123,'Smith',60000);
INSERT INTO EMP VALUES (234,'Jones',50000);
```

Place the following class in **SalaryTrigger.java**:

```
class SalaryTrigger {
   private int eno;
   public SalaryTrigger(int enum) {
      eno = enum;
   }
   public void check_sal_raise(float old_sal,
      float new_sal)
   {
      if (((new_sal - old_sal)/old_sal) > .10)
      {
         // raise too high  do something here
         System.out.println("Raise too high for employee " + eno);
      }
   }
}
```

The `SalaryTrigger` class constructor takes an integer, which it assigns to attribute `eno` (the employee number). An instance of `SalaryTrigger` is created for each row (that is, for each employee) in the table `EMP`.

The `check_sal_raise` method is a non-static method. To execute, it must be called by an object of its class. Whenever the salary column of a row in `EMP` is modified, an instance of `SalaryTrigger` corresponding to that row is created (if it does not already exist) with the employee number (*E#)* as the argument to the constructor. The trigger then calls the `check_sal_raise` method.

After creating the Java class, you attach it to the table, as follows:

```
ALTER TABLE EMP ATTACH JAVA SOURCE "SalaryTrigger" IN '.'
   WITH CONSTRUCTOR ARGS(E#);
```

This statement directs Oracle Database Lite to compile the Java source file **SalaryTrigger.java** found in the current directory, and attach the resulting class to the EMP table. The statement also specifies that, when instantiating the class, Oracle Database Lite should use the constructor that takes as an argument the value in the `E#` column.

After attaching the class to the table, create the trigger as follows:

```
CREATE TRIGGER CHECK_RAISE BEFORE UPDATE OF SALARY ON EMP FOR EACH ROW
   "check_sal_raise"(old.salary, new.salary);
/
```

This statement creates a trigger called `check_raise`, which fires the `check_sal_raise` method before any update to the salary column of any row in EMP. Oracle Database Lite passes the old value and the new value of the salary column as arguments to the method.

In the example, a row-level trigger fires a row-level procedure (a non-static method). A row-level trigger can also fire table-level procedures (static methods). However, because statement-level triggers are fired once for an entire statement and a statement may affect multiple rows, a statement-level trigger can only fire a table-level procedure.

The following command updates the salary and fires the trigger:

```
UPDATE EMP SET SALARY = SALARY + 6100  WHERE E# = 123;
```

This produces the following output:

```
Raise too high for employee 123
```

11.5.5 Trigger Arguments

If using attached stored procedures, as described in Section 11.3.2, "Model 2: Using the Attached Stored Procedure Development Model", row-level triggers do not support Java-to-SQL type conversion. Therefore, the Java datatype of a trigger argument must match the corresponding SQL datatype (shown in section Section 11.4, "Java Datatypes") of the trigger column. However, if you are using the load and publish model, Oracle Database Lite supports datatype casting.

Table 11–4 describes how trigger arguments work in each type of column.

Table 11–4 Trigger Arguments

Trigger Argument	New Column Access	Old Column Access
`insert`	Yes	No

Table 11–4 (Cont.) Trigger Arguments

Trigger Argument	New Column Access	Old Column Access
`delete`	No	Yes
`update`	Yes	Yes

Note: Triggers that have a `java.sql.Connection` object as an argument may be used only with applications that use the relational model.

11.5.6 Trigger Arguments Example

The following example shows how to create triggers that use IN/OUT parameters.

1. First, create the Java class `EMPTrigg`.

```
import java.sql.*;

public class EMPTrigg {
   public static final String goodGuy = "Oleg";

   public static void NameUpdate(String oldName, String[] newName)
   {
      if (oldName.equals(goodGuy))
         newName[0] = oldName;
   }

   public static void SalaryUpdate(String name, int oldSalary,
          int newSalary[])
   {
      if (name.equals(goodGuy))
         newSalary[0] = Math.max(oldSalary, newSalary[0])*10;
   }

   public static void AfterDelete(Connection conn,
                  String name, int salary) {
      if (name.equals(goodGuy))
         try {
            Statement stmt = conn.createStatement();
            stmt.executeUpdate(
             "insert into employee values('" + name + "', " +
                       salary + ")");
            stmt.close();
         } catch(SQLException e) {}
     }
 }
```

2. Create a new table EMPLOYEE and populate it with values.

```
CREATE TABLE EMPLOYEE(NAME VARCHAR(32), SALARY INT);
INSERT INTO EMPLOYEE VALUES('Alice', 100);
INSERT INTO EMPLOYEE VALUES('Bob', 100);
INSERT INTO EMPLOYEE VALUES('Oleg', 100);
```

3. Next, load the class into Oracle Database Lite.

```
CREATE JAVA CLASS USING BFILE ('c:\myprojects', 'EMPTrigg.class');
```

4. Use the `CREATE PROCEDURE` statement to publish the `EMPTrigg` methods that you want to call:

```
CREATE PROCEDURE NAME_UPDATE(
   OLD_NAME IN VARCHAR2, NEW_NAME IN OUT VARCHAR2)
   AS LANGUAGE JAVA NAME
   'EMPTrigger.NameUpdate(java.lang.String, java.lang.String[])';
   /

CREATE PROCEDURE SALARY_UPDATE(
   ENAME VARCHAR2, OLD_SALARY INT, NEW_SALARY IN OUT INT)
   AS LANGUAGE JAVA NAME
   'EMPTrigger.SalaryUpdate(java.lang.String, int, int[])';
   /

CREATE PROCEDURE AFTER_DELETE(
   ENAME VARCHAR2, SALARY INT)
   AS LANGUAGE JAVA NAME
   'EMPTrigger.AfterDelete(java.sql.Connection,
           java.lang.String, int)';
   /
```

5. Now, create a trigger for each procedure:

```
CREATE TRIGGER NU BEFORE UPDATE OF NAME ON EMPLOYEE FOR EACH ROW
   NAME_UPDATE(old.name, new.name);

CREATE TRIGGER SU BEFORE UPDATE OF SALARY ON EMPLOYEE FOR EACH ROW
   SALARY_UPDATE(name, old.salary, new.salary);

CREATE TRIGGER AD AFTER DELETE ON EMPLOYEE FOR EACH ROW
   AFTER_DELETE(name, salary);
```

6. Enter the following commands to fire the triggers and view the results:

```
SELECT * FROM EMPLOYEE;
UPDATE EMPLOYEE SET SALARY=0 WHERE NAME = 'Oleg';
SELECT * FROM EMPLOYEE;

DELETE FROM EMPLOYEE WHERE NAME = 'Oleg';
SELECT * FROM EMPLOYEE;

UPDATE EMPLOYEE SET NAME='TEMP' WHERE NAME = 'Oleg';
DELETE FROM EMPLOYEE WHERE NAME = 'TEMP';

SELECT * FROM EMPLOYEE;
```

11.6 Creating a Stored Procedure and Trigger

In this tutorial, you create a Java class `EMAIL`, load the class into Oracle Database Lite, publish its method to SQL, and create a trigger for the method. The `EMAIL` class appears in the source file `EMAIL.java`, and is available in the Java examples directory at the following location.

<*ORACLE_HOME*>`\Mobile\Sdk\Samples\JDBC`

`EMAIL` has a method named `assignEMailAddress`, which generates an email address for an employee based on the first letter of the employee's first name and up to seven letters of the last name. If the address is already assigned, the method attempts to find a unique email address using combinations of letters in the first and last name.

After creating the class, you load it into Oracle Database Lite using mSQL. For this example you use the SQL statement `CREATE JAVA`. Alternatively, you can use the `loadjava` utility to load the class into Oracle Database Lite. After loading the class, you publish the `assignEMailAddress` method to SQL.

Finally, you create a trigger that fires the `assignEMailAddress` method whenever a row is inserted into T_EMP, the table that contains the employee information.

As arguments, `assignEMailAddress` takes a JDBC connection object, the employee's identification number, first name, middle initial, and last name. Oracle Database Lite supplies the JDBC connection object argument. You do not need to provide a value for the connection object when you execute the method. `assignEMailAddress` uses the JDBC connection object to ensure that the generated e-mail address is unique.

11.6.1 Start mSQL

Start mSQL and connect to the default Oracle Database Lite. Since the Java application in this tutorial prints to standard output, use the DOS version of mSQL. From a DOS prompt, type:

```
msql system/mgr@jdbc:polite:polite
```

The SQL prompt should appear.

11.6.2 Create a Table

To create a table, type:

```
CREATE TABLE T_EMP(ENO INT PRIMARY KEY,
   FNAME VARCHAR(20),
   MI CHAR,
   LNAME VARCHAR(20),
   EMAIL VARCHAR(8));
```

11.6.3 Create a Java Class

Create and compile the Java class `EMAIL` in the file `EMAIL.java` in `C:\tmp`. `EMAIL.java` implements the `assignEMailAddress` method. The code sample given below lists the contents of this file. You can copy this file from the following location.

<*ORACLE_HOME*>`\Mobile\Sdk\Samples\JDBC`

```
import java.sql.*;

public class EMAIL {
   public static void assignEMailAddress(Connection conn,
            int eno, String fname,String lname)
            throws Exception
   {
      Statement stmt = null;
      ResultSet retset = null;
      String emailAddr;
      int i,j,fnLen, lnLen, rowCount;

      /* create a statement */
      try {
         stmt = conn.createStatement();
      }
      catch (SQLException e)
```

```
            {
               System.out.println("conn.createStatement failed: " +
               e.getMessage() + "\n");
               System.exit(0);
            }
            /* check fname and lname */
            fnLen = fname.length();
            if(fnLen > 8) fnLen = 8;
            if (fnLen == 0)
               throw new Exception("First name is required");
            lnLen = lname.length();
            if(lnLen > 8) lnLen = 8;
            if (lnLen == 0)
               throw new Exception("Last name is required");
            for (i=1; i <= fnLen; i++)
            {
               /* generate an e-mail address */
               j = (8-i) > lnLen? lnLen:8-i;
               emailAddr =
                     new String(fname.substring(0,i).toLowerCase()+
                     lname.substring(0,j).toLowerCase());
               /* check if this e-mail address is unique  */
               try {
                  retset = stmt.executeQuery(
                         "SELECT * FROM T_EMP  WHERE email = '"+
                         emailAddr+"'");
                  if(!retset.next()) {
                     /* e-mail address is unique;
                     * so update the email column */
                     retset.close();
                     rowCount = stmt.executeUpdate(
                         "UPDATE T_EMP SET EMAIL = '"
                         + emailAddr + "' WHERE ENO = "
                         + eno);
                     if(rowCount == 0)
                        throw new Exception("Employee "+fname+ " " +
                                lname + " does not exist");
                     else return;
                  }
               }
               catch (SQLException e) {
                  while(e != null) {
                     System.out.println(e.getMessage());
                     e = e.getNextException();
                  }
               }
            }
            /* Can't find a unique name */
            emailAddr = new String(fname.substring(0,1).toLowerCase() +
                 lname.substring(0,1).toLowerCase() + eno);
            rowCount = stmt.executeUpdate(
                 "UPDATE T_EMP SET EMAIL = '"
                 + emailAddr + "' WHERE ENO = "
                 + eno);
            if(rowCount == 0)
               throw new Exception("Employee "+fname+ " " +
                    lname + " does not exist");
            else return;
         }
      }
```

11.6.4 Load the Java Class File

To load the `EMAIL` class file into Oracle Database Lite, type:

```
CREATE JAVA CLASS USING BFILE
   ('c:\tmp', 'EMAIL.class');
```

If you want to make changes to the class after loading it, you need to:

1. Drop the class from the database, using `dropjava` or DROP JAVA CLASS
2. Commit your work
3. Exit mSQL
4. Restart mSQL

This unloads the class from the Java Virtual Machine.

11.6.5 Publish the Stored Procedure

You make the stored procedure callable from SQL by creating a call specification (call spec) for it. Since `assignEMailAddress` does not return a value, use the CREATE PROCEDURE command, as follows:

```
CREATE OR REPLACE PROCEDURE
   ASSIGN_EMAIL(E_NO INT, F_NAME VARCHAR2, L_NAME VARCHAR2)
   AS LANGUAGE JAVA NAME 'EMAIL.assignEMailAddress(java.sql.Connection,
int, java.lang.String,
     java.lang.String)';
```

11.6.6 Populate the Database

Insert a row into `T_EMP`:

```
INSERT INTO T_EMP VALUES(100,'John','E','Smith',null);
```

11.6.7 Execute the Procedure

To execute the procedure, type:

```
SELECT ASSIGN_EMAIL(100,'John','Smith')
  FROM dual
```

11.6.8 Verify the Email Address

To see the results of the ASSIGN_EMAIL procedure, type:

```
SELECT * FROM T_EMP;
```

This command produces the following output:

```
     ENO  FNAME                M LNAME                EMAIL
     ---- ------------------   - -------------------- --------
     100  John                 E Smith                jsmith
```

11.7 Create a Trigger

To make `ASSIGN_EMAIL` execute whenever a row is inserted into T_EMP, create an `AFTER INSERT` trigger for it. Create the trigger as follows:

```
CREATE TRIGGER EMP_TRIGG AFTER INSERT ON T_EMP FOR EACH ROW
  ASSIGN_EMAIL(eno,fname,lname);
```

A trigger named EMP_TRIGG fires every time a row is inserted into T_EMP. The actual arguments for the procedure are the values of the columns eno, fname, and lname.

You do not need to specify a connection argument.

11.7.1 Testing the Trigger

Test the trigger by inserting a row into T_EMP:

```
INSERT INTO T_EMP VALUES(200,'James','A','Smith',null);
```

11.7.2 Verify the Email Address

Issue a SELECT statement to verify that the trigger has fired:

```
SELECT * FROM T_EMP;
   ENO FNAME                M LNAME                EMAIL
   --- -------------------- - -------------------- --------
   100 John                 E Smith                jsmith
   200 James                A Smith                jasmith
```

11.8 Commit or Roll Back

Finally, commit your changes to preserve your work, or roll back to cancel changes.

11.9 Executing Java Stored Procedures from JDBC

After creating a Java stored procedures, you can execute the procedure from a JDBC application by performing one of the following:

- Pass a SQL SELECT string, which executes the stored procedure, to the Statement.executeQuery method.
- Use a JDBC CallableStatement.

The executeQuery method executes table-level and row-level stored procedures. CallableStatement currently only supports execution of table-level stored procedures.

11.9.1 Using the executeQuery Method

To call a stored procedure using the executeQuery method, perform the following:

1. Create a Statement object and assign the value returned by the createStatement method with the current connection object.
2. Execute the Statement.executeQuery method, passing the SQL SELECT string that invokes the Java stored procedure.

The following example executes a row-level procedure SHIP on a table named INVENTORY with the argument value stored in the variable *q*. The variable p contains the product ID for the product (row) for which you want to execute the stored procedure.

```
int res = 0;
Statement s = conn.createStatement();
ResultSet r = s.executeQuery("SELECT SHIP(" + q + ")" +
   "FROM INVENTORY WHERE PID = " + p);
if(r.next()) res = r.getInt(1);
r.close();
```

```
s.close();
return res;
```

If you need to execute a procedure repeatedly with varying parameters, use `PreparedStatement` instead of `Statement`. Because the SQL statements in a `PreparedStatement` are pre-compiled, a `PreparedStatement` executes more efficiently. Additionally, a `PreparedStatement` can accept `IN` parameters, represented in the statement with a question mark `(?)`. However, if the `PreparedStatement` takes a `long` type parameter, such as `LONG` or `LONG RAW`, you must bind the parameter using the `setAsciiStream`, `setUnicodeStream`, or `setBinaryStream` methods.

In the preceding example, if the `SHIP` procedure updates the database and the isolation of the transaction that issues the above query is `READ COMMITTED`, then you must append the `FOR UPDATE` clause to the `SELECT` statement, as follows:

```
"SELECT SHIP(" + q + ")" +
   FROM INVENTORY WHERE PID = " +
   p + "FOR UPDATE");
```

11.9.2 Using a Callable Statement

To execute the stored procedure using a callable statement, create a `CallableStatement` object and register its parameters, as follows:

```
CallableStatement cstmt = conn.prepareCall(
   "{?=call tablename.methodname() }");
cstmt.registerOutParameter(1, ...);
cstmt.executeUpdate();
cstmt.get..(1);
cstmt.close();
```

The following restrictions apply to JDBC callable statements:

- JDBC callable statements can only execute table-level stored procedures.
- Both IN and OUT parameters are supported. However, not all Java datatypes can be used as OUT parameters. For more information, see Section 11.4, "Java Datatypes".
- Procedure names correspond to the Java method names, and are case-sensitive.
- As with prepared statements, if the callable statement has a "`long`" type, such as: `LONG`, `LONG VARBINARY`, `LONG VARCHAR`, `LONG VARCHAR2`, or `LONG RAW`, you must bind the parameter using the `setAsciiStream`, `setUnicodeStream`, or `setBinaryStream` methods.

Note: When no longer needed, you should reclaim system resources by closing JDBC objects, such as `Resultset` and `Statement` objects.

12 Using Simple Object Data Access (SODA) for PalmOS and PocketPC Platforms

SODA is an interface used for Oracle Database Lite C++ development that provides object-oriented data access using method calls, relational access using SQL and object-relational mapping to bridge the gap between the two. Object functionality is roughly three times faster than ODBC for simple operations. It enables rich datatypes—such as arrays and object pointers—as well as standard SQL columns. A programmer can store any data structure in the database and not think about relational design or performing joins.

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

Object-relational mapping enables the application to access relational data as if it was object hierarchy. Thus, your application can replicate rich data types or object pointers to the Oracle database server.

Oracle Database Lite includes SODA Forms, which is a library that simplifies the development of GUI applications for Palm and PocketPC devices. See Section 12.6.2, "Develop Your GUI Using the SODA Forms Library" for more details.

The SODA API method calls are documented in the *SODA: Simple Object Data Access API Reference*, which is located off the `<ORACLE_HOME>/Mobile/index.htm` page. The full sample code that is demonstrated in this chapter is located in the `<ORACLE_HOME>/Mobile/doc/soda/sodadoc/html/sodasimple_8cpp-source.html` file.

- Section 12.1, "Getting Started With SODA"
- Section 12.2, "Using SQL Queries in SODA Code for PocketPC Platforms"
- Section 12.3, "Virtual Columns and Object-Relational Mapping"
- Section 12.4, "Behavior of Reference-Counted and Copy-By-Assignment Objects"
- Section 12.5, "Another Library for Exceptions (ALE)"
- Section 12.6, "Building a SODA Forms Application"
- Section 12.7, "SODA Forms Edit Modes"
- Section 12.8, "Customizing Your SODA Forms Application"
- Section 12.9, "Displaying a List Of Objects in a Table"
- Section 12.10, "SODA Forms UI Controls"

- Section 12.11, "OKAPI API"

12.1 Getting Started With SODA

In order to get started with SODA quickly, the following sections discuss the most frequently used classes:

- Section 12.1.1, "Overview of the SODA Classes"
- Section 12.1.2, "Demonstrating Frequently-Used SODA Classes"

12.1.1 Overview of the SODA Classes

When developing your C++ application, you would use the following classes the most:

- `DBSession` connects to the database and find and create classes.
- `DBClass` creates new database objects.
- `DBObject` modifies existing objects.
- `DBData` wraps an attribute value and is used for type conversion.
- `DBQueryExpr` builds single-table queries supported by SODA.
- `DBString (olString)` is a C string wrapper used by SODA.
- `DBList (olList)` is a template to store lists of values.
- `DBColList` and `DBDataList` instantiate these objects.

For example, implement the `DBObject` method, as follows:

```
DBObject obj;
...
obj["NAME"] = "Jack"
```

For full documentation for the SODA API method calls, see the *SODA: Simple Object Data Access API Reference*, which is located off the <*ORACLE_HOME*>`/Mobile/index.htm` page.

12.1.2 Demonstrating Frequently-Used SODA Classes

The following example demonstrates most of the SODA object-oriented functionality, as discussed in Section 12.1.1, "Overview of the SODA Classes".

```
void helloSODA() {
  puts("Hello SODA");
  try {
   DBSession sess("POLITE"); // Connect to the DSN, creating it if necessary

   // Find or create our class
   DBClass cls;
   try {
     cls = sess["PEOPLE"];
   } catch(DBException e) {
   cls = sess.createClass("PEOPLE", DBAttrList() <<
   DBAttr("ID", DB_INT) << DBAttr("NAME", DB_STRING));
  }
  // Create several objects. We can identify columns by name or positions
  DBObject o = cls.create("ID", 10, "NAME", "Alice");
  // The previous syntax of col1, val1, col2, val2 ... works for up to
// 32 columns. This version works for any number of columns
   cls.create(DBSetList() << "ID" << 10 << "NAME" << "Alice");
   cls.create(0, 20, 1, "Bob");

   // Note the automatic type conversion
   cls.create("ID", "314", 1, 3.14159265358);

   // Execute a query (will return two objects)
   DBCursor c = cls.createCursor(DBColumn("ID") == 10 ||
          DBColumn("NAME") == "Bob");

  DBObject ob;

  while (ob = ++c) {
   DBString s = ob["NAME"];
   puts(s);
   o["ID"] = (int)o["ID"]+1;
  }

  // Delete an object
  o.remove();

  // Clean up so that create class is successful next time
  sess.rollback();
 } catch(DBException e) {
 DBString s = e.getMessage();
 printf("Error: %s\n", (const char *)s);
 }
}
```

12.2 Using SQL Queries in SODA Code for PocketPC Platforms

To add SQL queries to your SODA code for PocketPC platforms, do the following:

1. Include `sodasql.h`, instead of `soda.h`.
2. Link your program with `sodasql.lib`.
3. Install `sodasql.dll` at runtime.
4. Create a `DBSqlSession` object instead of the `DBSession` object. Execute relational queries and other SQL statements with the help of `DBSqlStmt` and `DBSqlCursor` classes. Query results can be returned either as column values or as `DBObjects` for matching rows.

The following is a sample that uses the SODA relational interface:

```
void helloSQL() {
 try {
  puts("Hello SQL");
  DBSqlSession sess("POLITE");
  sess.execute("create table odtest(c1 int, c2 varchar(80))");
  DBSqlStmt stmt = sess.prepare("insert into odtest values(?,?)");

  // The values stand in for two ?'s in the statement above
  stmt.execute(5, "John");
  stmt.execute(10, "Mike");
  stmt.execute(15, "Alice");

  // Execute a single-table query that will return DBObject's for matching
  // rows. Use a standard SODA interface to access the objects
  DBSqlCursor c = sess.execute("odtest", "c1 < 15");
  while (++c) {
    DBObject o = c.getObject();
    DBString s = o["c2"];
    int val = o["c1"];
    printf("%d %s\n", val, (const char *)s);
    o["c1"] = val+1; // Can modify objects in addition to just reading them
  }

  // Execute a usual relational query
  c = sess.execute("select * from odtest");
  while (++c) {
    DBString s = c["c2"];
    printf("%d %s\n", (int)c["c1"], (const char *)s);
  }
 } catch(DBException e) {
 DBString s = e.getMessage();
 printf("Error: %s\n", (const char *)s);
 }
}
```

12.3 Virtual Columns and Object-Relational Mapping

A programmer does not view the data as column values that are stored in the database. For example, a master-detail relationship might be expressed as matching values in two tables, but for a program it is more natural to access a column in the master object, which contains an array of pointers to details.

The `DBVirtualCol` class enables the translation between the conceptual view of the data and the actual data in the tables. You can create a column that is completely under programmer's control through the `get`, `set` and `remove` methods and adding it to a class at runtime.

In fact, SODA contains a specialized `DBValueRel` class that extends the `DBVirtualCol` class to map master-detail relationships to object pointers. The following sample builds a binary search tree in the database using object-relational mapping:

```
struct HelloVirtual {
 int lpos, rpos, vpos;
 void visit(DBObject o);
 HelloVirtual();
};

void HelloVirtual :: visit(DBObject o) {
 while (o) {
 visit(o[lpos]);
 printf("%d\n", (int)o[vpos]);
 o = o[rpos];
 }
}

HelloVirtual :: HelloVirtual() {
 try {
   DBSession sess("POLITE");
   // TreeNode represent a binary tree with left and right pointers
   // that point back to parent's id column
   DBClass cls = sess.createClass("TreeNode",
   DBAttrList() << DBAttr("val", DB_INT) << DBAttr("id", DB_INT)
     << DBAttr("lchild", DB_INT) << DBAttr("rchild", DB_INT));
```

```
    // Create an index on id to speed up search
    cls.createIndex("i1", DBColList() << "id", true);
    // Set a sequence as a default value of id, so that we don't have to set it
    // explicitely
    DBSequence seq = sess.createSequence("s1");
    cls.defaultVal("id", seq);
    // Create the virtual columns
    DBValueRel lref("left", DBSrcCol(cls, "lchild") -> DBDstCol(cls, "id"),
        DB_UPD_DETAIL);
    DBValueRel rref("right", DBSrcCol(cls, "rchild") -> DBDstCol(cls, "id"),
        DB_UPD_DETAIL);
    // Cache column positions for frequent access
    lpos = cls["left"], rpos = cls["right"], vpos = cls["val"];
    // Root of the binary tree
    DBObject root;
    // Insert some random numbers into our binary search tree
    for (int i = 0; i < 50; i++) {
     int v = rand();
     DBObject o = cls.create(vpos, v); // Note automatically generated ids
     DBObject par; int dir;
     for(DBObject cur = root; cur; par = cur, cur = cur[dir])
     dir = (int)cur[vpos] >= v ? lpos : rpos;
     if (par) par[dir] = o; else root = o;
    }
    // Do in-order traversal of the tree, printing out numbers in sorted order
    visit(root);
  } catch(DBException e) {
  DBString s = e.getMessage();
  puts(s);
 }
}
```

12.4 Behavior of Reference-Counted and Copy-By-Assignment Objects

Most C++ classes in SODA are reference-counted, which means that assigning one variable of one type to another cannot copy an object, but creates another way to refer to the same object. For example, the `DBData` class represents values that can be stored in persistent objects.

The following example demonstrates reference-counting:

```
DBData d = 5; // Create a new object containing value 5
                 // and make a reference to it
DBData d2 = d; // Both reference the same object
d << 20; // Add another value to existing object.
         // Both d and d2 reference the new array
d2 = 10; // d references the array, d2 is reassigned to the new data.
d.clear(); // Clear the last reference to the array of 5 and 20
           //and free the array
```

The programmer does not need to free objects when they are no longer used. However, this method is relatively expensive and not practical for objects that are created and destroyed often, such as when new lists of values, such as `DBSetList`, are allocated for each SODA call. Various lists in SODA, such as `DBSetList`, `DBDataList` and so on, are copy-on-assignment rather than reference-counted objects.

The following example demonstrates copy-by-assignment:

```
DBSetList ls << "cost" << 1000;
DBSetList ls2 = ls; // Created a copy of ls
ls << "value" << "priceless" // Only ls is changed
ls2.clear(); // Just set this copy to size 0
```

Note: SODA includes non-database classes and templates for your convenience, which have names that start with `ol` rather than `DB`—such as `olHash`. Avoid making unnecessary copies of these classes.

You can optimize your implementation by not creating an unnecessary copy by passing a reference or a `const` reference to the object, rather than an object, for both reference-counted and copy-on-assignment classes when calling a function. For example, `void func(const DBData &v)` avoids creating an unnecessary copy.

Note: The `clear` method only nullifies a particular reference to reference-counted objects. `DBSession` and `DBCursor` classes provide a `close` method that releases the underlying database resources, even while the objects are still referenced. Using anything that relies on a closed cursor or database connection throws a `DBException`.

12.5 Another Library for Exceptions (ALE)

Many embedded compilers, such as Visual C++ for PocketPC, do not support C++ exceptions. Oracle Database Lite includes ALE, which is a library that closely mimics C++ exceptions. The following sections describe how to use ALE:

- Section 12.5.1, "Decorating Classes With ALE"
- Section 12.5.2, "New Operator and ALE"
- Section 12.5.3, "Global Variables"
- Section 12.5.6, "Troubleshooting ALE Runtime Errors"
- Section 12.5.7, "Compiling Your Program With ALE"
- Section 12.5.8, "ALE Code on Systems That Support Exceptions"

12.5.1 Decorating Classes With ALE

A decorated class is one that uses ALE for handling its exceptions. If your embedded compiler does not support exception handling, then use ALE, which relies on careful accounting of all objects that are already constructed or are being constructed. ALE supports stack unwinding and catching exceptions based on object type.

To use ALE, C++ source code needs to be modified where the try, catch and throw blocks are replaced with the ALE macros, which is known as decorating classes. The following is an example of a decorated class:

```
#include "ale.h"
struct Error {
    const char *msg;
    Error(const char *msg) :msg(msg) {}
};

aleTry {
    olArray<char> a(5);
    aleThrow(Error,Error("Adios"));// Or aleThrowObj(Error,("Adios"));
} aleCatch(Error,e) {
    puts(e.msg);
} aleCatch(aleBadAlloc,e) {
    puts("Tough!");
} aleCatchAll {
    puts("Some other exception happened\n");
    aleReThrow;
} aleEnd;
```

Table 12–1 ALE Macros for C++ Exceptions

Macro	Action
`aleTry`	Equivalent to C++ try. Use to enclose the code that might encounter any exception.
`aleCatch (type, varName)`	Catch exception of a given type and store it in the variable `varName`, which is local to the block. Unlike the regular C++ exceptions, the `type` name string must match the argument of the throw exactly.
`aleThrow (type, obj)`	Throw an exception contained in `obj` of `type`. ALE supports single inheritence of exceptions, as described in Section 12.5.4, "Exceptions and Inheritance".
`aleThrowObj (type, arg1, arg2, ....)`	Construct a new object of `type` with the specified arguments and throw it as an exception.
`aleCatchAll`	Catch any exceptions that are not handled explicitly.
`aleReThrow`	Rethrow the exception that is caught in the innermost `aleCatch` or `aleCatchAll` block.
`aleEnd`	Close the exception handling construct. Add a semi-colon `;` after `aleEnd`.

If your embedded compiler does not support exception handling, then use ALE, which relies on careful accounting of all objects that are already constructed or are being constructed. Your class is involved in exception handling if the class is on the stack when an exception is thrown, its constructor may throw an exception, or it has a decorated superclass and member—even if the class does not do any additional exception-related processing.

When you decorate one class with ALE, you must decorate all classes involved with this class. If you omit a decoration in one of the classes, then the program might fail. Therefore, it is best to decorate all your classes except for plain C-style structures that do not have any constructors or destructors.

If your class does not have any constructors with a body that throws exceptions (it is OK if the superclass or member constructor does), then you can use a simple form of class decoration by adding `ALELAST(ClassName)`, without a semicolumn, at the end of class declaration. `ALELAST` informs the library to register the class and clean-up if an error occurs. The following demonstrates how to use `ALELAST`:

```
template<class T> class PtrHolder {
   T *ptr;
public:
   ~PtrHolder() { delete ptr; }
   operator T *() { return ptr; }
   ALELAST(PtrHolder)
};
```

If any of the constructors throw exceptions, then you need to do the following:

1. Add `ALECLAST(ClassName)`, rather than `ALELAST` to the end of the class body.
2. Add `ALECONS(ClassName)` in the beginning of the body of each constructor.

This is demonstrated, as follows:

```
template<class T> class Array {
    T *a;
    size_t len;
public:
    Array(size_t len=0} : len(len) {
        ALECONS(Array);
        a = new T[len];
    }
    Array(const Array &arr) : len(arr.len) {
        ALECONS(Array);
        for (size_t i = 0; i < len; i++)
            a[i] = arr.a[i];
    }
    ...
    ALECLAST(Array)
};
```

In this example, the class contains an explicit copy constructor. If your class does not contain an explicit copy constructor and instances can be copied, then you need to explicitly write a copy constructor and add `ALECONS(ClassName);` rather than using a copy constructor that is generated by the compiler. If you need a more complicated initialization than a default or copy constructor, then use a global pointer—which can be initialized by another global object, rather than a global instance.

12.5.2 New Operator and ALE

Decorated classes can be safely used with the `new` and `delete` functions, including using `new` and `delete` for `array` and `placement new`. However, systems that do not support exceptions usually do not declare `std::bad_alloc` and `std::no_throw` types. Use `aleBadAlloc` and `aleNoThrow` instead of using `std::bad_alloc` and `std::no_throw` types.

One design decision is whether to decorate classes with constructors that only throw `bad::alloc` if they run out of memory using `ALELAST` or `ALECLAST`. If you are writing classes for a single application that does not allocate much memory, you might dispense with error checking and just use `ALELAST`. Your program might crash because of incorrect cleanup calls if it runs out of memory. If your class allocates a lot of memory or you are writing a highly-reusable framework, then it is best to use `ALECLAST` and decorate all the constructors.

12.5.3 Global Variables

If you need a global or static variable to be decorated with ALE, declare it using `aleGlobal` template, as follows:

```
aleGlobal<MyType> myGlobal; // Initialized with default constructor
aleGlobal<MyType> myGlobal1(MyType("Hello", 5")); // Initialized with copy
constructor.
...
MyType *t = myGlobal; // Declared variables behave as pointers
```

Declare all global or static decorated instances using `aleGlobal` or you may receive runtime errors.

12.5.4 Exceptions and Inheritance

Unlike regular C++ exception handling, ALE requires that class names in `aleThrow` and `aleCatch` match exactly. Typedef names, throwing a subclass, and catching a superclass will not work. To build a hierarchy of exceptions, add `ALEPARENT` declaration to the subclass, as follows:

```
class BaseE {
    ALELAST(BaseE)
};
class DerivedE : public BaseE {
    ALELAST(DerivedE)
    ALEPARENT(BaseE)
};
```

`DerivedE` can be caught as `BaseE`. If you use multiple inheritance, then the first base class must be declared as a parent.

12.5.5 Using ALE with PocketPC ARM Compilers

The Microsoft Embedded Visual C++ for ARM has a bug that is triggered when an ALE-decorated object (or any object with embedded pointers) is passed by value to a function or method. The affected code receives an ALE fatal error message at runtime. To avoid this problem, always pass SODA objects and instances of other classes that use ALE as a constant reference rather than value. For example, modify the following code:

```
void createName(DBClass cls, DBString name) {
cls.create("name", name);
}
```

to the following implementation:

```
void createName(const DBClass &cls, const DBString &name) {
cls.create("name", name);
```

12.5.6 Troubleshooting ALE Runtime Errors

If your classes are not decorated properly, then you will receive runtime errors. On PocketPC, ALE displays a message box explaining the problem, and then the program terminates. In addition, the error and the dump of the ALE stack is appended to `aleDump.txt`, which exists in the root directory of the device. In simple cases, the error message pinpoints the exact problem; for example `ALECONS is missing for class MyArray`. Usually, one of the classes found near the top of the ALE stack is not decorated properly. If you do not decorate a class and its superclasses or members are decorated, then you may receive a runtime error and see the superclasses/members on the stack.

12.5.7 Compiling Your Program With ALE

To build a program that uses ALE, include `ale.h` from the Oracle Database Lite SDK and link with the `olStdDll.lib` library. You need `olStdDll.dll` at runtime.

12.5.8 ALE Code on Systems That Support Exceptions

For systems that already support C++ exceptions, like Win32, Oracle Lite includes a dummy `ale.h` that defines the same macros, but uses regular C++ exceptions to implement them. If you are writing code that must execute on both Win32 and

PocketPC, remember to test the code with the actual ALE library to ensure that all your classes are decorated correctly. `ALELAST`, `ALECLAST` or `ALECONS` have no effect on the Win32 platform.

12.6 Building a SODA Forms Application

The following sections describe how to create a SODA Forms Application for PalmOS and PocketPC platforms:

- Section 12.6.1, "Development Environment Requirements"
- Section 12.6.2, "Develop Your GUI Using the SODA Forms Library"
- Section 12.6.3, "Designing the UI for PocketPC"
- Section 12.6.4, "Customizing the Database Schema"
- Section 12.6.5, "Binding UI to Data"
- Section 12.6.6, "Setting List Choices for Status Contol on PocketPC"
- Section 12.6.7, "Customizing the Table in OrderForm"
- Section 12.6.8, "Monitoring the Logic"

12.6.1 Development Environment Requirements

SODA Forms relies on SODA, which is an easy-to-use C++ interface for the Oracle Lite database engine. Read SODA documentation before continuing with this manual. Make sure you understand objects, queries, cursors and virtual columns as a way to customize database schema for a particular application.

- When developing for the Palm environment, use a PalmOS programming book. Focus on how to use Constructor for PalmOS, which is a visual tool to design forms. Also, familiarize yourself on how to use Metrowerks CodeWarrior to compile programs and PalmOS Emulator to execute them. Refer to `FormOrders.mcp` and `build.html` in the Palm documentation for instructions on creating a CodeWarrior project that uses SODA Forms.

  Find and open `FormOrders.mcp` in the Oracle Database Lite PalmOS SDK. Execute the demo on your emulator or device with the Oracle Database Lite runtime installed and examine its behavior. When you are ready to continue, open the `FormOrders.cpp` and follow the code by reading this section.

- When developing for the PocketPC environment, refer to the Microsoft Development Network. Focus on how to use Embedded Visual C++ to create resources and compile programs for PocketPC. Also, familiarize yourself with Windows UI controls, including their formats and styles.

  If you want to use SQL in your application, understand the SODA SQL support. The `FormsOrder` demo uses SODA SQL to support custom queries.

  Find and open `FormOrders.vcp` in `SodamFormCE\FormOrders` directory under the samples for SODA on the PocketPC. This is the demo for PPC2003.

12.6.2 Develop Your GUI Using the SODA Forms Library

SODA Forms is a quick way to create data entry GUI.

12.6.2.1 Traditional Way to Develop Native Data Entry Applications

The GUI for data entry applications performs the same actions, as follows:

- Copy values from different columns of a database record into controls, such as text fields and checkboxes.
- Track what the user is doing by handling various UI events, like "field entered" or "button clicked". The user typically examines or edits the data on the screen and presses a button or picks a menu item to designate the next action.

To perform the action requested by the user, the application performs database and UI calls. For example, to save changes, the application retrieves the values in the UI controls and eventually executes a SQL update statement. An application needs to be able to discard changes, delete records, create new records, go to the previous or the next entry, and search for data that satisfies user-selected conditions.

Normally, you duplicate the UI code for each screen supported by the program, which can result in a bulky program. Instead, use Soda Forms, as described in Section 12.6.2.2, "Trimming Your PalmOS and PocketPC UI Code With SODA Forms", to streamline your UI code.

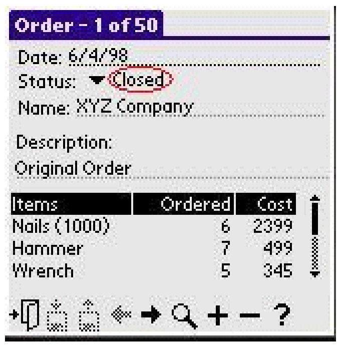
12.6.2.2 Trimming Your PalmOS and PocketPC UI Code With SODA Forms

We moved the boilerplate UI code to a library and to enable the programmer to concentrate on application logic, not the tasks of copying values from database rows to UI controls. Oracle Database Lite provides this library for both PalmOS and PocketPC.

The following example creates a form:

```
DBSession sess("OrdersODB"); // Open a database connection
DBClass cls = sess["ORD_MASTER"]; // Locate a table in the database
DBForm frm(OrderForm, cls, orderCols, OL_COUNTOF(orderCols)); // Initialize a form
frm.edit(cls.createCursor()); // Let the user edit all records in ORD_MASTER table
```

If using the PalmOS UI designer, then the form is created visually with the Constructor with values from the first row of `ORD_MASTER` table1, as follows:

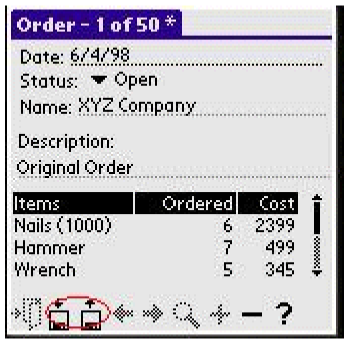


Order - 1 of 50

Date: 6/4/98
Status: Closed
Name: XYZ Company

Description:
Original Order

Items	Ordered	Cost
Nails (1000)	6	2399
Hammer	7	499
Wrench	5	345

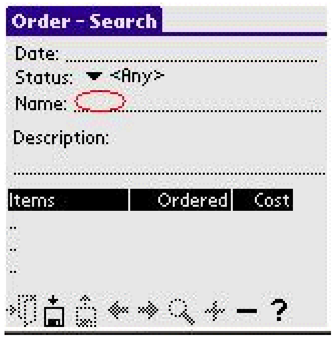


Order - 1 of 50 *

Date: 6/4/98
Status: Open
Name: XYZ Company

Description:
Original Order

Items	Ordered	Cost
Nails (1000)	6	2399
Hammer	7	499
Wrench	5	345


Order - Search

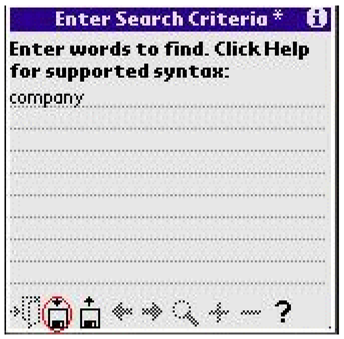
Date:

Status: <Any>

Name:

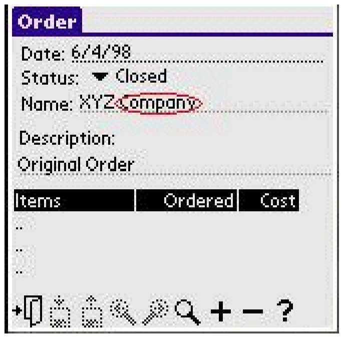
Description:

Items	Ordered	Cost


Enter Search Criteria *

Enter words to find. Click Help for supported syntax:

company


Order

Date: 6/4/98

Status: Closed

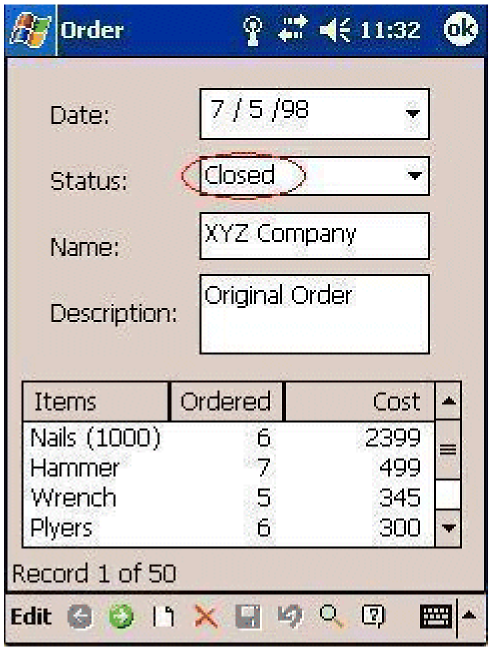
Name: XYZ company

Description:

Original Order

Items	Ordered	Cost

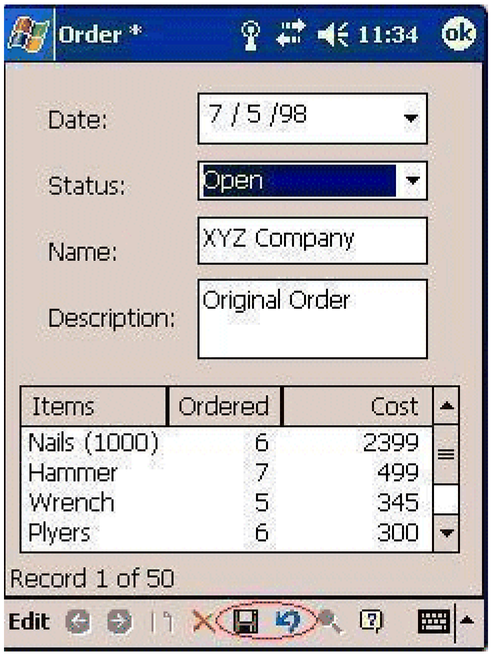
If designing an application for PocketPC, then the form is created with Embedded Visual C++ resource editor with values from the first row of `ORD_MASTER` table1, as follows:



Date:	7 / 5 /98
Status:	Closed
Name:	XYZ Company
Description:	Original Order

Items	Ordered	Cost
Nails (1000)	6	2399
Hammer	7	499
Wrench	5	345
Plyers	6	300

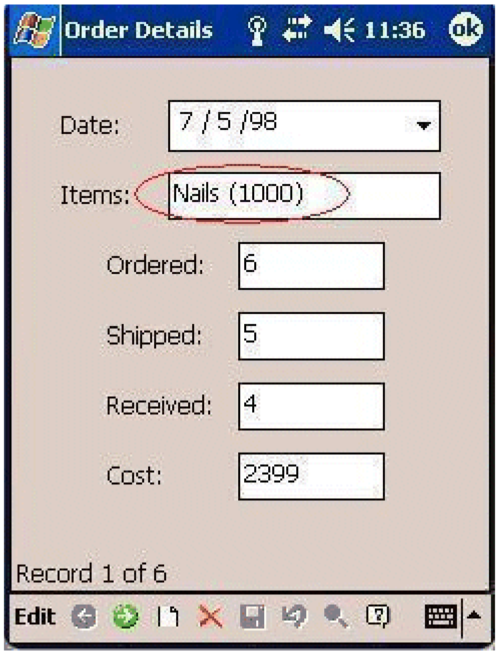
Record 1 of 50



Date:	7 / 5 /98
Status:	Open
Name:	XYZ Company
Description:	Original Order

Items	Ordered	Cost
Nails (1000)	6	2399
Hammer	7	499
Wrench	5	345
Plyers	6	300

Record 1 of 50



Date: 7 / 5 /98

Items: Nails (1000)

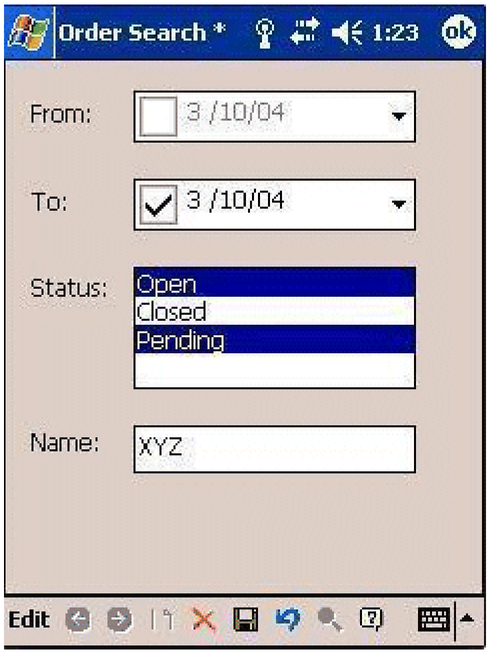
Ordered: 6

Shipped: 5

Received: 4

Cost: 2399

Record 1 of 6



From: [ ] 3 /10/04

To: [✓] 3 /10/04

Status: [✓ Open] Closed [✓ Pending]

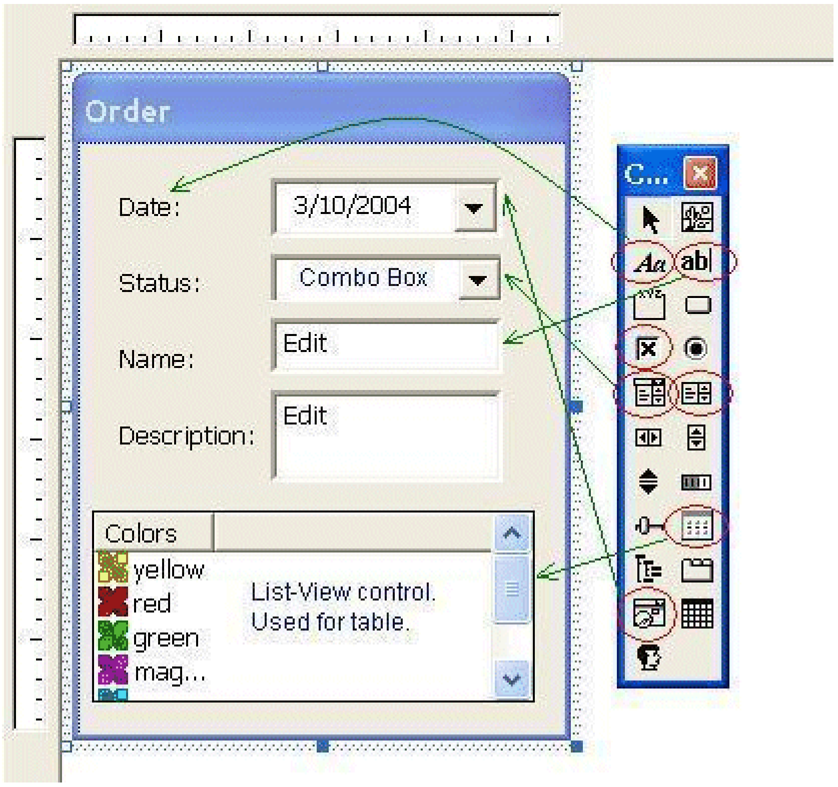
Name: XYZ

The user can update the current record and then either save the changes or discard them and reload from the database. He or she can also search data using any values visible on the screen, scroll through all the records as well as delete existing entries or create new ones. No special code is needed to support these operations.

On PocketPC, you can also see a lens button on the toolbar. When this button is clicked, the application can launch a cusomtized dialog for entering search conditions, retrieve their values, an dexecute custom queries. For PocketPC, we only support queries implemented by the application. Whereas, PalmOS has built-in queries.

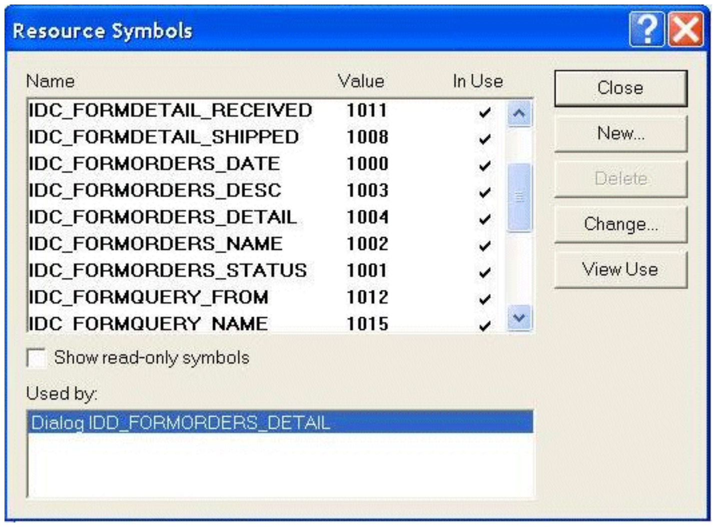
12.6.3 Designing the UI for PocketPC

For the PocketPC platform, a SODA form is a Windows dialog. Thus, the first step is to design dialogs for every form using the Embedded Visual C++ resource editor. In the `FormOrders` project, click on the resource tab, open the dialog folder and click on the `IDD_FORMORDERS_MASTER` dialog. This shows the dialog for the master form, as follows:



The controls on the master form, labels (also called static controls, such as "Date:", "Status:", and so on), edit controls (name and description fields), Date-Time picker (for date field), ComboBox (for status field), and list view control for the detail table. There are corresponding icons on the toolbox used to create these controls.

Each control has separate formats and styles that can be customized through the resource editor to find the appearance and behavior you want. For every control there is a resource symbol with numeric id assigned to it. Right-click on the `FormOrders.rc` and select `Resource Symbols`, as follows:



Name	Value	In Use
IDC_FORMDETAIL_RECEIVED	1011	✓
IDC_FORMDETAIL_SHIPPED	1008	✓
IDC_FORMORDERS_DATE	1000	✓
IDC_FORMORDERS_DESC	1003	✓
IDC_FORMORDERS_DETAIL	1004	✓
IDC_FORMORDERS_NAME	1002	✓
IDC_FORMORDERS_STATUS	1001	✓
IDC_FORMQUERY_FROM	1012	✓
IDC_FORMQUERY_NAME	1015	✓

Used by:
Dialog IDD_FORMORDERS_DETAIL

You can either create your own identifiers or let the resource editor create them for each symbol. Look at the header file, `resource.h`, which is generated by the resource editor, to see how identifiers are assigned to resource symbols using `#define` statements.

12.6.4 Customizing the Database Schema

The following SODA code connects the application to an Oracle Lite database and retrieves the `DBClass` objects for two tables used in the demo—`ORD_MASTER` and `ORD_DETAIL`:

```
DBSession sess("OrdersODB"); // Open a database connection
DBClass mCls = sess["ORD_MASTER"];
DBClass dCls = sess["ORD_DETAIL"];
```

However, values stored in the database do not exactly match what should be shown to the user. SODA allows each application to customize how the schema is shown. Both the `ORD_MASTER` and `ORD_DETAIL` tables have ID columns, which are primary keys and should be initialized to some unique value when a new record is created. To avoid asking the user to pick unique values, set the default value of each column to a sequence, as follows:

```
mCls.defaultVal("ID", sess.findSequence("OrderSeq"));
dCls.defaultVal("ID", sess.findSequence("DetailSeq"));
```

In addition, the UI shows the order status as "Open", "Closed" or "Pending"; however, the "STATUS" column in `ORD_MASTER` stores numbers, such as 0, 1 or 2. The following line creates a virtual column named `strStatus` that contains a mapped value:

```
DBMapCol mc("strStatus", mCls, "STATUS", DBDataList() << 0 << "Open" << 1 <<
"Closed" << 2 << "Pending" << DBNULL << DBNULL);
```

Finally, modify the Order screen to contain a table of items included in the order. SODA Forms populates a table by reading a column that stores a list of `DBObject` pointers and copying values from each object to a table row. In our case, `ORD_DETAIL` rows relate to a row in `ORD_MASTER` through the values of the `KEY` column, which matches the value of the master ID. The following declaration creates a pseudo-column that contains an array of pointers and updates `KEY` values as assigned:

```
DBValueRel rDet("detail", DBSrcCol(mCls, "ID") -> DBDstCol(dCls, "KEY"),  DB_UPD_
CASCADE);
```

If a mapping you are looking for is not part of SODA, then you can create your own by subclassing the `DBVirtualCol` class. Default values and virtual columns are transient and not stored in the database. Thus, declare them in every application that requires access to your abstractions.

12.6.5 Binding UI to Data

Once you complete customizing the database schema, you can map columns in the database to values that appear on the user screen when editing a particular record. The following sections describe how to map the columns for both PalmOS and PocketPC:

- Section 12.6.5.1, "Binding UI to Data for the PalmOS Environment"
- Section 12.6.5.2, "Binding UI to Data for the PocketPC Environment"

12.6.5.1 Binding UI to Data for the PalmOS Environment

For each column that you are going to map, you need to define a `DBFormCols` structure that has the following fields.

- The resource id assigned to a particular UI control by the PalmOS Constructor. The Constructor generates a header file that has a `#define` for each item. For example, if you define a list named `Status` in a form named `Orders`, then `OrderStatusList` is defined to the ID of that control.
- A column name to which the UI control is to be bound.
- An optional field that specifies how the control should be edited. For example, you can specify that a text field should be edited as a date, number or string. If omitted, the default method is used.

Here is the mapping for our two forms:

```
//Mapping for ORD_MASTER table
static const DBFormCols orderCols[] = {
    {OrderDateField, "DDATE", DBFormEditDate},
    {OrderNameField, "NAME"},
    {OrderDescField, "DESCRIPTION"},
    {OrderStatusList, "strStatus"},
    {OrderDetailTable, "detail"}
};
//Mapping for ORD_DETAIL table
static const DBFormCols detailCols[] = {
    { DetailDateField, "DDATE", DBFormEditDate },
    { DetailItemsField, "DESCRIPTION" },
    { DetailOrderedField, "QTYORDERED", DBFormEditDigits },
    { DetailShippedField, "QTYSHIPPED", DBFormEditDigits },
    { DetailRecievedField, "QTYRECEIVED", DBFormEditDigits },
    { DetailCostField, "COST", DBFormEditDigits }
};
```

Once the mapping is in place, create `DBForm` objects by specifying the resource ID of the form itself, `DBClass` of objects that the form will be used to edit and the mapping table with the number of elements it contains:

```
DBForm mFrm(OrderForm, mCls, orderCols, OL_COUNTOF(orderCols));
DBForm dFrm(DetailForm, dCls, detailCols, OL_COUNTOF(detailCols));
```

12.6.5.2 Binding UI to Data for the PocketPC Environment

For each column, you need to define a `DBFormCols` structure that has the following fields.

- The resource id assigned to a particular UI control.
- The column name to which the UI control will be bound.
- The optional argument that specifies how the control should be edited. For PocketPC, the only relevant value is `DBFormEditListIndex`, which asks to use the index of the selection, rather than its string value, for `ListBox` and `ComboBox` controls. If omitted, the default method is used.

Here is the mapping for our two forms:

```
//Mapping for ORD_MASTER table
static const DBFormCols orderCols[] = {
{IDC_FORMORDERS_DATE, "DDATE", DBFormEditDate},
{IDC_FORMORDERS_NAME, "NAME"},
{IDC_FORMORDERS_DESC, "DESCRIPTION"},
{IDC_FORMORDERS_STATUS, "strStatus"},
{IDC_FORMORDERS_DETAIL, "detail"}
};
//Mapping for ORD_DETAIL table
static const DBFormCols detailCols[] = {
    { IDC_FORMDETAIL_DATE, "DDATE", DBFormEditDate },
    { IDC_FORMDETAIL_ITEMS, "DESCRIPTION" },
    { IDC_FORMDETAIL_ORDERED, "QTYORDERED", DBFormEditDigits },
    { IDC_FORMDETAIL_SHIPPED, "QTYSHIPPED", DBFormEditDigits },
    { IDC_FORMDETAIL_RECEIVED, "QTYRECEIVED", DBFormEditDigits },
    { IDC_FORMDETAIL_COST, "COST", DBFormEditDigits }
};
```

Once the mapping is in place, create the `DBForm` objects by specifying the resource ID of the form itself, the `DBClass` of objects that the form will be used to edit and the mapping table with the number of elements it contains, as follows:

```
DBForm mFrm(OrderForm, mCls, orderCols, OL_COUNTOF(orderCols));
DBForm dFrm(DetailForm, dCls, detailCols, OL_COUNTOF(detailCols));
```

12.6.6 Setting List Choices for Status Contol on PocketPC

For listbox and ComboBox controls in the PocketPC environment, set the string list of choices. You cannot set the list choices for the listbox in the resource editor, and it can be difficult to set for the combobox. Thus, set the list choices for both the listbox and the combobox programmatically through the `setListItems` function of `DBFormCol`, as follows:

```
DBList<DBString> stList;
stList << "Open" << "Closed" << "Pending";
mFrm[IDC_FORMORDERS_STATUS].setListItems(stList);
```

12.6.7 Customizing the Table in OrderForm

Specify which columns of the `ORD_DETAIL` table appear in the table and how the values are aligned within a cell and in the title of each column. The following array contains the information:

```
static const DBFormTblCols detailTblCols[] = {
    { "DESCRIPTION", "Items" },
    { "QTYORDERED", "Ordered", DBFormColRight },
    { "COST", "Cost", DBFormColRight }
};
```

Retrieve a handle to the table column and set its format, as follows:

```
DBFormCol tCol = mFrm[OrderDetailTable];
tCol.setTableInfo(dCls, detailTblCols, OL_COUNTOF(detailTblCols));
```

12.6.8 Monitoring the Logic

The following few lines of code constitute the logic of `FormOrders` application:

```
// Load all objects of ORD_MASTER into a form
DBCursor c = mCls.createCursor();
mFrm.load(c);
// Handle table select event in master form to launch the detail form
while (mFrm.edit() == OrderDetailTable)
        dFrm.edit(mFrm[OrderDetailTable]);
```

From this, the following occurs:

1. A list of objects from `ORD_MASTER` to edit is loaded into the order form.
2. The `mFrm.edit` function displays the UI and handles many of the user actions internally. Before the call returns, the user could have made changes, created new records, searched through the data, and so on.
3. The return values from the function call are an identifier of the UI control that was activated, which was not handled internally. In the case of the Orders demo, it would either be the `OrderDetailTable`—meaning that a row in that table was clicked—or the `DBFormItemExit`—if the exit icon on the toolbar was clicked.
4. If the user clicked the table, the demo asks the `DBForm` detail to edit the list of objects contained in the table.
5. Once the edit is completed, any pointers to new objects are saved in the `detail` virtual column of the `ORD_MASTER` row. The `DBMapCol` updates the `KEY` column of the objects to the identifier value that matches the master.

You can use SODA Forms to write concise UI code without replicating a boilerplate. The next sections explore more advanced capabilities of the library.

12.6.9 Compiling Your SODA Application

SODA can be compiled on Windows, PocketPC and PalmOS environments. Refer to the PalmOS documentation for instructions on building applications for that platform.

To build a SODA application on Win32, perform the following:

1. Add the Oracle Database Lite SDK to the include and library path in your compiler options.
2. Include the appropriate `.h` files and link with appropriate library files, as follows:
   - For most applications, include `<soda.h>` and link with `olStdDll.lib` and `sodadll.lib`.
   - If you are using Visual Studio.Net 2003 and you are building an application using SODA SQL binding, include the `<sodasql.h>` file instead of `soda.h` and link with `olStdDll.lib`, `sodadll.lib`, and `sodasql.lib`.
   - If you are using Visual C++ 6.0 and building and application using SODA SQL binding, then link with `olStdDll6.lib`, `sodadll6.lib`, and

`sodasql6.lib` libraries. The `olStdDll` library contains utility classes that are not related to database, such as `olString` and `olHash`. It does not depend on the rest of Oracle Database Lite runtime, except `ceansi.dll`, on PocketPC platforms.

- To build a SODA Forms application for PocketPC platform, include `SodaForm.h` into your program and link with the following import libraries: `sodadll.lib` (soda library), `sodasql.lib` (library for soda sql) and `sodaform.lib` (sodaform runtime). Also, install the Oracle Database Lite runtime and `sodadll.cab` in order to run your application. `Sodadll.cab` contains `sodadll.dll`, `sodasql.dll`, `sodaform.dll` and `SodaFormHelp.html` (default help file for SODA Forms). Currently, we support SODA Forms for two PocketPC platforms: Pocket PC 2002 and Pocket PC 2003. The soda libraries and `sodadll.cab` files are provided in the SDK for each platform—both for the device and the emulator.

SODA includes a software emulation library that requires some changes in syntax when using C++ exceptions, but keeps the program structure intact. See Section 12.5, "Another Library for Exceptions (ALE)" on how to support C++ exceptions.

12.7 SODA Forms Edit Modes

The following sections describe the SODA Forms edit modes:

- Section 12.7.1, "Editing a Single Object"
- Section 12.7.2, "Editing a List of Objects"
- Section 12.7.3, "Creating a New Object"
- Section 12.7.4, "Popping Up A Dialog"

12.7.1 Editing a Single Object

The code below enables a user to edit a specific object. To edit a list of objects, see Section 12.7.2, "Editing a List of Objects":

```
DBObject o;
DBFormItem id = frm.edit(o);
```

In this case, toolbar will not have arrows to scroll or "new" icon to create a new object. Query is disabled. If the user saves the changes or deletes the object, `DBFormItemSave` and `DBFormItemDelete` are returned respectively. With list edit, save or delete do not return, since the user can still make additional changes to another record.

12.7.2 Editing a List of Objects

The following example loads a list of objects into a form, and enables the user to insert, delete, update or query:

```
olList<DBObject> ls;
ls << obj1 << obj2 << obj3;
DBFormItem id = frm.edit(ls);
```

When the edit returns, the `DBObject` list is updated with the new list of objects that reflects creation and deletion. The `DBFormItem` variable has one of the following values:

Table 12–2 DBFormItem Identifier Values

Value	Explanation
DBFormItemExit	User clicked one of the following: ■ on PalmOS, the exit icon on the toolbar ■ on PocketPC, an OK button on the navigation bar
Identifier of a table column	User clicked on a row in a table. Use the frm[id].getSelectedRowfunction to determine which one it is.
Identifier of a button	User clicked on a button in the form.
Identifier of a menu item	User clicked on a menu item not handled internally by SODA forms.
Identifier of a pop-up list or checkbox	The UI control was changed and you called frm[resId].setChangeNotify(true) on that column.
DBFormItemRevert	You set DBFormRevertExit edit flag and the user reverted a change.

There are several ways to customize the editing. The following example demonstrates a customization:

```
DBCursor cur = cls.createCursor(DBColumn("status") == "Active");
frm.load(cur, DBFormNew|DBFormUpdate|DBFormDirtyExit, DBSetList() << "zip" <<
94403 << "status" << "Active");

DBFormItem id;
while ((id = frm.edit()) != DBFormItemExit)
    if (id == CustomerBelmontButton) {
        frm[CustomerZipField] = 94002;
        frm.dirty();
    }
olList<DBObject> ls;
frm.getList(ls);
```

The `DBFormNew` or `DBFormUpdate` edit modes are specified instead of the default `DBFormListEdit` mode, which disallows the deletion of records while creating and updates are OK. `DBFormReadOnly` enables users to view and search the data.

The `DBFormDirtyExit` flag means that a button press or menu selection would exit edit even if the form is dirty. The default is to beep and wait until the user saves or reverts the change.

Specifying a `DBSetList` provides initial values that are given when a new object is created. For bound, enabled columns, the user can change that value. For unbound database columns or columns bound to read-only UI controls, this is the final value.

You can specify objects to edit in two ways – by giving an explicit list of objects or by providing a `DBCursor`. In the later case, the results of a query are loaded into the form.

In this case, load the list of objects once, call edit one or more times, and then retrieve the final edited list. SODA Forms provides you a choice between calling the edit method with all the arguments or calling load, edit without arguments and then `getList` or `getObject` functions. Examples in this document only show one possibility and do not discuss each edit flag.

12.7.3 Creating a New Object

The example below loads initial values into a form and asks the user to modify the values. The user then either clicks **Save** to create a new record or clicks **Delete** to cancel the creation.

```
DBObject o = frm.create(DBSetList() << "zip" << 94403 << "status" << "Active");
```

This call returns the new object or `DBNULL`, if the creation was canceled. However, if you use the load/edit sequence, then call the `getObject` method to retrieve the object handle, if `DBFormItemSave` is returned by the `edit` method.

12.7.4 Popping Up A Dialog

Although SODA Forms is designed for editing database records, you can use the same interface to retrieve input from the user. The following example enables the user to edit two fields and then retrieve the result:

```
DBForm frm(CustomerForm);
frm[CustomerZipField] = 94002;
frm[CustomerStatusField] = "Active";
DBFormItem id = frm.dialog();
if (id == DBFormItemSave) {
       DBString status = frm[CustomerStatusField];
       int zip = frm[CustomerZipField];
       ...
}
```

The values set before the `frm.dialog` call are default values. The user can make changes and then click **Revert** to restore the defaults and try again. Finally, the user clicks **Save** to send the changes to the program.

12.7.5 Custom Queries for PocketPC Environment

If you want to search on the list of objects loaded into a form, you can perform a custom query, which is the query logic implemented by the application. Execute the `FormOrders` demo and click on the lens toolbar button on the main form. It pops up a form—a dialog—to enter the search parameters for a custom query. Examine the `IDD_FORMORDERS_QUERY` dialog under the `FormOrders` resources to see how it enters the following search criteria: from- and to- dates, multiple list choices for the order status, and keyword search on the company name.

The following code sets up the query form:

```
//orders query form
//this constructor will automatically put the form into a dialog mode
DBForm qFrm(IDD_FORMORDERS_QUERY);
//set list choices
qFrm[IDC_FORMQUERY_STATUS].setListItems(stList); //same list as for the master
form
qFrm[IDC_FORMQUERY_STATUS].setEditType(DBFormEditListIndex); //cannot use virtual
column here
```

In order to enable the lens toolbar button, specify `DBFormCustomQuery` mode when loading the master form, as follows:

```
// Load all objects of ORD_MASTER into a form
DBCursor c = mCls.createCursor();
long mMode = DBFormListEdit | DBFormCustomQuery;
mFrm.load(c, mMode);
```

The full logic of the `FormOrders` application is as follows:

```
for(;;) {
    DBFormItem i = mFrm.edit();
    if (i == IDC_FORMORDERS_DETAIL)
        dFrm.edit(mFrm[i]);
    else if (i == DBFormItemQuery) {
        DBFormItem j = qFrm.edit();
        if (j == DBFormItemSave) {
mFrm.load(doQuery(sess, qFrm), mMode);
mFrm.setTitle("Search Results");
        }
        else if (j == DBFormItemDelete) {
mFrm.load(mCls.createCursor(), mMode);
mFrm.setTitle("Order");
        }
    }
    else //DBFormItemExit
        break;
}
```

Call the `edit` method on the master form and look at the return value. If the row in the table was clicked (`i == IDC_FORMORDERS_DETAIL`), then call the `edit` method on the detail form and then return to the master form through the loop as shown earlier in the document. If the user clicks on the query(lens) button (`i == DBFormItemQuery`), then pop up a query dialog (`qFrm.edit`). Once the user sets up the query parameters, the user can click on the **Save** button to execute the query (`j == DBFormItemSave`), **Delete** button to cancel the query (**j == DBFormItemDelete**) or exit the form to come back to the previous screen. The `doQuery` function creates the `DBSqlCursor` based on the search parameters in the query form. To execute the query, load the master form using the `DBSqlCursor` and change the form title to `Search Results`. If the query was canceled, reload the master form with the original list of objects (`mCls.createCursor`) and change its title back to the original.

The `doQuery` function retrieves the search parameters from the query form as `DBData` from its columns. Then it creates a SQL where-clause string and the binding list for it. The multiple choices for the status list are stored inside `DBData` as an array of integers. For the company name field, the search is performed using a `LIKE` expression—`LIKE %s%`—so that any substring matches.

12.8 Customizing Your SODA Forms Application

You can customize the UI of your application by configuring your resource file. Also, edit the `sodares.rsrc` file to modify the UI resources used by the SODA Forms library.

12.8.1 Customizing Help Messages

For PalmOS, set the Help identifier for your form to a string resource id to customize the help message displayed when the user clicks the `?` icon. Alternatively, examine SodaHelp, WordsHelp, RangeHelp and QueryHelp strings in the `sodares.rsrc` file for the default help messages when the Help ID for your form is 0.

On PocketPC, when the user clicks on the help button on the toolbar, SODA Forms launches the Windows CE help editor and loads the help file. The default help file is `SodaFormHelp.html`, which describes editing the database record. This help file is copied under the `\Windows` directory on PocketPC during installation. If you want to display different helpfiles or customize help for every screen in your application, then SODA Forms calls the `frm.setHelpFile` method where you can pass in your help file name. Your custom help file should be located under the `\Windows` directory on your PocketPC. You may include images and hyperlinks in your help file.

12.8.2 Menus

On PalmOS, if the Menu identifier for your form is 0, then SODA Forms uses SodaMenu defined in the `sodares.rsrc` file. Customize the menu or copy it into your own resource file and add more items after the standard items.

For PocketPC, SODA Forms uses the PocketPC Menubar, which is a combination of menus and the toolbar. The default Menubar comes with the SODA Forms and includes the "Edit" menu and tool bar buttons ("Left", "Right", "New", "Delete", "Save", "Revert", "Query", "Help"). The menubar can be created in the resource editor, so you can create your own or modify the default one. Read the MSDN library on how to create Menubar resources.

Remember to keep the same identifiers for the Menubar itself (`IDR_DBFORM_MENUBAR`) and the predefined menu items. You can locate these identifiers in the file `SodaRes.h`, which comes with the SODA Forms source code. You should have all the menu and toolbar items that are in the default menubar with the identifiers defined in `SodaRes.h`. You can modify toolbar icons, caption, and place toolbar items in your menus, as long as you keep the same identifiers.

To create or modify the existing menubar within SodaForms, then rebuild the project and copy the new `SodaForm.dll` on the device for the changes to take effect.

12.8.3 Default Button

For PalmOS, set the default button identifier for your form to specify what is to be returned from the edit if the user exits the application, such as when the user presses the home button.

12.8.4 Toolbar Icons

For PalmOS, you can modify the appearance of SODA Forms toolbar by editing bitmaps in SodaToolbar and SodaToolbarDis (for disabled items) bitmap families, or add new family members for color icons and high screen resolutions. Make sure not to change size or positions of the items on the toolbar.

12.9 Displaying a List Of Objects in a Table

SODA Forms allows one table control on a form to be bound to a list of objects. This list can be specified by the user, using one of the overloaded `setObjects` methods in the `DBFormCol` class. You can also bind a table to a database column that contains a list of objects. SODA supports the object pointer array datatype directly, but only normalized relational data can be replicated to the Mobile Server. Use `DBValueRel` virtual columns to map master-detail relationships into pointers.

To bind a table to the database column, perform the following:

- Add the column name for each environment, as follows:
  - For PalmOS, add the column name and table resource identifier to the `DBFormCols` array that you pass to the `DBForm` constructor.

    - For PocketPC, add the column name and the `ListView` resource identifier to the `DBFormCols` array that you pass to the `DBForm` constructor.
- Call the `setTableInfo` method in the corresponding `DBFormCol` object to specify which columns of the objects are to be displayed in the table, what are the column titles, and how the data is to be aligned.
- On PalmOS, to make a table scrollable, add a vertical scrollbar to the right of the table on the form. The library automatically detects and uses it.

Users can scroll through the table, click on the headers to sort on the column value and click a row to select it. In this case, the `edit` call returns with the object identifier of the selected row and executing the `getSelectedPos` method on the `DBFormCol` returns the row selected or `DB_NEW_POS`, if the user clicked on an empty space. These events are suppressed if the form is dirty.

The application handles any table click. However, you should load the list of objects into another `DBForm`, let user edit it, and then save the changes back to the table and to the corresponding database column. `DBForm` contains an overloaded `edit(DBFormCol)` method that does all the work automatically for you if you only have a single level of a master-detail relationship. For more complicated cases, you could do the following:

1. Maintain a stack of `DBForm` objects.
2. Call the `DBForm::load(DBFormCol)` method when you push a new form on the stack.
3. Call `DBFormCol::saveTo(DBFormCol)` before the pop.

12.10 SODA Forms UI Controls

SODA Forms supports multiple kinds of UI resources. For each, the values stored in the database and editing behavior can be customized by specifying an edit type in the third field of `DBFormCols`, as described in xxx, and optionally calling methods on the corresponding `DBFormCol` object.

In PalmOS, when editing a list of objects, the user can enter query mode by clicking a lens icon on the toolbar. In this mode, the user can click on various UI elements and enter an appropriate search condition. The library uses the control resource and edit types to determine what kind of search is supported.

The following tables summarize the behavior of the PalmOS and PocketPC control types:

- Table 12–3, " Control Types for PalmOS"
- Table 12–4, " Control Types for PocketPC"

When developing on PalmOS, use the following control types:

Table 12–3 Control Types for PalmOS

Control and Edit Type	Edit Behavior	Value Stored in Database	Query Behavior
Text field/ DBFormEditString (default edit type)	User enters any characters in the text field.	Any trailing spaces or newlines are stripped from user input. If the result is an empty string, null value is stored in the database. Otherwise, the trimmed result is stored.	User is able to enter space or comma-separated words in a popup dialog. By default, records that contain at least one word are displayed. Use +word to indicate that the word must be present in the results. Use -word to indicate that the word must not be present. For example, "snow -winter refresh retrace" locates records that contain the word "snow," and do not contain the word "winter," as well as contain either "refresh" or "retrace". Note that "snow" matches "snowball," but not "whatsnow." Use "snow -snowball" to filter the results. However, in Asian languages, the word is matched anywhere. Once finished, choose **Save** to set the new search criteria for this column, **Revert** to go back to the previous value or **Exit** to go back without making changes.
Text field/ DBFormEditNumber	The user enters any number, for example 3.14e-99.	If the field contains spaces, a null value is stored in the database. Otherwise the number entered is stored.	User is able to search for several values, such as a range (like 5-15) or an arithmetic relation (<, <=, >, >=). The dialog box enables the user to select a search criteria and the value(s) to search for.
Text field/ DBFormEditDigits	The user is able to enter digits 0-9	If the field contains spaces, a null value is stored in the database. Otherwise the number entered is stored.	User is able to search for several values, such as a range (like 5-15) or an arithmetic relation (<, <=, >, >=). The dialog box enables the user select a search criteria and value(s) to search for.
Text field/ DBFormEditDate, DBFormEditTime or DBFormEditDateTime	Calendar and/or clock dialogs pop up to pick a new value. If tbl[resId]. setAllowNull(true) is called on the column, then the Cancel button on the dialog enters a null value.	Date and/or time is stored in the database.	Same as for DBFormEditNumber, but clicking on any value fields brings up clock and/or calendar dialogs. Cancel button always clears the field in query mode, so that the user can remove some of the values to match or exclude.

Table 12–3 (Cont.) Control Types for PalmOS

Control and Edit Type	Edit Behavior	Value Stored in Database	Query Behavior
Checkbox	Can be clicked on or off. If tbl[resId]. setAllowNull(true) is called, the checkbox becomes a tri-state, where grayed-out appearance means "Unknown"	True, false or null (for unknown value) is stored in the database.	Tristate mode is always active. Grayed out checkbox means no search is done on that column.
Pop-up list/ DBFormEditDefault (default edit type)	User can choose the value from a pop-up list.	The string of the selected item is stored in the database. On input, unknown values or null are shown as an empty string. To allow user to choose a null value, include an empty string as one of the list selections.	User will be able to make multiple selections. Clicking on an item toggles its selection. If no items are selected, or <Any> is selected, then no search is done on that column.
Specify the list object ID in DBFormCols. SODA Forms automatically finds and updates the popup trigger object. Pop-up list/ DBFormEditListIndex	User can choose the value from a pop-up list.	0-based index of the selection is stored in the database. This mode is useful where the selection strings are language specific, but the database column should always have the same values.	User can make multiple selections. Clicking on an item toggles its selection. If no items are selected, or <Any> is selected, then no search is done on that column.

The form developed on PocketPC can have controls that are not bound to database, such as push buttons. Clicking on a button returns from the edit method with the button Id. When developing on PocketPC, use the following control types:

Table 12–4 Control Types for PocketPC

Control Type	Edit Behavior	Value stored in database
Edit	This is a text field for entering characters. Different styles can be specified in the resource editor, to change the appearance and the set of characters that can be entered into the field. For example, the `Number` style only allows numbers to be entered.	Any trailing spaces or newlines will be stripped from user's input. If the result is an empty string, null value will be stored in the database. Otherwise, the trimmed result will be stored. If the control is bound to a numeric column (control should have "Number" style), the entered number will be stored.
Date-Time Picker	The control stores and shows date and/or time values. It displays a month calendar to modify the value. You can use any of the following formats: `Short Date`, `Long Date`, `Time`, or a custom format. To set the custom format, call the `tbl[resId].setDateFormat` method with the custom format string. See the MSDN library for the `DateTime_SetFromat` method on how to construct the custom format. Use the `Show None` style to null the date/time. This places a check box on the left and the control has an unspecified (null) value when the check box is not checked. The `Allow Edit` style enables manual editing of the date string.	Date/time value is stored in the database. If the control has the `Show None` style and is in unspecified state, then a null value is stored.
Checkbox	The checkbox can be checked on or off. If Tri-state style is specified, then the checkbox can have a greyed-out appearance if the value is unknown.	True, false or null (for unknown value) is stored in the database.
Listbox	Displays a list of text items. Set iether single or multiple selection type in the resource editor. The latter is useful for the query mode when you want to query by multiple choices within the list.	Single-selection listbox: If the column edit type is default, the string of the selected item is stored in the database. On input, unknown values or null are shown as an empty string. To allow user to choose a null value, include an empty string as one of the list selections. If the column edit type is DBFormEditListIndex, 0-based index of the selection is stored in the database. This mode is useful where the selection strings are language specific but the database column should always have the same values. Multiple-selection listbox: List selections are stored as array of strings or numbers (when edit type is DBFormEditListIndex). Remember that DBData can store arrays of values of certain type. If there are no selections, null value is stored.
ComboBox	This is a combination of a listbox and the edit control. `Drop List` is a popup list. `Dropdown` is a popup list that contains edit control for editing the selection. Multiple selections are not supported for the Combobox.	Same as listbox. For "Dropdown" format, with the default edit type, the strings which are not in the original list can be loaded into the edit area of the combo box or stored in the database.

12.11 OKAPI API

OKAPI is an older interface to Oracle Lite object kernel and is deprecated.

13 Developing Mobile Applications for PalmOS Devices

Oracle Database Lite for Palm OS supports Open Database Connectivity (ODBC) and Simple Object Database Access (SODA) programming interfaces. In addition, you can build and execute Oracle Database Lite applications using Metrowerks CodeWarrior 9. These subjects are described in the following sections:

13.1 Installing Oracle Database Lite Runtime on the Device

You need to install the `Runtime\olSetup.prc` on the device to run Oracle Database Lite applications. If you are installing on the emulator, click the **olSetup** icon in the "Oracle Lite" program group; if you are installing on the device using HotSync, `olSetup` is executed automatically.

Note: A sync with Mobile Server replaces the Oracle Database Lite runtime on the device with what is installed on the server. Choose the "Ignore apps" option in the synchronization settings to suppress this behavior.

The `tutorial.html` file demonstrates how you build a small application for the Palm environment. The file is located in the following directory:

<*ORACLE_HOME*>`\Mobile\Sdk\Palm\doc\tutorial.html`

Access additional Palm-specific documentation, including the tutorial, in the following directory:

<*ORACLE_HOME*>`\Mobile\Sdk\Palm\doc\index.html`

13.2 Uninstalling or Replacing Oracle Database Lite Runtime

Oracle Database Lite runtime includes deLite, an application that can be used for the following tasks:

- Remove all Oracle Database Lite, but leave applications and shared libraries in place. This option can be used if Oracle Database Lite becomes corrupted (the application crashes or displays invalid data). Do a synchronization to restore the user's data to a device.
- Remove both Oracle Database Lite runtime and the databases. Use this option if you no longer need Oracle Database Lite on the device or before manually installing a new version.
- Uninstall Oracle Database Lite and then install a current version from the specified Mobile Server (by default, the same one used for synchronization). Use this to reset the device to a known version of the Oracle Database Lite or upgrade from a version prior to 5.0.2.9.0, that does not support automatic upgrade.

13.3 Running Oracle Database Lite on Palm OS Emulator

To install Oracle Database Lite runtime on the PalmOS emulator, choose "Install application/database" from the right-click menu and select `olSetup.prc` in the Mobile client or Mobile SDK directory. Run `olSetup` once to install the Oracle Database Lite runtime. Navigate to the emulator debug options and clear the "MemMgr semaphore" and "Proscribed function calls" checkboxes. Oracle Database Lite runtime uses these features properly and the emulator warnings for these conditions should be disabled or ignored.

Go to Preferences and check "Redirect NetLib calls to Host TCP/IP". This enables you to synchronize with the Mobile Server using the emulator.

13.4 Running Oracle Database Lite on Palm OS Simulator

Oracle Database Lite works on PalmOS 5.x devices; however, the debug version of the PalmOS Simulator fails when executing `olSetup`, because of a bug in the simulator. The release version of the simulator works properly.

13.5 Using Oracle Database Lite Base Libraries

Oracle Database Lite provides several libraries that do not directly provide database functionality, but are used by the rest of the runtime. Follow the following steps to add these libraries to your project:

1. Replace the CodeWarrior with `cwStartup.lib` (or `cwStartup4B.lib` if using 4-byte integers). Make sure the library is in the first segment. Add `pslm_app.lib` to the first segment as well. See `pslm.html` for more details.
2. Add `libc_stub.lib` and `olstd.lib` to any segment of your application.
3. Include `olstd.h` in your source files

4. If your PilotMain is started with a launch code that allows access to global variables, call `psCLibrary.open(true)` before using Oracle Database Lite.
5. Use Constructor to generate a PREF resource and set stack size of your application to 8 KB.
6. Oracle Database Lite interfaces may change between versions. To avoid compatibility problems, Oracle Database Lite shared libraries return errors if the application is not linked with the matching version of the stub library. When upgrading Oracle Lite runtime, re-link your application with the new stubs.

These steps enable access to the Oracle Database Lite C library, which provides many of the ANSI C functions that are otherwise missing from the PalmOS platform. Examine `libc.h` for a list. The `olstd.h` file defines C++ classes such as a hash table, which are documented as a part of SODA.

The `libc.h` file defines standard I/O functions such as printf for debugging purposes. To see the output, run `java BigBrother` on the same PC that is running Palm emulator or `java BigBrother <IP Address>` to capture output from a PalmOS device connected through PPP. Note that the program will be blocked when it tries to use `printf` until the listener is connected.

13.6 Building a SODA Application

To use SODA, add the following libraries to your project: `olSDT.lib`, `okapi_stub.lib`, `soda1.lib` and `soda2.lib`. Include `soda.h` in the source files. See Chapter 12, "Using Simple Object Data Access (SODA) for PalmOS and PocketPC Platforms" for more details.

13.7 Building a SODA Forms Application

To use SODA Forms, include all the SODA libraries and also add `sodaform.lib` and `sodares.rsrc` into your application. The later file contains UI resources used by SODA Forms. Avoid using resource ids above 30000 for your own resources to prevent conflicts.

Include "`sodaform.h`" in your source files.

The file **sodaforms.htm** discusses SODA Forms, a library for rapid development of data entry applications on Palm. The file is located in the following directory.

`<ORACLE_HOME>\Mobile\Sdk\Palm\doc\index.html`

13.8 Building an ODBC Application

To use ODBC (actually a subset of standard ODBC that we support on Palm), include "`odbc.h`" and link with `odbc_stub.lib`.

13.9 Packaging your Application with Oracle Database Lite Runtime

The file **olSetup.prc** is the Oracle Database Lite installer which extracts a number of **.prc** files when synchronized with a device or run on the emulator. It is possible to make a version of **olSetup** with additional applications by modifying and running **makesetup.bat** in the Sdk/setup directory.

The next step is to remove "**olSetup application**" in `ORACLE_HOME\Mobile\sdk\Palm\sdk\setup` directory from the Mobile Server and publish another application with the new version of olSetup.prc as one of the deployed files. The following events will happen on the next sync:

1. Any changes to data will be first pushed to the server.
2. Oracle Database Lite libraries and databases will be uninstalled from the device.
3. New Oracle Database Lite runtime (and any applications you added) will be installed from olSetup.prc
4. A full synchronization will be done with the server to restore the databases.

This process will upgrade the version of Oracle Database Lite on the device and avoid any database compatibility problems.

13.10 Customizing Oracle Database Lite Runtime

In addition to adding your application, you might want to customize Oracle Database Lite runtime itself. The following changes can be made in `makesetup.bat`:

Table 13–1

Change	Effect
Add olEncryptTransport.prc	Enable AES encryption of data during synchronization. Note that this does not work with external authentication on the server side.
Remove olLibCrypto.prc	Disable AES encryption altogether, including database encryption.
Remove olCompressTransport.prc	Disable compression during sync. Can be useful on devices with very little memory
Remove odbc.prc	If you are only using SODA
Remove msql.prc	You may not need this tool on end-user devices
Substitute okapi.prc from Sdk\setup directory with okapi.prc from Sdk\setup\card subdirectory (copy okapi.prc from Sdk\setup\card one level up) and rerun makesetup.bat	Will create olSetup.prc for the storage card version of Oracle Database Lite. Enables you to use the storage card on palm device to store Oracle Database Lite databases instead of main memory. This will greatly relax the limits on the database size (will be limited only by storage card size). Olite databases will be located in OLDB directory of the storage card. Since storage card support constitutes different Olite installation, we do not currently support accessing both storage card and in-memory databases from the same application.

13.11 Using Mobile Sync for Palm

Mobile Sync (mSync) for Palm enables a user or a developer to synchronize data with the Mobile Server. The user can configure and execute the application manually. Then, tpa the Sync button to manually initiate synchronization over the default network connection configured on the PDA.

Alternatively, you can invoke synchronization with pre-configured settings, as follows:

- HotSync automatically synchronizes the Oracle Database Lite databases if the Oracle Database Lite conduit is installed on the desktop.
- The `DBSession::sync()` method, which is part of SODA interface, launches msync and attempts to synchronize over the network connection.
- SODA Forms applications have the "Sync data with the Mobile Server" option in their "Form" menu.

13.11.1 Configuring msync

When msync executes, it displays a screen with the most common synchronization settings. Table 13–2 shows the controls for the synchronization settings that are displayed by msync.

Table 13–2 List of Controls for Synchronization Settings

Option	Use
User Name	Case-insensitive user name on the Mobile Server
Password	Case-insensitive password on the Mobile Server
Change (password)	If selected, New and Confirm fields are shown. Enter the new password twice to guard against typing mistakes. On the next successful sync, Mobile Server password is changed.
Save password	Tap this checkbox to save the password on the PalmOS PDA. You will not have to reenter password every time you sync, however anyone with physical access to your Palm will be able to sync, and possibly discover your password. This option is required to use HotSync and automatic sync through SODA or SODA Forms applications.
Server	Enter `hostname` or `hostname:port` of your Mobile Server. When synchronizing over the network, you can enter the IP address to bypass DNS configuration problems.
Proxy	When this checkbox is active, an additional Proxy field appears on screen. Enter the hostname or hostname:port of your HTTP proxy server. This option only applies during HTTP sync. To configure a proxy server for HotSync, check Internet Explorer settings on your desktop.
Secure	Activates the secure sync over HTTPS rather than plain HTTP. You need a PalmOS 5.2 or later PDA to do a secure network sync. However, any device can HotSync using this option. HTTPS sync normally requires a valid certificate to be purchased and installed on the Mobile Server. For development purposes, you may prepend @! to server's hostname to test without a valid certificate. This option should never be suggested to end users, as it undermines the security provided by HTTPS.
Forced	Tap this checkbox to do a complete refresh on the next sync. This can solve some data consistency problems. Note that this option is automatically cleared after one sync.
Log button	Tap this button to launch LiteLog application. You will be able to see the log of the recent failed and successful sync attempts, as well as any crashes or critical errors encountered by Oracle Database Lite applications.
Sync button	Starts a sync over the default network connection.
Cancel button	Exits msync and re-starts the SODA or SODA Forms application, if any, that initiated it.

The msync executable also provides the synchronization settings item under the Options menu, which can be used to adjust less frequently used settings, as described in Table 13–3.

Table 13–3 Synchronization Settings in the Options Menu

Option	Use
Hangup after sync	Hangup the network connection immediately after the sync is done. Normally, the connection will be left on and disconnected when the timeout configured in the Network settings panel expires.
Push only	Send locally made changes to the server, but do not get any data back. This is a quick way to backup local data to the server.
Ignore apps	Disable application deployment and auto-upgrade of the Oracle Database Lite runtime. Tap this checkbox to sync with a version of Mobile Server different from the client version without causing an upgrade.
Remote Hotsync	Tap this checkbox when doing a Network hotsync to enable successful retries if the Palm Desktop times out.
NLS Code	Enter an Oracle-supported language code to sync using the appropriate character set. Most Japanese, Chinese or Korean devices are automatically detected by Oracle Database Lite, but some third-party language add-ons are not.

13.11.2 Using HotSync to Synchronize Data with the Mobile Server

The following sections discuss how to use HotSync to synchronize data with the Mobile Server.

- Section 13.11.2.1, "Configuring HotSync for a PalmOS Device"
- Section 13.11.2.2, "HotSync Timeout Errors"
- Section 13.11.2.3, "Configuring PalmOS Emulator for HotSync"

13.11.2.1 Configuring HotSync for a PalmOS Device

To synchronize Oracle Database Lite databases over HotSync, perform the following:

1. Install Mobile client for PalmOS on a machine that already has Palm Desktop.
2. Start the msync executable on the PDA and configure all the sync options, including the "Save password" checkbox.
3. Find and enable "Stay on in Cradle" options in the PalmOS Prefs application.

Every HotSync automatically synchronizes Oracle Database Lite data with the Mobile Server. After HotSync completes, refer to the LiteLog on the device or HotSync log on the desktop to check for errors.

13.11.2.2 HotSync Timeout Errors

If the Mobile Server sends a large volume of data to the PDA, then the Palm Desktop may timeout during HotSync and a message box pops up on the desktop while the PDA is still processing sync data. Dismiss the dialog or let it time out automatically. Then, msync detects this condition and automatically performs another HotSync, which should process successfully.

Note: Note that if you are viewing the HotSync log when the PDA reconnects and in some other conditions the retry may fail. Generally, the error can be ignored, except if another application registered a low priority conduit that is supposed to run after Oracle Database Lite. In this case, manually perform another HotSync.

This timeout issue exists within the Palm Desktop software, rather than within Oracle Database Lite. In particular, the `SyncCallRemoteModule` API that is used by a conduit to invoke an application on Palm can be problematic. If the application takes a lot of time to execute, Palm Desktop times out and aborts HotSync. If PalmSource provides a configurable timeout in a future release of Palm Desktop, the message box can be avoided by increasing the timeout value.

13.11.2.3 Configuring PalmOS Emulator for HotSync

It is possible to perform a Network HotSync with the Palm emulator with the following steps:

1. Enable Network in HotSync tray menu on the desktop running the Palm emulator.
2. Check the "Remote HotSync" checkbox in msync options menu.
3. Make sure that "Redirect NetLib calls to Host TCP/IP option" is enabled.
4. Run the HotSync application. Click the "Modem" (rather than "Local") push button.
5. In Modem sync preferences, choose "Network" rather than "Direct to modem".
6. In "Primary PC setup", enter "localhost" as the hostname and "127.0.0.1" as the IP address.
7. Click on the connection field below the HotSync icon and choose "AT&T WorldNet" or any other connection.
8. Click on the HotSync icon in the middle of the screen to do a network HotSync.

13.11.3 Using Network Sync

The msync executable can open a direct connection to the Mobile Server without going through the Palm Desktop. If you have a valid network connection, such as through a modem, integrated cellular phone or Bluetooth, then you can tap the Sync button on the msync main screen.

13.11.3.1 Synchronizing Using a Cradle and Windows Desktop

It is possible to connect a PalmOS device to the network using a cradle connected to your PC rather than a dedicated modem. First, you need to enable Incoming Connections. Modern PalmOS devices use a USB cradle, that requires 3rd-party software to establish a network connection. One such product is Softick PPP, which can be purchased from `http://www.synclive.com/ppp`. Note that we don't offer support for any issues you may encounter with such 3rd-party software.

For serial cradles, follow the following steps:

Setting up the desktop

1. Uncheck "Local Serial" option in the HotSync pop-up menu and leave it off while you are using the cradle for networking

2. Go to Modem control panel and create a new modem of type "Communications cable between two computers". Set "Maximum port speed" to 56K and "Flow control" to Hardware.
3. In Windows XP, open the Network control panel, choose "Create a new connection", then "Setup a new connection" and finally "Accept incoming connections". Choose the modem you just added in "Devices for Incoming Connections". Select at least one user in "Users you allow to connect". When you get to TCP/IP settings, choose "Allow callers to access my local area network".
4. If your PC is not using DHCP, you need to find two sequential unused IP addresses on your local subnet and enter them in the fields under "Specify TCP/IP address.
5. Older versions of Windows have different ways to enable incoming connections. For example, in Windows NT 4.0 the equivalent functionality is known as "Remote Access". You may need to adapt the instructions for the version running on your desktop.

Setting up the device

1. Go to Connections panel in Palm preferences. Edit the "Cradle/Cable" connection details by setting speed to 56K and flow control to Hardware.
2. In Network panel, create a new service. Enter username and password of the user you selected while setting up the desktop. Set the "Connection" to "Cradle/Cable".
3. Choose Details/Script for your service. Enter the following script:

   Send: `CLIENT`

   Send: `CLIENT`

   Wait For: `CLIENTSERVER`
4. Test the service by clicking Connect button. Once the connection is successful, choose Options/View Log from the Preferences menu. Type ping servername or ping serverip. If the ping succeeds, you should be able to sync with that Mobile Server by tapping the appropriate button in msync.

13.11.3.2 Network Sync With PalmOS Emulator

To use msync on the emulator, make sure "Redirect NetLib calls to Host TCP/IP" option is set in the preferences. To test SSL, run PalmOS simulator version 5.2 or later.

13.12 Palm Shared Library Manager (PSLM)

This document discusses the Palm Shared Library Manager (PSLM). It includes the following topics:

- Section 13.12.1, "Overview"
- Section 13.12.2, "Trying out PSLM"
- Section 13.12.3, "Writing a PSLM Library"
- Section 13.12.4, "Building a Shared Library Project"
- Section 13.12.5, "Calling a PSLM Library from Your Application"
- Section 13.12.6, "Building an Application Using PSLM"
- Section 13.12.7, "Exceptions Across Modules"

- Section 13.12.8, "Cloaked Shared Libraries"
- Section 13.12.9, "Patching the CodeWarrior Runtime"

13.12.1 Overview

PalmOS provides built-in facilities to load and use shared libraries. However, native support has severe limitations that make it virtually impossible to port existing code of significant size. The size of native shared libraries is limited to 32-64K, depending on code structure. In addition, global variables and many important C++ features, such as virtual functions and exceptions, can not be used. There are a couple of techniques to overcome these limitations, such as PRC-Tools glib support and CodeWarrior 9 "expanded mode". However, none of them succeeds in making a shared library as easy to develop as an application.

Oracle's solution, PSLM, allows one to build a shared library as a regular Palm application. It is possible to use multiple segments, C++ virtual tables and exceptions and global variables, even ones with constructors and destructors. PSLM doesn't require any support from the OS and only uses limited compiler support (a patch the the publicly available sources of the CW runtime library). It does require some extra code to be written, but the existing code of the application and an existing static library does not need to be modified.

13.12.2 Trying out PSLM

There is a sample using the framework in <*ORACLE_HOME*>`\Mobile\Sdk`. After you do "Build All" on the project file, there will be two PRC files in that directory - `SampleLibrary.prc` and `SampleApp.prc`. Install Oracle Database Lite Runtime using **olSetup**.If you touch the "PSLM Sample" icon on the Palm now, it will just print a couple of message boxes and exit. What happens inside is considerably more interesting. `SampleLibrary.prc` is a PSLM shared library that has global variables and even makes use of another shared library, the ANSI C library that comes with Oracle Database Lite. Look at `Sample.cpp` for the implementation of the shared library and `AppStart` of `Starter.cpp` to see how it is called.

13.12.3 Writing a PSLM Library

A PSLM library is a C++ class that extends PSLibrary. It exposes all it's functionality as virtual functions. Here is how the `SampleLibrary` class looks like.

```
class SampleLibrary : public PSLibrary {
protected:
        /*
         * Overloaded PSLM functions.
         */
        virtual pslmError startup();
        virtual void cleanup(bool isFinal);
public:
        /**
         * Increment an internal counter by a specified value and then
         * return a result as a string (global buffer that will be reused
         * on next call).
         */
        virtual const char *getCounter(int incVal);
        /**
         * Reset a counter to the specified value
         */
        virtual void setCounter(int newVal);
};
```

This library defines two APIs - getCounter and setCounter. The remaining two virtual methods - startup and cleanup are called by PSLM itself and are very important. Basically, a PSLM library must use it's startup and cleanup methods rather than constructor and destructor to manage it's state. Also, it must be able to handle another startup after cleanup is done. This is one of the few artifacts caused by lack of the compiler/OS support. The constructor of a shared library is called normally, but must not use PSLM itself or even call the methods of it's own object. The destructor is actually not called at all. Note that it's perfectly Ok to have a pointer to another object that is constructed during startup and deleted during cleanup.

startup() method is the place to do initialization, including loading additional libraries. Let's look at the startup method of SampleLibrary:

```
pslmError SampleLibrary :: startup() {
     PSLibContext ps(this); // Establish access to globals
     cleanOrder = 5; // Unload before libraries with clean order 4 and below on
exit
     return psCLibrary.open();
}
```

The first line of this method is the most important, but we'll come back to it in a moment. The second line lets you specified the order in which the libraries will be unloaded on program exit. If B depends on A, A should have a smaller cleanOrder. The last line loads the C library and returns success or error of that application.

startup() function can fail by returning a value rather than 0. In this case, the library being loaded is removed and the error is returned to the caller.

cleanup() method should free all the memory, closing network connections, unload dependencies and so on. However, if the isFinal argument is set to true it must not unload other libraries because the program is exiting and it will interfere with PSLM closing libraries correctly. Here is the cleanup method of SampleLibrary:

```
void SampleLibrary :: cleanup(bool isFinal) {
        PSLibContext ps(this);
        if (!isFinal)
                psCLibrary.close();
}
```

Let's look at a regular method of SampleLibrary, together with the variables it's using:

```
class SampleBuf {
        char *buf;
public:
        SampleBuf(int size) : buf(new char[size]) {}
        ~SampleBuf() { delete[] buf; }
        operator char *() { return buf; }
};
SampleBuf myBuf(128);
int counter;
const char * SampleLibrary :: getCounter(int incVal) {
        PSLibContext ps(this);
        counter += incVal;
        sprintf(myBuf, "%d", counter); // Use LibC - another shared library
        return myBuf;
}
```

Note that this method uses two global variables and one of them even has a constructor and a destructor. This is Ok, although global constructors and destructors can only do simple things. They shouldn't call PSLM and shouldn't use other shared libraries unless you are sure they are always loaded.

Look at the highlighted line, PSLibContext ps(this). Every exposed virtual method of a PSLM shared library must start with this line. The constuctor of a PSLibContext sets the context to that of the library passed as an argument. This is what enables a library to use it's global variables, virtual functions and exceptions. If you omit this line, you will get crashes, call random places in memory or even introduce hard-to-track memory corruption. Also, remember to delete the context before calling any callback in the main program and re-create it afterwards. If you get this right, you have mastered PSLM.

The last piece to consider is the library's PilotMain, which is very simple:

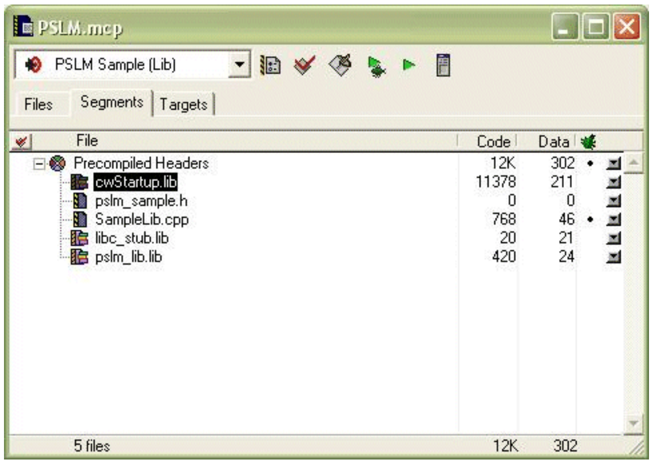
```
UInt32 PilotMain( UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags) {
        if (cmd == psLibLaunchCode)
                psLibInit(cmdPBP, new SampleLibrary()); // Never returns
        return 0;
}
```

psLibLaunchCode is what PilotMain gets when the library is open. psLibInit takes a pointer to a subclass of PSLibrary. It never returns directly. Instead, it returns the control back to the program that opened the library.

13.12.4 Building a Shared Library Project

The following illustration, Figure 13–1, shows the CodeWarrior project for SampleLibrary:

Figure 13–1 The CodeWarrior Project for SampleLibrary



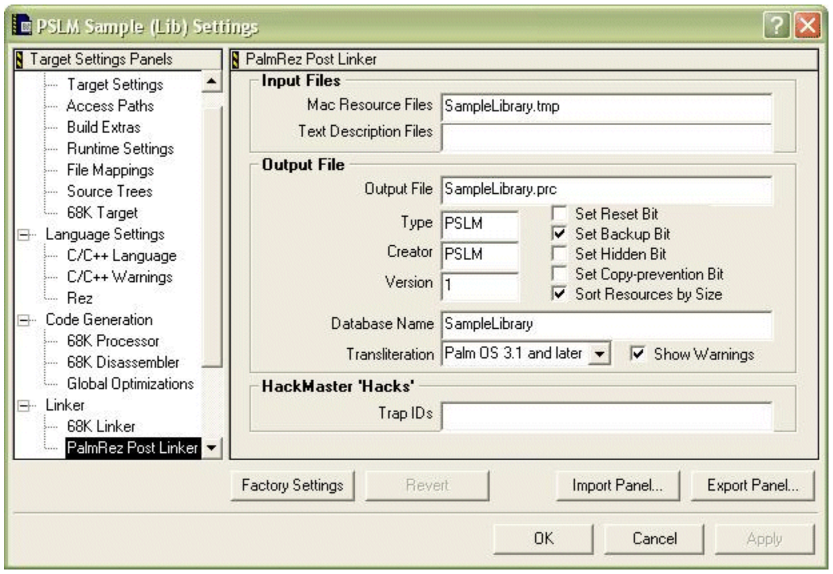
File	Code	Data
Precompiled Headers	12K	302
cwStartup.lib	11378	211
pslm_sample.h	0	0
SampleLib.cpp	768	46
libc_stub.lib	20	21
pslm_lib.lib	420	24
5 files	12K	302

First, note that the usual CodeWarrior startup library (PalmOSRuntime_2i_A5.lib) has been replaced with cwStartup.lib from our distribution. You can use this patched startup library for any project, but it must be used to build a PSLM shared library. I tried to avoid requiring a custom runtime, but recent Metrowerks changes and especially PalmOS5 support made a patch necessary. Use cwStartup4B.lib if you are building a project using 4-byte integers. Finally, you might want to patch your own runtime if you are using a version of CodeWarrior newer than 8.3.

Both `cwStartup.lib` and `pslm_lib.lib` must be in the first segment of the application, otherwise it will crash when loaded. Other files can be in any number of segments. In this example, the included SampleLib.cpp and libc_stub.lib libraries are static helpers for ANSI C shared libraries. A `pslm` library must not contain any UI resources, because they will be used instead the corresponding ones of the application. Be sure to exclude your `Starter.rsrc` from shared library targets.

The following illustration, Figure 13–2, shows the "PalmRez Post Linker" section of the SampleLibrary project:

Figure 13–2 The PalmRez Post Linker Section



PalmRez Post Linker

Input Files

Mac Resource Files: SampleLibrary.tmp

Text Description Files:

Output File

Output File: SampleLibrary.prc

Type: PSLM

Creator: PSLM

Version: 1

☐ Set Reset Bit
☑ Set Backup Bit
☐ Set Hidden Bit
☐ Set Copy-prevention Bit
☑ Sort Resources by Size

Database Name: SampleLibrary

Transliteration: Palm OS 3.1 and later ☑ Show Warnings

HackMaster 'Hacks'

Trap IDs:

A PSLM library is linked as an application, but we do not want it to show up as a Launcher icon. Change type and creator of the .PRC file to PSLM in order to hide it. Also, set the database name to whatever name you are planing to use when you load the library.

13.12.5 Calling a PSLM Library from Your Application

To call a shared library from your application, first use a template class to declare a proxy object for that library:

```
PSLibObject<SampleLibrary> sampleLib("SampleLibrary");
```

The quoted SampleLibrary is the Palm Database Name you specified in the project settings, while the quoteless one if the name of the class that exposes the APIs. You can make these two different if you want. You can load the library using:

```
sampleLib.open(true);
```

In the above statement, "true" argument means that a fatal exception will be displayed on the device if the library can not be open. For a nicer error handling, and especially if the library is optional, just do sampleLib.open() without arguments and process the returned Err value if not errNone.

Once opened, you can pretend that sampleLib is a SampleLibrary * and write code such as the following:

```
StrPrintF(buf, "Value after increment by 3: %s", sampleLib->getCounter(3));
```

This is actually not very convenient if you originally just had a static library that defined getCounter. Remedy this problem with preprocessor directives like this one:

```
#define getCounter (sampleLib->getCounter)
#define setCounter (sampleLib->setCounter)
```

At this point, you can use getCounter(3), same as with a static library.

Should you call sampleLib.close() to unload SampleLibrary? You can if you need to free the resources immediately. open() and close() keep a use count and only unload when it drops to 0. Note though that all the libraries will be automatically unloaded (ordered by increasing cleanOrder) when the program exits.

If you use sampleLib in more than one file in your program, you should load it in your AppStart and then declare it as follows in other files:

```
extern PSLibObject<SampleLibrary> sampleLib;
```

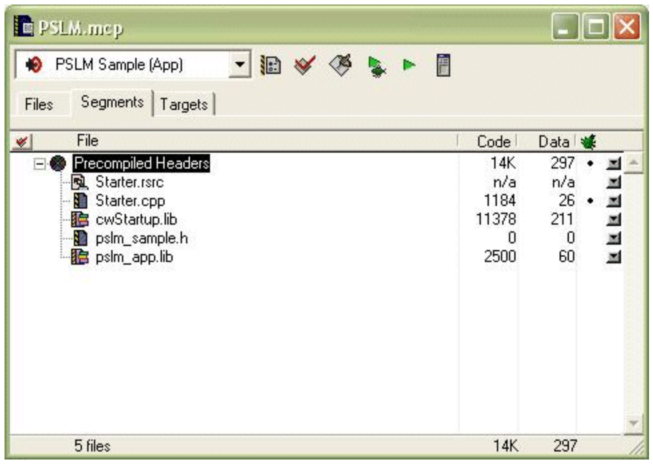
In the other extreme, you can have a function that loads a plugin, lets it do some work and then unloads it before returning. In this case, you can declare a local variable of type PSLibObject and even pass a dynamic argument as a library name to support user-defined plugins.

The last technique is linking statically to the code that was intended to be a shared library. If you add SampleLib.cpp to the project (comment out it's tiny PilotMain), you can do sampleLib.init() before open. This will call a statically linked default constructor of the template argument and then register the object as a fake shared library. One interesting result is if MAIN loads LIB1 statically and LIB2 dynamically and then LIB2 tries to load LIB1, it will get a static copy embedded in main and the dynamic LIB1 doesn't need to be installed on a device. This allows the main program to determine exactly which components are statically linked.

13.12.6 Building an Application Using PSLM

An application using PSLM must be linked with `pslm_app.lib` and it must reside in the first segment. Although this example uses `cwStartup.lib`, applications can use a regular runtime libraries and only shared libraries need a patched one. Figure 13–3 shows a PSLM sample application.

Figure 13–3 PSLM Sample Application



File	Code	Data
Precompiled Headers	14K	297
Starter.rsrc	n/a	n/a
Starter.cpp	1184	26
cwStartup.lib	11378	211
pslm_sample.h	0	0
pslm_app.lib	2500	60
5 files	14K	297

13.12.7 Exceptions Across Modules

You are free to use exceptions inside the shared library, as long as they are also caught inside. Unfortunately, its currently not possible to throw an exception in a library and catch it in the main program. What you want to do, is catch the exception at the top level API method and store it as a private field in the PSLibrary subclass. Then add a non-virtual method that checks that field and re-throws an exception. Basically, non-virtual methods are always static. If you use them both in the library and it's caller, you must link them with both. For this case, it's the easiest to use an inline method for re-throwing the exception.

13.12.8 Cloaked Shared Libraries

Certain C++ features, such as global variables and exception handling, allocate large amounts of dynamic heap when the program is loaded into memory. PSLM has a feature that allows allocating a shared library's data segment in storage heap instead. To use this feature, add one more argument to the PSLibObject template:

```
PSLibObject<SampleLibrary, true> sampleLib("SampleLibrary");
```

Internally, this will cause PSLibContext to call MemSemaphoreReserve(true) in the constructor and MemSemaphoreRelease(true) in the destructor to temporarily un-protect storage heap while a method of the cloaked library is executing. In some cases, for example if a shared library returns a pointer to its global variable to the caller, you may need to do it yourself to modify the data. You can declare a variable of PMLock class on the stack to unprotect the storage heap for the duration of its scope.

Note that you can not get input from the user while the memory semahore is locked. Anything that calls EvtGetEvent directly or through another system call will hang. Therefore, cloaked libraries are only suitable for tasks that don't require user's input.

13.12.9 Patching the CodeWarrior Runtime

PSLM requires a patched version of CodeWarrior runtime libraries to link a shared library. The Oracle Database Lite build includes `cwStartup.lib` and `cwStartup4B.lib`, which are pre-patched versions of the runtime libraries that come with CodeWarrior 8.3. If you want to make your own patched runtime, you need to modify `PalmOS_Startup.cpp`.

This section is much more difficult than the rest of the document. You need some experience reading system-level code and applying other people's patches to follow it. Otherwise, you may want to stick with our pre-patched version or ask someone with the experience for help.

Let's start with a unified diff generated for CodeWarrior 8.3:

```
--- PalmOS_Startup_old.cpp      2002-07-19 18:41:26.000000000 -0700
+++ PalmOS_Startup.cpp  2002-09-08 15:51:29.000000000 -0700
@@ -396,6 +396,12 @@


 #endif /* SUPPORT_A4_CONST_GLOBALS */


+SysAppInfoPtr pslmGetAppInfo(
+       SysAppInfoPtr *rootAppPP,
+       SysAppInfoPtr *actionCodeAppPP)
+       SYS_TRAP(sysTrapSysGetAppInfo);
+
+

 /*
  *     Main entry point for applications
  */
@@ -408,8 +414,9 @@
        SysAppInfoPtr   appInfoP;
        Int16                  abort_result = 0;
        Boolean                globals_are_setup;
-       _CW_Features    features;
-#if SUPPORT_A4_CONST_GLOBALS
+       _CW_Features    lFeatures;
+       static _CW_Features gFeatures;
+ #if SUPPORT_A4_CONST_GLOBALS
        UInt32                 originalA4 = GetA4();
        MemPtr                 originalExtraP;


@@ -435,18 +442,31 @@
     }
 #endif

+

        /*
         *      Call the standard system code for allocating and initializing
globals and
         *      setting up A5, and getting the command line arguments
         */
-       err = SysAppStartup(&appInfoP, &prevGlobalsP, &globalsP);
+       // PSLM - try to find and execute custom startup code
+#define psLibLaunchCode ((UInt16)0xC001)
+       typedef Err (*appStartup)(SysAppInfoPtr* appInfoPP, MemPtr*
prevGlobalsPtrP,
+                                                      MemPtr* globalsPtrP);
+       appStartup start = NULL;
+       SysAppInfoPtr uiP, curP;
+       appInfoP = pslmGetAppInfo(&uiP, &curP);
+       if (appInfoP->cmd == psLibLaunchCode)
+               start = (appStartup)appInfoP->extraP;
+       if (start)
+               err = start(&appInfoP, &prevGlobalsP, &globalsP);
+       else
+               err = SysAppStartup(&appInfoP, &prevGlobalsP, &globalsP);
        if (err) {
                ErrDisplay("Error launching application");
                return 0;
        }


        globals_are_setup = (appInfoP->launchFlags & sysAppLaunchFlagNewGlobals)
!= 0;
-

+       _CW_Features &features = globals_are_setup ? gFeatures : lFeatures;
        /* initialize runtime globals */
 #if SUPPORT_A4_CONST_GLOBALS
        originalExtraP = appInfoP->extraP;
```

If you have a similar version of PalmOS_Startup.cpp, you can place this diff into a patch program. But a patch will fail if Metrowerks made a lot of code changes. Let me walk you through the changes so that you can still make a functionally equivalent patch.

First, we need to declare a prototype of a PalmOS system function that is not declared in regular Palm SDK. Put the prototype just before __Startup__:

```
SysAppInfoPtr pslmGetAppInfo(
        SysAppInfoPtr *rootAppPP,
        SysAppInfoPtr *actionCodeAppPP)
        SYS_TRAP(sysTrapSysGetAppInfo);
/*
 *      Main entry point for applications
 */
extern "C" UInt32 __Startup__(void)
```

Next, 8.3 version of __Startup__ declares a local variable of type _CW_Features and then stores it in a global pointer that is used elsewhere. This is not good for PSLM, because it will continue calling functions in a shared library after __Startup__ is removed from the stack (by a longjmp in psLibInit). Find the declaration of the variable:

```
_CW_Features features;
```

Instead we need to declare both a global version (for PSLM) and a local version (for a sublaunch without access to globals in other projects):

```
_CW_Features lFeatures;
static _CW_Features gFeatures;
```

Now, find this line right after the call to SysAppStartup:

```
globals_are_setup = (appInfoP->launchFlags & sysAppLaunchFlagNewGlobals) != 0;
```

At this point, we know if the program has global access and if it uses a correct version of features:

```
globals_are_setup = (appInfoP->launchFlags & sysAppLaunchFlagNewGlobals) != 0;
_CW_Features &features = globals_are_setup ? gFeatures : lFeatures;
```

Consider the following call:

```
err = SysAppStartup(&appInfoP, &prevGlobalsP, &globalsP);
```

The following code shows what the call should be turned into:

```
#define psLibLaunchCode ((UInt16)0xC001)
        typedef Err (*appStartup)(SysAppInfoPtr* appInfoPP, MemPtr*
prevGlobalsPtrP, MemPtr* globalsPtrP);

        appStartup start = NULL;
        SysAppInfoPtr uiP, curP;
        appInfoP = pslmGetAppInfo(&uiP, &curP);
        if (appInfoP->cmd == psLibLaunchCode)
                start = (appStartup)appInfoP->extraP;
        if (start)
                err = start(&appInfoP, &prevGlobalsP, &globalsP);
        else
                err = SysAppStartup(&appInfoP, &prevGlobalsP, &globalsP);
```

SysAppStartup initializes global variables and multiple code segments for normally loaded applications. PSLM libraries are loaded somewhat abnormally and, in PalmOS 5, SysAppStartup can no longer initialize them correctly without messing up the calling program. An equivalent process is now performed by the code inside PSLM and we must modify CodeWarrior runtime to call this custom function for a shared library launch code. To save space, the internal function only does the same work as PalmOS 1.0, so do npt disable the support for old devices in build options.

14 Oracle Database Lite ADO.NET Provider

The following sections discuss the Oracle Database Lite ADO.NET provider for Microsoft .NET and Microsoft .NET Compact Framework. The Oracle Database Lite ADO.NET provider resides in the `Oracle.DataAccess.Lite` namespace.

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

- Section 14.1, "Discussion of the Classes That Support the ADO.NET Provider"
- Section 14.2, "Running the Demo for the ADO.NET Provider"
- Section 14.3, "Limitations for the ADO.NET Provider"

14.1 Discussion of the Classes That Support the ADO.NET Provider

To use the Oracle Database Lite ADO.NET provider from your own project, add a reference to `Oracle.DataAccess.Lite_wce.dll`. This section describes the following classes for the Oracle Database Lite ADO.NET provider.

- Section 14.1.1, "Establish Connections With the OracleConnection Class"
- Section 14.1.2, "Transaction Management"
- Section 14.1.3, "Create Commands With the OracleCommand Class"
- Section 14.1.4, "Maximize Performance Using Prepared Statements With the OracleParameter Class"
- Section 14.1.5, "Large Object Support With the OracleBlob Class"
- Section 14.1.6, "Data Synchronization With the OracleSync Class"

14.1.1 Establish Connections With the OracleConnection Class

The `OracleConnection` interface establishes connections to Oracle Database Lite. This class implements the `System.data.IDBConnection` interface. When constructing an instance of the `OracleConnection` class, implement one of the following to open a connection to the back-end database:

- Pass in a full connection string as described in the Microsoft ODBC documentation for the `SQLDriverConnect` API, which is shown below:

  ```
  OracleConnection conn = new OracleConnection
       ("DataDirectory=\\orace;Database=polite;DSN=*;uid=system;pwd=manager");
  conn.Open();
  ```

- Construct an empty connection object and set the `ConnectionString` property later.

With an embedded database, we recommended that you open the connection at the initiation and leave it open for the life of the program. When you close the connection, all of the `IDataReader` cursors that use the connection are also closed.

14.1.2 Transaction Management

By default, Oracle Database Lite connection uses the autocommit mode. Alternatively, you can start a transaction with the `BeginTransaction` method in the `OracleConnection` object. Then, when finished, execute either the `Commit` or `Rollback` methods on the returned `IDbTransaction`, which either commits or rolls back the transaction. Once the transaction is completed, the database is returned to autocommit mode.

Within the transaction, use SQL syntax to set up, remove and undo savepoints.

For Microsoft Pocket PC-based devices, Oracle Database Lite supports only one process to access a given database. When a process tries to connect to a database that is already in use, the `OracleConnectionOpen` method throws an `OracleException`. To avoid this exception being thrown, close a connection to allow another process to connect.

14.1.3 Create Commands With the OracleCommand Class

The `OracleCommand` class implements the `System.Data.IDBCommand` interface. Create any commands through the `CreateCommand` method of the `OracleConnection` class. The `OracleCommand` has constructors recommended by the ADO.NET manual, such as `OracleCommand(OracleConnection conn, string cmd)`.

However, if you use the `OracleCommand` constructors, it is difficult to port the code to other platforms, such as the ODBC provider on Windows 32. Instead, create the connection and then use interface methods to derive other objects. With this model, you can either change the provider at compile time or use the reflection API at runtime.

14.1.4 Maximize Performance Using Prepared Statements With the OracleParameter Class

Parsing a new SQL statement can take significant time; thus, use prepared statements for any performance-critical operations. Although, `IDbCommand` has an explicit `Prepare` method, this method always prepares a statement on the first use. You can reuse the object repeatedly without needing to call `Dispose` or change the `CommandText` property.

14.1.4.1 SQL String Parameter Syntax

Oracle Database Lite uses ODBC-style parameters in the SQL string, such as the `?` character. Parameter names and data types are ignored by the driver and are only for the programmer's use.

For example, assume the following table:

```
create table t1(c1 int, c2 varchar(80), c3 data)
```

You can use the following parameters in the context of this table:

```
IDbCommand cmd = conn.CreateCommand();
cmd.CommandText  = "insert into t1 values(?,?,?);"
cmd.Parameters.Add("param1", 5);
cmd.Parameters.Add("param2", "Hello");
cmd.Parameters.Add("param3", DateTime.Now);
cmd.ExecuteNonQuery();
```

Note: The relevant class names are `OracleParameter` and `OracleParameterCollection`.

14.1.5 Large Object Support With the OracleBlob Class

The `OracleBlob` class supports large objects. Create a new `OracleBlob` object to instantiate or insert a new BLOB object in the database, as follows:

```
OracleBlob blob = new OracleBlob(conn);
```

Since the BLOB is created on a connection, you can use the `Connection` property of `OracleBlob` to retrieve the current `OracleConnection`.

Functions that you can perform with a BLOB are as follows:

- Section 14.1.5.1, "Using BLOB Objects in Parameterized SQL Statements"
- Section 14.1.5.2, "Query Tables With BLOB Columns"
- Section 14.1.5.3, "Read and Write Data to BLOB Objects"

14.1.5.1 Using BLOB Objects in Parameterized SQL Statements

You can use the BLOB object in parameterized SQL statements, as follows:

```
OracleCommand cmd = (OracleCommand)conn.CreateCommand();
cmd.CommandText = "create table LOBTEST(X int, Y BLOB)";
cmd.ExecuteNonQuery();
cmd.CommandText = "insert into LOBTEST values(1, ?)";
cmd.Parameters.Add(new OracleParameter("Blob", blob));
cmd.ExecuteNonQuery();
```

14.1.5.2 Query Tables With BLOB Columns

You can retrieve the `OracleBlob` object using the data reader to query a table with a BLOB column, as follows:

```
cmd.CommandText = "select * from LOBTEST";
IDataReader rd = cmd.ExecuteReader();
rd.read();
OracleBlob b = (Blob)rd["Y"];
```

Or you can write the last line of code, as follows:

```
OracleBlob b = (OracleBlob)rd.getvalue(1);
```

14.1.5.3 Read and Write Data to BLOB Objects

The `OracleBlob` class supports reading and writing to the underlying BLOB, and retrieving and modifying the BLOB size. Use the `Length` property of `OracleBlob` to get or to set the size. Use the following functions to read and write to the BLOB, as follows:

```
public long GetBytes(long blobPos, byte [] buf, int bufOffset, int len);
public byte [] GetBytes(long blobPos, int len);
public void SetBytes(long blobPos, byte [] buf, int bufOffset, int len);
public void SetBytes(long blobPos, byte [] buf);
```

For example, the following appends data to a BLOB and retrieves the bytes from position five forward:

```
byte [] data = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };
 blob.SetBytes(0, data); //append data to the blob
byte [] d = blob.GetBytes(5, (int)blob.Length - 5); //get bytes from position 5 up
to the end
blob.Length = 0; //truncate the blob completely
```

Use the `GetBytes` method of the data reader to read the BLOB sequentially, but without accessing it as a `OracleBlob` object. You should not, however, use the `GetBytes` method of the reader and retrieve it as a `OracleBlob` object at the same time.

14.1.6 Data Synchronization With the OracleSync Class

You can perform a synchronization programatically with one of the following methods:

- Section 14.1.6.1, "Using the OracleSync Class to Synchronize"
- Section 14.1.6.2, "Using the OracleEngine to Synchronize"

14.1.6.1 Using the OracleSync Class to Synchronize

To programmatically synchronize databases, perform the following:

1. Instantiate an instance of the `OracleSync` class.
2. Set relevant properties, such as username, password and URL.
3. Call the `Synchronize` method to trigger data synchronization.

This is demonstrated in the following example:

```
OracleSync sync = new OracleSync();
sync.UserName = "JOHN";
sync.Password = "JOHN";
sync.ServerURL = "mobile_server";
sync.Synchronize();
```

The attributes that you can set are described in Table 14–1.

Table 14–1 OracleSync Attributes

Attibute	Description
`UserName`	Assign a string in quotes with the name of the user for synchronization.
`Password`	Assign a string in quotes with the password for the user.
`ServerURL`	Assign a string in quotes with the Mobile Server host name.
`ProxyHost`	Assign a string in quotes with the host name of the proxy server.
`ProxyPort`	Assign a string in quotes with the port of the proxy server.
`Secure`	Set to true if using SSL; false if not.
`PushOnly`	If true, upload changes from the client to the server only, as download is not allowed. This is useful when the data transfer is a one way transmission from the client to server.
`HighPriority`	Set to true if requesting a high priority synchronization.

Table 14–1 (Cont.) OracleSync Attributes

Attibute	Description

`SetTableSyncFlag`	Three arguments are required for `SetTablesyncFlag`, as follows:
	`sync.SetTableSyncFlag (String pub_name,`
	`String tbl_name, boolean remove)`
	Passing `pub_name`, null `tbl_name`, `remove` = 0 turns off `syncFlag` for everytable in that publication. Passing `pub_name`, `tbl_name`, `remove` = 1 turns on `syncFlag` for that specific table. Thus, you can set synchronization off for all tables, then turn on each individual table that you want to synchronize.

If you want to retrieve the synchronization progress information, set the `SyncEventHandler` attribute of the `OracleSync` class before your execute the `sync.synchronize` method, as follows.

```
sync.SetEventHandler (new OracleSync.SyncEventHandler
                              (MyProgress), true);
```

You pass in your implementation of the `MyProgress` method, which has the following signature:

```
Void MyProgress(SyncStage stage, int Percentage)
```

14.1.6.2 Using the OracleEngine to Synchronize

You can synchronize with the same engine that performs the synchronization for the `msync` tool. You can actually launch the GUI to have the user enter information and click **Synchronize** or you can enter the information programmatically and synchronize without launching the GUI.

14.1.6.2.1 Launch the msync Tool for User Input You can launch the `msync` tool, so that the user can modify settings and initialize the synchronization, by executing the following:

```
OracleEngine.Synchronize(false)
```

Providing the `false` as the input parameter tells the engine that you are not providing the input parameters, but to bring up the `msync` GUI for the user to input the information.

14.1.6.2.2 Set the Environment and Synchronize With the OracleEngine You can set the information and call for a synchronization through the `OracleEngine` class without bringing up the GUI.

If you accept the default synchronization settings, provide `true` as the input parameter to automatically synchronize, as follows:

```
OracleEngine.Synchronize(true)
```

You can execute the synchronize method with three input parameters that define a specific server: the server name, usename and password.

```
OracleEngine.Synchronize("S11U1", "manager", "myserver.mydomain.com")
```

Alternatively, you can configure a string that contains the options listed in Table 14–2 with a single `String` input parameter and synchronize, as follows:

```
OracleEngine.Synchronize(args)
```

In the above example, the `String args` input parameter is a combination of the options in Table 14–2.

```
String args = "S11U1/manager@myserver.mydomain.com /save /ssl /force"
```

Include as many of the options that you wish to enable in the `String`.

Table 14–2 Command Line Options

Option	Description
`username/password@server[:port] [@proxy:port]`	Automatically synchronize to the specified server.
`/a`	Automatically synchronize to saved preferred server.
`/save`	Save user info and exit.
`/proxy:(proxy_server)[:port]`	Connect by specific proxy server and port.
`/ssl`	Synchronize with SSL encryption.
`/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.

14.1.7 Creating a Database for Testing

In a non-production environment, you may want to create a database to test your ADO.NET application against. In the production environment, the database is created when you perform the `OracleEngine.Synchronize` method (see Section 14.1.6.2, "Using the OracleEngine to Synchronize" for more information). However, to just create the database without synchronization, you can use the `CreateDatabase` method of the `OracleEngine` class. To remove the database after testing is complete, use the `RemoveDatabase` method. These methods are only supported when you install the Mobile Development Kit (MDK).

Note: Use the CAB file provided with the MDK.

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

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

14.2 Running the Demo for the ADO.NET Provider

This release comes with sample code that illustrates the Oracle Database Lite ADO.NET provider. The demo is a timecard application for a cable technician who might install, remove, or repair service and keep track of the hours worked. To use the Oracle Database Lite ADO.NET provider from your own project, add a reference to `Oracle.DataAccess.Lite_wce.dll`.

Perform the following to run the demo:

1. If you have not already done so, install the .NET Compact Framework on your device using `netcfsetup.msi`.
2. Install Oracle Database Lite on your device—such as the `olite.us.pocket_pc.arm.CAB`—from the following directory:

   <*ORACLE_HOME*>`\Mobile\Sdk\wince\Pocket_PC\cabfiles`
3. Open the `ClockIn_wce.csdproj` from the `ADO.NET\ADOCE\Clockin_wce` directory with Visual Studio.NET 2003. Make sure that the `Oracle.DataAccess.Lite` reference in the project points to the DLL in the `ADO.NET\ADOCE` directory.
4. Select `Deploy Application` from the `Project` menu to install the ClockIn sample application on your Pocket PC device.
5. Use the file manager to launch `msql` in the `\OraCE` directory on your device. Go to the `Tools` tab and click `Create` to create the `POLITE` database and its corresponding ODBC data source. Exit `msql`.
6. Use the file manager to start the `ClockIn` demo in the `\Program` files directory.

Choose the job type and time from the drop down lists at the bottom of the screen and Click **Add** to enter a new work item and update summary on the title bar. Click on an existing work item row to remove it. You can also navigate to a different date to review past work.

Examine the `MainForm.cs` in the `ClockIn` subdirectory. Notice the following items, which demonstrate the functionality discussed in this chapter:

1. Creating an Oracle Database Lite connection.
2. Using prepared statements and cleaning up at program exit.
3. Using LiteDataAdapter to retrieve data into disconnected ResultSet and delete an existing row.
4. Using DataGrid to display data on screen.

You can make some changes to become familiar with ADO.NET development, such as:

1. Add checking for overlapping work items and give an appropriate error.
2. Add an ability to edit an existing work item and give arbitrary start/end times and description by clicking on a row.
3. Add sync support to ClockIn. You need to define a primary key on the ClockIn table using a sequence.

14.3 Limitations for the ADO.NET Provider

The following are limitations to the Oracel Database Lite ADO.NET provider:

- Section 14.3.1, "Partial Data Returned with GetSchemaTable"

14.3.1 Partial Data Returned with GetSchemaTable

The Oracle Database Lite ADO.NET provider method—`GetSchemaTable`—only returns partial data. For example, it claims that all of the columns are primary key, does not report unique constraints, and returns null for `BaseTableName`, `BaseSchemaName` and `BaseColumnName`. Instead, to retrieve Oracle Database Lite meta information, use `ALL_TABLES` and `ALL_TAB_COLUMNS` instead of this call to get Oracle Database Lite meta information.

14.3.2 Creating Multiple DataReader Objects Can Invalidate Each Other

The Oracle Database Lite ADO.NET provider does not support multiple concurrent `DataReader` objects created from a single `OracleCommand` object. If you need more than one active `DataReader` objects at the same time, create them using separate `OracleCommand` objects.

The following example shows how if you create multiple `DataReader` objects from a single `OracleCommand` object, then the creation of `reader2` invalidates the `reader1` object.

```
OracleCommand cmd = (OracleCommand)conn.CreateCommand();
cmd.CommandText = "SELECT table_name FROM all_tables";
cmd.Prepare();
IDataReader reader1 = cmd.ExecuteReader();
IDataReader reader2 = cmd.ExecuteReader();
```

14.3.3 Calling DataReader.GetString Twice Results in a DbNull Object

Calling the `GetString` method of `DataReader` twice on the same column and for the same row results in a `DbNull` object. The following example demonstrates this in that the second invocation of `GetString` results in a `DbNull` object.

```
 IDataReader dr = cmd.ExecuteReader();
        String st = null;
        while(dr.Read())
        {
                st = dr.GetString (1);
                st = dr.GetString (1);
        }
```

14.3.4 Thread Safety

To build a thread-safe program, make sure that different threads use separate `IDbCommand` and `IDataReader` objects. The `OracleConnection` and `IDbTransaction` methods can be called concurrently, except for when used to open and close the connection.

15 Oracle Database Lite Transaction Support

When an application connects to the local client database—Oracle Database Lite—it begins a transaction with the database. There can be a maximum of 64 connections to Oracle Database Lite. Each connection to Oracle Database Lite maintains a separate transaction, which conform to ACID requirements.

A transaction can include a sequence of database operations, such as `SELECT`, `UPDATE`, `DELETE`, and `INSERT`. All operations either succeed and are committed or are rolled back. Oracle Database Lite only updates the database file when the commit is executed. If an event, such as a power outage, interrupts the commit, then the database is restored during the next connection.

- Section 15.1, "Locking"
- Section 15.2, "What Are the Transaction Isolation Levels?"
- Section 15.3, "Configuring the Isolation Level"
- Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types"

15.1 Locking

Oracle Database Lite supports row-level locking. Whenever a row is read, it is read locked. Whenever a row is modified, it is write locked. If a row is read locked, then different transactions can still read the same row. However, a transaction cannot access a row if it is a write locked row by another transaction.

15.2 What Are the Transaction Isolation Levels?

Each transaction is isolated from another. Even though many transactions run concurrently, transaction updates are concealed from other transactions until the transaction commits. You can specify what level of isolation is used within the transaction, as listed in Table 15–1:

Table 15–1 Isolation Levels

Isolation Level	Description
Read Committed	In Oracle Database Lite, a READ COMMITTED transaction first acquires a temporary database level read lock, places the result of the query into a temporary table, and then releases the database lock. During this time, no other transaction can perform a commit operation. No data objects are locked. All other transactions are free to perform any DML operation—except commit—during this time. Since a commit operation locks the database in intent exclusive mode, a read committed transaction, while gathering the query result, will block another transaction that is trying to commit or vice versa. A READ COMMITTED transaction provides the highest level of concurrency, as it does not acquire any data locks and does not block any other transaction from performing any DML operations. In addition, the re-execution of the same query (SELECT statement) may return more or less rows based on other transactions made to the data in the result set of the query. **Note**: A SELECT statement containing the FOR UPDATE clause is always executed as if it is running in a REPEATABLE READ isolation level. A SELECT statement can execute Java stored procedures. If the transaction executing the Java stored procedure is in the READ COMMITTED isolation level and the Java stored procedure updates the database, then the SELECT statement that executes the Java stored procedure must have a FOR UPDATE clause. Otherwise, Oracle Database Lite issues an error.
Repeatable Read	In this isolation level, a query acquires read locks on all of the returned rows. More rows may be read locked because of the complexity of the query itself, the indexes defined on its tables, or the execution plan chosen by the query optimizer. The REPEATABLE READ isolation level provides less concurrency than a READ COMITTED isolation level, transaction because the locks are held until the end of the transaction. A "phantom" read is possible in this isolation level, which can occur when another transaction inserts rows that meet the search criteria of the current query and the transaction re-executes the query. If a FOR UPDATE clause is used in a query, a short-term update lock is acquired on the current row(s) being selected. If a row is updated, the lock is converted into an exclusive lock. An exclusive lock prevents any other transaction running in an isolation level other than READ COMMITTED to access this row. If the row is not updated but the next row is fetched, the update lock is downgraded to a read lock, permitting other transactions to read the row.
Serializable	This isolation level acquires shared locks on all tables participating in the query. The same set of rows is returned for the repeated execution of the query in the same transaction. Any other transaction attempting to update any rows in the tables in the query is blocked.
SingleUser	In this isolation level only one connection is permitted to the database. The transaction has no locks and consumes less memory.

Refer to the documentation for ODBC for more information on isolation levels.

15.3 Configuring the Isolation Level

The default isolation level is READ COMMITTED. You can modify the isolation level for a data source name (DSN) by using the ODBC Administrator—which you can bring up by executing odbcad32—or by manually editing the ODBC.INI file. We recommend that you use the odbcad32 tool, as it will inform you if you have an incorrect combination of isolation level and cursor type. See Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types" for more information.

When you bring up the ODBC Administrator, under the User DSN tab, double-click the Oracle Lite 40 ODBC driver for which you want to modify the isolation level. Select the default cursor type from the pull-down list.

If you decide to edit the `ODBC.INI` file by hand, then set the isolation level as follows:

```
IsolationLevel = XX
```

where the value for `XX` is Read Committed, Repeatable Read, Serializable, or Single User.

Alternatively, you can define the isolation level of a transaction by using the following SQL statement:

```
SET TRANSACTION ISOLATION LEVEL <ISOLATION_LEVEL>;
```

where `ISOLATION_LEVEL` is `READ COMMITTED`, `REPEATABLE READ`, `SERIALIZABLE`, or `SINGLE USER`.

See Section 15.4, "Supported Combinations of Isolation Levels and Cursor Types", for information on how certain isolation levels and scrollable cursors sometimes cannot be used in combination.

15.4 Supported Combinations of Isolation Levels and Cursor Types

If you use the ODBC Administrator—which you can bring up by executing `odbcad32`—then this tool informs you if you are using an incorrect combination of isolation level and cursor type.

We support these types of cursors

- Forward only cursors allow you to only move forward through the returned result set. You cannot go backwards, nor can you view any additional modifications. To return to a row, you would have to close the cursor, reopen it and then move to the row you wanted to see. However, it is the fastest cursor for moving through a result set.
- Scrollable cursors are the most flexible as they allow you to go forward and backward through the returned result set, but are also expensive. The other advantage of using a scrollable cursor is you can see modifications directly after they occur.

The three supported types of scrollable cursors are as follows:

- Static—The result set appears to be static; that is, it does not detect modifications made to the membership, order, or values of the result set after the cursor is opened. This cursor can detect its own modifications, just not the modifications of others.
- Dynamic—Any modifications to the result set can be detected and viewed when the row is re-fetched.
- Keyset Driven—The abilities of this cursor is between the static and dynamic. It can detect modifications to the values in the rows of the result set; however, it cannot detect changes to the membership and order of the result set.

Refer to the documentation for ODBC for more information on cursor types.

For some cursors, you cannot combine them with certain isolation levels. Table 15–2 shows the supported combinations of isolation levels and cursor types. Unsupported combinations generate error messages.

Table 15–2 Supported Combinations

	Forward Only Cursor	**Scrollable Static Cursor**	**Scrollable Keyset Driven Cursor**	**Scrollable Dynamic Cursor**
Isolation Level				
Read Committed	Supported	Supported	Unsupported	Unsupported
Repeatable Read	Supported	Unsupported	Supported	Supported
Serializable	Supported	Unsupported	Supported	Supported
Single User	Supported	Supported	Supported	Supported

16 Oracle Database Lite Development Performance Considerations

The following sections describe the methods you can manage the performance of your use of Oracle Lite Database:

- Section 16.1, "Increasing Synchronization Performance"
- Section 16.2, "Determine Execution of SQL Query With EXPLAIN PLAN"
- Section 16.3, "Synchronization Performance Tuning for the Mobile Server"

16.1 Increasing Synchronization Performance

The following sections describe how you can manipulate the synchronization performance:

- Section 16.1.1, "Synchronization Disk Needs May Impose on WinCE Platform Available Space"
- Section 16.1.2, "Shared Maps"
- Section 16.1.3, "Priority-Based Replication"
- Section 16.1.4, "Caching Publication Item Queries"
- Section 16.1.5, "Use Map Table Partitions to Streamline Users Who Subscribe to a Large Amount of Data"

16.1.1 Synchronization Disk Needs May Impose on WinCE Platform Available Space

During synchronization, files are created within the Mobile Server directories for synchronization management. This may cause space problems on the WinCE device. To counter space constraints for the storage card on the WinCE platform, you can designate the `Temp` directory for all synchronization temporary files by adding the following entry in the `ALL DATABASES` section in the `POLITE.INI` or `POLITE.TXT` file.

```
TEMP_DIR=\Storage Card\Temp
```

16.1.2 Shared Maps

It is very common for publications to contain publication items that are used specifically for lookup purposes. In other words, the server may change these snapshots, but the client would never update them directly. Furthermore, many users often share the data in this type of snapshot. For example, there could be a publication item called `zip_codes`, which is subscribed to by all Mobile users. The main function of Shared Maps is to improve scalability for this type of publication item by allowing users to share record state information and, thus, reduce the size of the resulting replication map tables.

Shared Maps can also be used with updatable snapshots if the developer is willing to implement their own conflict detection and resolution logic.

This section discusses the shared maps feature in terms of concepts and performance attributes.

16.1.2.1 Concepts

Shared maps shrink the size of map tables for large lookup publication items and reduce the MGP compose time. Lookup publication items contain "lookup" data that is not updatable on the clients and that is shared by multiple subscribed clients. When multiple users share the same data, their query subsetting parameters are usually identical.

For example, a query could be the following:

```
SELECT * FROM  WHERE EMP WHERE DEPTNO = :dept_id
```

In the preceding example, all users that share data from the same department have the same value for `dept_id`. The default sharing method is based on subscription parameter values.

In the following example, the query is:

```
SELECT * FROM WHERE EMP WHERE DEPTNO = ( SELECT DEPTNO FROM
        EMP WHERE EMPNO = :emp_id )
```

In this example, users from the same departments still share data. Their subsetting parameters, however, are not equal because each user has a unique `emp_id`. To support the sharing of data for these types of queries (as illustrated by the example), a grouping function can be specified. The grouping function returns a unique group `id` based on the client `id`.

There is also another possible use for shared maps. It is possible to use shared maps for shared updatable publication items. This type of usage, however, requires implementation of a custom **dml** procedure that handles conflict resolution.

16.1.2.2 Performance Attributes

The performance of the MGP compose cycle is directly proportional to:

```
NC * NPI
```

where:

`NC` = number of clients.

`NPI` = number of publication items that must be composed.

With shared maps, the length of the MGP cycle is proportional to: `NC*(NPI - NSPI) + NG*NSPI`

where:

`NSPI` = number of shared publication items.

`NG` = number of groups.

Note that if `NG = NC`, the MGP performance is similar in both cases. However, with fewer groups and more shared publication items, the MGP compose cycle becomes faster.

Also note that map storage requirements are governed by the same factors.

16.1.2.3 Usage

To set up a publication item to be shared, use the `AddPublicationItem` API and enable the shared flag. It is also possible to toggle the shared property of a publication item once it is added to the publication with the `SetPublicationItemMetadata` API. Both the `AddPublicationItem` API and the `SetPublicationItemMetadata` API allow users to specify a PL/SQL grouping function. The function signature must be the following:

```
(
CLIENT in VARCHAR2,
PUBLICATION in VARCHAR2,
ITEM in VARCHAR2
)return VARCHAR2.
```

The returned value must uniquely identify the client's group. For example, if client **A** belongs to the group **GroupA** and client **B** belongs to the group **GroupB**, the group function **F** could return:

```
F ('A','SUBSCRIPTION','PI_NAME') = 'GroupA'
```

```
F ('B','SUBSCRIPTION','PI_NAME') = 'GroupB'
```

The implicit assumption of the grouping function is that all the members of the **GroupA** group share the same data, and that all the members of the **GroupB** group share the same data.. The group function uniquely identifies a group of users with the same data for a particular **PUBLICATION ITEM**.

For the query example in Section 16.1.2.1, "Concepts", the grouping function could be:

```
Function get_emp_group_id (
       clientid in varchar2,
       publication in varchar2,
       item in varchar2
) return varchar2 is
       group_val_id varchar2(30);
begin
       select DEPTNO into group_val_id
       from EMP where EMPNO = clientid ;
       return group_val_id;
end;
```

NOTE: This function assumes that `EMPNO` is the Consolidator Manager client id. If the `group_fnc` is not specified, the default grouping is based on subscription parameters.

16.1.2.4 Compatibility and Migration

Shared maps are not compatible with raw id based clients prior to 5.0.2.

Those clients are supported; however, the map data is private until the clients migrate to 5.0.2 or later.

The migration of the existing mobile server schema to 10*g* must be done in the following steps to minimize the number of client complete refreshes.

1. Run one cycle of MGP.
2. The clients must sync with the server to get the latest changes prepared by the MGP.

3. Stop the Web server and MGP to migrate the server to 10*g*. This automatically sets all the nonupdatable publication items to shared items. If any shared publication items need to use grouping functions or any publication items need to change their sharing attribute, execute custom code that calls the appropriate Consolidator Manager API. See the `SetPublicationItemMetadata` API in Section 16.1.2.3, "Usage".
4. The `ShrinkSharedMaps` Consolidator Manager API must be called to set the clients to use shared map data and remove old redundant data from the maps.
5. Start the Web server and MGP.

16.1.3 Priority-Based Replication

With priority-based replication, you can limit the number of rows per snapshot by setting the flag **Priority** to 1 (the default is 0).

For example, if you have a snapshot with the following statement:

```
select * from projects where prio_level in (1,2,3,4)
```

With the **Priority** flag set to 0 (the default), all projects with **prio_level** 1,2,3,4 will be replicated.

In a high priority situation, the application can set the flag to 1, which will cause MGP to check for **Restricting Predicate**. A Restricting Predicate is a conditional expression in SQL. The developer can set Restricting Predicate in the **AddPublicationItem()** method, as in the following example:

```
prio_level = 1
```

MGP appends (AND) the expression to the snapshot definitions when composing data for the client. In this case, the high priority statement would be:

```
SELECT * FROM projects where prio_level in (1,2,3,4) AND prio_level = 1;
// a restricting predicate snapshot
```

In this case, only projects with level =1 will be replicated to the client.

This advanced feature is available only through the Consolidator Manager API. It is not available through the Packaging Wizard.

To summarize, there are two steps to enable this feature:

1. Provide a restricting predicate expression in the **AddPublicationItem()** function.
2. Set the PRIORITY flag to 1 in the Mobile Sync API.

Note: You cannot use fast refresh synchronization with high priority.

16.1.4 Caching Publication Item Queries

This feature allows complex publication item queries to be cached. This applies to queries that cannot be optimized by the Oracle query engine. By caching the query in a temporary table, the Sync Server template can join to the snapshot more efficiently.

Storing the data in a temporary table does result in additional overhead to MGP operation, and the decision to use it should only be made after first attempting to optimize the publication item query to perform well inside the Sync Server template. If the query cannot be optimized in this way, the caching method should be used.

The following example is a template used by the MGP during the compose phase to identify client records that are no longer valid, and should be deleted from the client:

```
UPDATE pub_item_map map
SET delete = true
WHERE client = <clientid>
AND NOT EXISTS (SELECT 'EXISTS' FROM
    (<publication item query>) snapshot
     WHERE map.pk = snapshot.pk);
```

In this example, when `<publication item query>` becomes too complex, because it contains multiple nested subqueries, unions, virtual columns, connect by clauses, and other complex functions, the query optimizer is unable to determine an acceptable plan. This can have a significant impact on performance during the MGP compose phase. Storing the publication item query in a temporary table, using the publication item query caching feature, flattens the query structure and enables the template to effectively join to it.

16.1.4.1 Enabling Publication Item Query Caching

The following API enables publication item query caching.

Syntax

```
public static void enablePublItemQueryCache(String name)
      throws Throwable
```

The parameters for `enablePublItemQueryCache` are listed in Table 16–1:

Table 16–1 The enablePubItemQueryCache Parameters

Parameters	Description
`name`	A string specifying the name of the publication item.

Example

```
consolidatorManager.enablePubItemQueryCache("P_SAMPLE1");
```

If you are using an input string from the input parameter `argv` array, cast it to a `String`, as follows:

```
consolidatorManager.enablePubItemQueryCache( (String) argv[0]);
```

16.1.4.2 Disabling Publication Item Query Caching

The following API disables publication item query caching.

Syntax

```
public static void disablePubItemQueryCache(String name)
      throws Throwable
```

The name parameter for `disablePubItemQueryCache` is listed in Table 16–2:

Table 16–2 The disablePubItemQueryCache Parameters

Parameters	Description
`name`	A string specifying the name of the publication item.

Example

```
consolidatorManager.disablePubItemQueryCache("P_SAMPLE1");
```

16.1.5 Use Map Table Partitions to Streamline Users Who Subscribe to a Large Amount of Data

Sync Server database objects called map tables are used to maintain the state for each Mobile client. If there are a large number of clients, and each client subscribes to a large amount of data, the map tables can become very large creating scalability issues. Using the following APIs, map tables can be partitioned by client id, making them more manageable.

The API allows you to create a map table partition, add additional partitions, drop one or all partitions, and merge map table partitions. Map table partitions can be monitored using the `ALL_PARTITIONS` database catalog view.

Note: This form of partitioning is not related to the partition functionality provided by Oracle Server, and is used exclusively by Oracle Database Lite 10*g*.

16.1.5.1 Create a Map Table Partition

Creates a partition for the referenced publication item map table. If there is data in the map table, it is transferred to the partition being created. After the partition has been successfully created, the map table can be truncated to remove redundant data using the SQL command `TRUNCATE TABLE`.

Note: Records removed from the server through a `truncate` command will not be removed from the client unless a complete refresh is triggered. The truncate command is considered a DDL operation. Consequently, the necessary DML triggers do not fire and therefore the operations are not logged for fast refresh.

Syntax

```
public static void partitionMap
   (String pub_item,
    int num_parts,
    String storage,
    String ind_storage) throws Throwable
```

The parameters of `partitionMap` are listed in Table 16–3.

Table 16–3 The partitionMap Parameters

Parameter	Definition
`pub_item`	The publication item whose map table is being partitioned.
`num_parts`	The number of partitions.
`storage`	A string specifying the storage parameters. This parameter requires the same syntax as the SQL command `CREATE TABLE`. See the *Oracle SQL Reference* for more information.

Table 16–3 (Cont.) The partitionMap Parameters

Parameter	Definition
`ind_storage`	A string specifying the storage parameters for indexes on the partition. This parameter requires the same syntax as the SQL command `CREATE INDEX`. See the *Oracle SQL Reference* for more information.

Example

```
consolidatorManager.partitionMap("P_SAMPLE1", 5, "tablespace mobileadmin",
"initrans 10 pctfree 70");
```

16.1.5.2 Add Map Table Partitions

Adds a partition for the referenced publication item's map table. If there is data in the map table, it is transferred to the partition being created. After the partition has been successfully created, the map table can be truncated to remove redundant data using the SQL command `TRUNCATE TABLE`.

> **Note:** Records removed from the server through a `truncate` command will not be removed from the client unless a complete refresh is triggered. The truncate command is considered a DDL operation. Consequently, the necessary DML triggers do not fire and therefore the operations are not logged for fast refresh.

Syntax

```
public static void addMapPartitions
   ( String pub_item,
    int num_parts,
    String storage,
    String ind_storage) throws Throwable
```

The parameters of `addMapPartitions` are listed in Table 16–4:

Table 16–4 The addMapPartitions Parameters

Parameter	Definition
`pub_item`	The publication item whose map table is being partitioned.
`num_parts`	The number of partitions.
`storage`	A string specifying the storage parameters. This parameter requires the same syntax as the SQL command `CREATE TABLE`. See the *Oracle Database Lite SQL Reference* for more information.
`ind_storage`	A string specifying the storage parameters for indexes on the partition. This parameter requires the same syntax as the SQL command `CREATE INDEX`. See the *Oracle Database Lite SQL Reference* for more information.

Example

```
consolidatorManager.addMapPartitions("P_SAMEPLE1",5,"tablespace
mobileadmin","initrans 10 pctfree 40");
```

Note: Map Partitions are created only for existing users. New users are placed in the original map table.

16.1.5.3 Drop a Map Table Partition

Drops the named partition. In the following example, the `partition` parameter is the name of the partition. Partition names must be retrieved by querying the `ALL_PARTITIONS` table view `CV$ALL_PARTITIONS` since partitions are named by Data Synchronization.

Syntax

```
public static void dropMapPartition( String partition) throws Throwable
```

Example

```
consolidatorManager.dropMapPartition("MAP101_1");
```

16.1.5.4 Drop All Map Table Partitions

Drops all partitions of the map table for the named publication item.

Syntax

```
public static void dropAllMapPartitions( String pub_item) throws Throwable
```

Example

```
consolidatorManager.dropAllMapPartitions("P_SAMPLE1");
```

16.1.5.5 Merge Map Table Partitions

Merges the data from one partition into another. Partition names must be retrieved by querying the `ALL_PARTITIONS` table view `CV$ALL_PARTITIONS`, since partitions are named by Data Synchronization.

Syntax

```
public static void mergeMapPartitions
   ( String from_partition,
    String to_partiton) throws Throwable
```

Example

```
consolidatorManager.mergeMapPartition(""MAP101_1", "MAP101_2");
```

16.2 Determine Execution of SQL Query With EXPLAIN PLAN

If you want to access data on the local client Oracle Lite database, then you can use the EXPLAIN PLAN to determine the performance of your SQL query execution on the Oracle Lite database. To execute a SQL statement, Oracle might need to perform several steps. Each of these steps either physically retrieves rows of data from the database or prepares them in some way for the user issuing the statement. The combination of the steps Oracle uses to execute a statement is called an execution plan, which includes an access path for each table that the statement accesses and an ordering of the tables (the join order) with the appropriate join method. The execution plan shows you exactly how Oracle Database Lite executes your SQL statement.

The components of an execution plan include the following:

- An ordering of the tables referenced by the statement.

- An access method for each table mentioned in the statement.
- A join method for tables affected by join operations in the statement.

The EXPLAIN PLAN command stores the execution plan chosen by the Oracle Database Lite optimizer for SELECT, UPDATE, INSERT, and DELETE statement. See the section "Tuning SQL Statement Execution with the EXPLAIN PLAN" in the *Oracle Database Lite SQL Reference* for full details on how to manually create an EXPLAIN PLAN.

16.3 Synchronization Performance Tuning for the Mobile Server

The following sections provide details on how to tune the synchronization on the Mobile Server for your applications:

- Section 16.3.1, "Tuning Queries to Manage Synchronization Performance"
- Section 16.3.2, "Architecture Design of Mobile Server and Oracle Database for Synchronization Performance"\
- Section 16.3.3, "Configuring Back-End Oracle Database to Enhance Synchronization Performance"
- Section 16.3.4, "Configure Your Mobile Server to Increase Synchronization Performance"
- Section 16.3.5, "Designing Application Tables and Indexes for Synchronization Performance"

16.3.1 Tuning Queries to Manage Synchronization Performance

You can increase synchronization performance by monitoring the performance of the SQL queries in your applications. The following sections provide details on how to tune your queries:

- Section 16.3.1.1, "Avoid Using Non-Mergable Views"
- Section 16.3.1.2, "Tune Queries With Consperf Utility"
- Section 16.3.1.3, "Manage the Query Optimizer"

16.3.1.1 Avoid Using Non-Mergable Views

You should avoid using database query constructs that prevent a view from being mergable, as publication item queries that use non-mergable views do not perform well. Examples of such constructs are union, minus, and connect by. For more information on mergable views, see the Oracle database documentation.

16.3.1.2 Tune Queries With Consperf Utility

Once you have defined your application, use the `consperf` utility to profile the performance of the publication item queries. Mobile Server does not execute your publication item queries directly; instead the query is wrapped into a template query, which is executed by Mobile Server. The template query may have an unexpected query execution plan, resulting in poor performance. The `consperf` utility generates an EXPLAIN PLAN execution plan for those template queries, allowing you to tune your publication item query for best performance. In addition, consperf generates timing information for the execution of all template queries, so that you can identify bottleneck queries. For more information on the consperf utility, see the "Analyzing Performance of Publications With the Consperf Utility" section in the Synchronization chapter in the *Oracle Database Lite Administration and Deployment Guide*.

16.3.1.3 Manage the Query Optimizer

You must make sure that the optimizer picks the correct execution path when you either are using the cost-based optimizer or you have set the optimizer settings to `choose`. The optimizer can pick the correct execution path only when all of the tables are properly analyzed and statistics have been gathered for these tables.

The Mobile Server uses temporary tables during synchronization. Once a number of users have been created, and they have synchronized with Mobile Server, run consperf with the `gatherstats` option to generate the statistics information for the temporary tables. For more information on the consperf utility, see the "Analyzing Performance of Publications With the Consperf Utility" section in the Synchronization chapter in the *Oracle Database Lite Administration and Deployment Guide*.

16.3.2 Architecture Design of Mobile Server and Oracle Database for Synchronization Performance

It is recommended that you run Mobile Server and the Oracle database on different machines. If possible, use multi-CPU machines for both Mobile Server and the Oracle database. Run the Oracle database should in dedicated server mode; use of the multi-threaded server is not recommended.

16.3.3 Configuring Back-End Oracle Database to Enhance Synchronization Performance

You can configure the Oracle Database in such a way as to enchance your Mobile Server synchronization performance, as follows:

- Section 16.3.3.1, "Tablespace Layout"
- Section 16.3.3.2, "Database Parameter Tuning"

16.3.3.1 Tablespace Layout

Proper tablespace layout across multiple disks can significantly improve the performance of Mobile Server data synchronization, as it reduces movement of the disk heads and improves I/O response time.

- Section 16.3.3.1.1, "SYNCSERVER Tablespace"
- Section 16.3.3.1.2, "Map Tables and Indexes"

16.3.3.1.1 SYNCSERVER Tablespace The SYNCSERVER tablespace is the default tablespace. You can precreate this tablespace yourself and decide where it resides. For more information, see the `SYNCSERVER` Tablespace Layout section in the Performance chapter in the *Oracle Database Lite Administration and Deployment Guide*.

16.3.3.1.2 Map Tables and Indexes During synchronization, map tables are used extensively. Map tables are internal tables, and have table names using the following pattern: `CMP$pub_item_name`. Each map table has four separate indexes. By default, both map table and indexes are created in the default tablespace `SYNCSERVER`.

You can improve performance if you move the map table indexes to a different disk than the map table itself. Create a separate tablespace (for example: `MAPINDEXES`) on a different disk and manually move all indexes. Because the process of moving the indexes requires you to drop and re-create the indexes, you should move the index before many users have synchronized. Otherwise recreating the indexes on the map tables may be very time consuming, as map tables grow with the number of users who have synchronized.

To move the indexes on a map table, do the following:

1. Identify all indexes on the map table (`CMP$pub_item_name`). There are three or four indexes. Move all of them.
2. For each index, record the type of index and column lists.
3. If the index is a primary key index, then remove the primary key constraint on the map table.
4. Drop the index.
5. Recreate the index using the same name, type and column list. Use the storage clause in the create index statement to specify the new tablespace. You may also specify different storage parameters. Refer to the Oracle database documentation for more information on how to create indexes and storage clause parameters.

Note: Repeat step 3 through 5 for all other indexes on the map table.

16.3.3.2 Database Parameter Tuning

Tuning the database for Mobile Server is similar to any Oracle database tuning required for any query intensive applications. Configure the SGA to be as large as possible on your system to maximize the caching capabilities and avoid I/O wherever possible.

Tune your Oracle database with the following database parameters:

- `db_block-buffers`
- `sort_area_size`
- `log_buffers`

Refer to the Oracle database tuning guide for more information on database tuning.

16.3.4 Configure Your Mobile Server to Increase Synchronization Performance

There are two parameters in the `[CONSOLIDATOR]` section of the `webtogo.ora` file for tuning synchronization.

- `MAX_THREADS`

  The `MAX_THREADS` parameter is used by the MGP and controls the number of concurrent threads. As a rule, do not set this higher than 1.5 times the number of CPUs on the database machine. For example, if your system has four CPUs, theb you should not set it higher than six.

- `MAX_CONCURRENT`

  The `MAX_CONCURRENT` parameter controls how many users can synchronize in parallel. Once the maximum number of concurrent synchronization requests is reached, additional requests block until one or more synchronization requests completes. If you do not set this parameter, then there is no maximum.

Each synchronization request requires a number of system resources, such as creating a database connection, using memory, and so on. If you have too many requests competing for the same resources, then the overall performance can be poor. Limiting the number of parallel requests improves the average response time.

Set this parameter if you notice that the synchronization performance is not linear. For example, if twice the number of parallel requests results in a synchronization time that is five times longer for each client, then you probably have resource contention. The value depends on your environment and should be determined on a trial and error basis.

16.3.5 Designing Application Tables and Indexes for Synchronization Performance

Your clients may perform a large number of insert and delete operations on snapshots, and then synchronize their data changes with the Mobile Server. If this is the case, then consider placing the application tables and the indexes on those tables on separate disks.

17 Oracle Database Lite Security

The following sections detail security issues for Oracle Database Lite:

- Section 17.1, "Authenticating Users With Your Own User Management System"
- Section 17.2, "Providing Your Own Encryption Module for the Client Oracle Lite Database"

17.1 Authenticating Users With Your Own User Management System

You can provide an external authenticator for the Mobile Server to authenticate users with passwords as well as their access privileges to applications. For example, in an enterprise environment, you may have your user data, such as employee information, stored in a LDAP-based directory service. The Mobile Server can retrieve the user information from the LDAP directory—or from any custom User Management System—if configured with your own implementation of an external authenticator. The Mobile Server links the external user information to the Mobile Server repository.

17.1.1 Implementing Your External Authenticator

In order to use an external authenticator, you must implement the `oracle.lite.provider.Authenticator` JAVA interface and configure the implementation in the `webtogo.ora` file.

Note: Sample code for an external authenticator can be found in `<`*`ORACLE_HOME`*`>\Mobile\Server\samples\devmgr\java\SampleAuthenticator.java`

Implement the following methods in your external authenticator. The Mobile Server invokes each of these methods as appropriately.

- The Initialization Method for the External Authenticator
- The Destruction Method for the External Authenticator
- The Authentication Method for the External Authenticator
- The User Instantiation Method for the External Authenticator
- Retrieve the User Name or the User Global Unique ID
- Log Off User
- Change User Password

The Initialization Method for the External Authenticator

Mobile Server invokes the `initialize` method before calling any other method of provider class. This method will be called only once when the provider is initialized.

```
Method: void initialize (String metaData) throws Exception
Parameter: String metaData (Reserved for future use)
```

The Destruction Method for the External Authenticator

Mobile Server invokes the `destroy` method when the system shutdowns. Provider implementation should implement all the cleanup code in this method.

```
Method: void destroy() throws Exception
Parameter: None
```

The Authentication Method for the External Authenticator

Authenticate a user and return a session handle. The returned session handle is passed to the `logOff` method when the user logs off from the system. Note that the `logOff` method may not be called for each successful `authenticate` method call. Some of the Mobile Server clients may use the `authenticate` method to verify the user credential and not for logging on to the system.

```
Method: Object authenticate (String uid, String pwd) throws SecurityException
Parameter: User Id (or User Name) and password string
Return: Session handle or null
```

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 17.1.3, "User Initialization Scripts" for more information.

```
Method: StringBuffer getInitilizationScripts (Object sid)
Parameter: Session handle returned by 'authenticate' method
Return: 'StringBuffer' containing User's initialization scripts
```

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
```

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
```

Change User Password

```
Method: void changePassword (Object sid, String pwd) throws SecurityException
Parameter: Session handle returned by the authenticate method and new password
string
```

17.1.2 Registering External Authenticator

To register your external authenticator, modify the `webtogo.ora` file and set your external `Authenticator` JAVA class name in the `EXTERNAL_AUTHENTICATION` section, as follows:

```
[EXTERNAL_AUTHENTICATION]
CLASS  = SampleAuthenticator
EXPIRATION = 1800
```

The Mobile Server caches the user instantiated through the external authenticator for a period of time in order to improve efficiency. The default expiration time for the cached user object is 30 minutes (or 1800 seconds). Customize this value by setting a new value for the `EXPIRATION` parameter.

17.1.3 User Initialization Scripts

Mobile Server invokes the `getInitilizationScripts` 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
```

17.2 Providing Your Own Encryption Module for the Client Oracle Lite Database

The database on the Mobile client—also known as the Oracle Lite database—uses Advanced Encryption Standard (AES) for encrypting the database. However, you can provide your own encryption module for the client database. The following sections describe how to implement and plug-in your own encryption module.

- Section 17.2.1, "Encryption Module APIs"
- Section 17.2.2, "Plug-In Custom Encryption Module"

17.2.1 Encryption Module APIs

Oracle Database Lite invokes your encryption APIs when performing encryption duties, instead of the internal AES encryption module. Thus, you must develop and include the following APIs in your encryption module:

- Section 17.2.1.1, "Initializing the Encryption Module"
- Section 17.2.1.2, "Delete Encryption Context"
- Section 17.2.1.3, "Create the Encryption Key"
- Section 17.2.1.4, "Encrypt Data"
- Section 17.2.1.5, "Decrypt Data"

Note: All of the functions in this section are in Windows format. Adjust appropriately if developing on a UNIX environment.

17.2.1.1 Initializing the Encryption Module

Implement the `encCreateCtxt` function to initialize the external encryption module. Oracle Database Lite invokes this function when initializing encryption. This function returns an encryption context handle to Oracle Database Lite, which it passes back on all subsequent API calls. The context handle is displayed as a `void*`, so that you can make it any type of structure you desire.

```
__declspec(dllexport) void* encCreateCtxt()
```

17.2.1.2 Delete Encryption Context

When Oracle Database Lite is finished with the encryption module, it invokes the `encDeleteCtxt` function to delete the encryption context—which was created with the `encCreateCtxt` function.

```
__declspec(dllexport) void encDeleteCtxt(voie * ctx);
```

17.2.1.3 Create the Encryption Key

Oracle Database Lite invokes your `encCreateKey` function to create the encryption key within the encryption context, as follows:

```
__declspec(dllexport) void encCreateKey (void* ctx, const unisgned char* key, int
len, int dir);
```

Where the input parameters are as follows:

- `ctx`—The encryption context, which is created in the `encCreateCtxt` function.
- `key`—Pointer to the key to be created.
- `len`—Length of the encryption key.
- `dir`—Encryption direction or type, where 1: encryption, 2: decryption, 3: both encryption and decryption.

17.2.1.4 Encrypt Data

Oracle Database Lite invokes your `encEncryptData` function to encrypt the data that is to be sent, as follows:

```
__declspec(dllexport) void encEncryptData (void* ctx, const unisgned char* data,
int len, unsigned char* out);
```

Where the input parameters are as follows:

- `ctx`—The encryption context, which is created in the `encCreateCtxt` function.
- `data`—Pointer to the data to be encrypted.
- `len`—Length of the data in bytes.
- `out`—Output buffer.

This function returns the number of bytes copied to the output buffer.

17.2.1.5 Decrypt Data

Oracle Database Lite invokes your `encDecryptData` function to decrypt the data that it receives. This function copies the result to the output buffer.

```
__declspec(dllexport) void encDecryptData (void* ctx, const unisgned char* data,
int len, unsigned char* out);
```

Where the input parameters are as follows:

- `ctx`—The encryption context, which is created in the `encCreateCtxt` function.
- `data`—Pointer to the data to be decrypted.
- `len`—Length of the data in bytes.
- `out`—Output buffer.

This function returns the number of bytes copied to the output buffer.

17.2.2 Plug-In Custom Encryption Module

Once implemented, you can plug-in your custom encryption module by adding the `[All Databases]` section to the `POLITE.INI` configuration file. You must either implement your encryption module into a DLL for the Windows environment or into a Shared Object (.SO) for the UNIX environment.

For example, if you created the encryption module as a DLL called `my_enc.dll`, which is located in the `C:\my_dir` directory, then you would add this module as the default encryption module in the `POLITE.INI` configuration file, as follows:

```
[All Databases]
EXTERNAL_ENCRYPTION_DLL=C:\my_dir\my_enc.dll
```

18

Oracle Database Lite Tracing

The Oracle Lite database is used in conjunction with other products such as Oracle forms, SQLJ, Web Servers, and OC4J. When an unexpected error is reported by the software system, users need to identify the location and cause of the error. Errors can be caused due to problems in the application code, Oracle tools—such as forms, SQLJ, OC4J—or in the Oracle Lite database. Errors also occur in simple environments where a user application talks directly to the Oracle Lite database through JDBC or ODBC drivers. It may not be obvious which component is at fault—whether it is the user application, JDBC or ODBC drivers, or the core database runtime system.

If the optimizer spends too much time evaluating alternative plans or collecting index statistics, a query may take a long time for compilation. If the execution plan selected by the optimizer is not optimal, the query may also take a long time during execution. Based on these criteria, the tracing facility provides the compilation time and the execution plan.

The following sections describe how to set and use tracing.

- Section 18.1, "Enabling Trace Output"
- Section 18.2, "Description of Trace Information"

18.1 Enabling Trace Output

To enable Trace output, include the following line in the `POLITE.INI` configuration file.

```
OLITE_SQL_TRACE= yes
```

Note: Any value other than "yes" disables the tracing feature. The parameter value is checked once during database startup. Hence, users must set this value before connecting to the database.

When you enable tracing, the trace information is dumped to a file named `oldb_trc.txt` in the current working directory of the database process. If the file already exists, then the trace output is appended to the end. If it does not exist, then a new file is automatically created. For a database service on Windows or the Oracle Lite database daemon for a Linux platform, the current working directory is specified by the `wdir` parameter during startup of the database service or daemon.

Note: To implement the tracing feature, the database process must contain permissions to create the trace file in the current working directory.

18.2 Description of Trace Information

The following trace information is provided:

Table 18–1 Trace Output

Trace Output	Description
Statement Text	Each time a SQL statement is prepared, its text is dumped into the trace file. The SQL statement itself is output without any formatting. If a SQL statement contains a new line character, it is also included in the SQL statement output.
Compilation Time	After the SQL statement is compiled, the compilation time is printed.
Execution Plan	If there are no errors, the execution plan is printed when available. Only statements that contain a `WHERE` clause generate an execution plan. The printed plan contains the execution order of tables for each sub-select.
Bind Value	If a SQL statement contains markers, then the bind value is printed for every line. Each value for the marker or bind variable is printed on a separate line in the following format. `Marker [<number>]: <Value>` Where, `<number>` is the number of the marker and `<value>` denotes the value of the marker before execution.
Temporary Table Created	Each time a temporary table is created, its name is dumped into the trace file.
Table Access	Each time a table is accessed, the following information is dumped into the trace file: ▪ Table Name: The name of the table been accessed is dumped into the trace file. ▪ Access Method: The access method used by the database is dumped into the trace file. For a description of how this information is presented, see Section 18.2.1, "Table Name Output".
Temporary Table Sorted	Each time a temporary table is sorted, its name and sorting time (in milliseconds) are dumped into the trace file.
First Fetch Time	If the SQL statement is a SELECT statement, the time spent on fetching the first row is dumped into the trace file.
Tid	The thread ID is dumped into the trace file in front of some of the dumped information. The thread is displayed in the following format: `Tid: <thread id>`

18.2.1 Table Name Output

The name of the table that is currently being accessed and the method used to access the table are printed in the following formats.

- If the table is accessed sequentially, the format is:

Table Name: `<table name>`

Access Method: `Sequential`

Where `<table name>` is the name of the table being accessed.

- If indices are used, the format is:

  Table Name: `<table name>`

  Access Method: `Term[<number>], Index No: <index number>,`
  `IndexName: <index name>`

  `<table name>` is the name of the table being accessed.

  `Term[<number>]` is the internal representation of the conjunct search conditions in the WHERE clause.

  `<index number>` is the index number. Each index has an unique number in the database.

  `<index name>` is the name of the index if any.

Part II

Development Tutorials

This part contains tutorials that you can use to familiarize yourself with creating applications. Part II contains the following chapters:

19 Building Mobile Web Applications: A Tutorial

This tutorial guides you through the relevant phases of implementing a Web application for Mobile devices. Topics include:

- Section 19.1, "Overview"
- Section 19.2, "Developing the Application"
- Section 19.3, "Packaging the Application"
- Section 19.4, "Publishing the Application"
- Section 19.5, "Administering the Application"
- Section 19.6, "Running the Application on the Mobile Client for Web-to-Go"

19.1 Overview

This tutorial uses a simple "To Do List" application to guide you through implementing a Web application for Mobile devices. Each phase is described: creation, deployment, and administration. The "To Do List" application enables the user to maintain 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.

This overview is followed by five sections, each of which contains several topics that represent a unique phase in the life cycle of the To Do List application. When you complete each section, you can either review its contents, view related documentation, or proceed to the next one.

This tutorial uses a limited set of the functionality that is available. For a complete list of functionality and limitations, see Chapter 7, "Developing Mobile Web Applications". For more information on Oracle Database Lite concepts, refer the *Oracle Database Lite Concepts Guide*.

19.1.1 Before You Start

This tutorial assumes that you have installed and configured the Mobile Development Kit for Web-to-Go and the Mobile Server on the same computer. Before you start the tutorial, ensure that the development computer and the client computer meet the requirements specified below.

19.1.1.1 Development Computer Requirements

As Table 19–1 describes, the development computer must contain the following components.

Table 19–1 Development Computer Requirements

Requirement	Description
Windows User Login	The Windows login user on the development computer must be assigned `ADMINISTRATOR` privileges.
Installed Java Components	Java Development Kit 1.4.2 or higher.
Installed Oracle Components	Oracle Database 8.1.7 or higher.
	Mobile Server (Oracle Database Lite CD-ROM)
	The Mobile Development Kit for Web-to-Go (Oracle Database Lite CD-ROM)

19.1.1.2 Client Computer Requirements

The client computer is used to test your Mobile Web applications in online or offline mode. Using a browser, the client computer must connect to the Mobile Server over a network.

19.2 Developing the Application

This section describes how to develop and test the To Do List application, using the Mobile Development Kit for Web-to-Go. As Table 19–2 describes, the To Do List application contains the following components.

Table 19–2 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 ToDoList 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 To Do List JSP generates an HTML page which displays the list of items that must be completed. You can access the To Do List JSP from the following location.

```
<ORACLE_HOME>\mobile\sdk\wtgsdk\src\tutorial\ToDoList.jsp
```

The JavaBean

The To Do List JSP uses a JavaBean to perform operations with the Oracle database. You can access the To Do List JavaBean from the following location.

```
<ORACLE_HOME>\mobile\sdk\wtgsdk\src\tutorial\ToDoBean.java
```

The Java Servlet

The To Do List Java Servlet inserts a new To Do Item in the Oracle database, and uses the To Do List JSP to regenerate the HTML page. You can access the To Do List Java Servlet from the following location.

```
<ORACLE_HOME>\mobile\sdk\wtgsdk\src\tutorial\InsertToDo.java
```

In this section, the following tasks are discussed.

- Section 19.2.1, "Step 1: Creating Database Objects in Oracle Database Lite"
- Section 19.2.2, "Step 2: Compiling the Application"
- Section 19.2.3, "Step 3: Defining the Application and Registering the Servlet"
- Section 19.2.4, "Step 4: Conducting a Trial Run"

The Mobile Development Kit for Web-to-Go always uses Oracle Database Lite as the development database.

The Mobile Development Kit for Web-to-Go also uses a Web server that is referred to as the Mobile Client Web Server.

19.2.1 Step 1: Creating Database Objects in Oracle Database Lite

In this step, you will create the To Do List application's database objects in Oracle Database Lite.

During the development phase, the To Do List application's servlet stores the To Do items in Oracle Database Lite. Later, during the deployment phase, you will copy the database objects from Oracle Database Lite to the Oracle database.

19.2.1.1 The To Do List Application Database Objects

The To Do List application uses the following database objects.

1. The `TODO_ITEMS` table.

   The application stores To Do Items in this database table. As Table 19–3 describes, the To Do Items table contains the following columns.

Table 19–3 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

2. 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.

19.2.1.2 Required Action

Create the database objects in Oracle Database Lite using mSQL, which is an interactive tool that allows you to execute SQL statements against Oracle Database Lite. It is similar to SQL*Plus. To create the database objects, you must run the SQL script named `tutorial.sql`. Using the Command Prompt, enter the following statements.

1. `cd <ORACLE_HOME>\mobile\sdk\wtgsdk\src\tutorial`
2. `msql system/manager@jdbc:polite:webtogo @tutorial.sql`

Note: The mSQL tool requires a user name and password. Enter `system` as the User Name and substitute `manager` with any alphanumeric string.

There is a mandatory space between `webtogo` and `@tutorial.sql`.

19.2.2 Step 2: Compiling the Application

In this step, you will compile the application by performing the following tasks.

1. Set the `CLASSPATH` to include required libraries.
2. Compile the Java Servlet and JavaBean.
3. Install the JSP.

19.2.2.1 Required Action

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.

   ```
   cd <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 InsertToDo.java
      ```

      This creates the following servlet class file.

      ```
      ORACLE_
      HOME\mobile\sdk\wtgsdk\root\tutorial\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
      ```

19.2.3 Step 3: Defining the Application and Registering the Servlet

In this step, you must perform the following tasks.

- Use the Packaging Wizard to create the To Do List application.
- Add application files.
- Register the application's servlet with the Mobile Client Web Server.

In the development environment, every application and its associated servlets must be registered with the Mobile Client Web Server. You do not need to register the To Do List JSP or JavaBean.

19.2.3.1 The Packaging Wizard

As a Mobile application developer, you can use the Packaging Wizard to create or modify Web-to-Go applications. During this tutorial, you will first run the Packaging Wizard in development mode and subsequently in regular mode. In development mode, you will use the Packaging Wizard to perform the following functions.

- Define the Web-to-Go application
- Add files
- Compile JSP files
- Register Servlets

Running the Packaging Wizard in development mode disables the panels that it uses exclusively during deployment. As you will publish the application to your local machine, you do not have to enter the application's connectivity or database information in the Packaging Wizard.

For more information on how to use the Packaging Wizard, refer to Chapter 6, "Using the Packaging Wizard".

19.2.3.2 Required Action

Define the To Do List application and register its servlet by performing the following steps.

1. Start the Packaging Wizard in debug mode. Using the Command Prompt, enter the following commands.

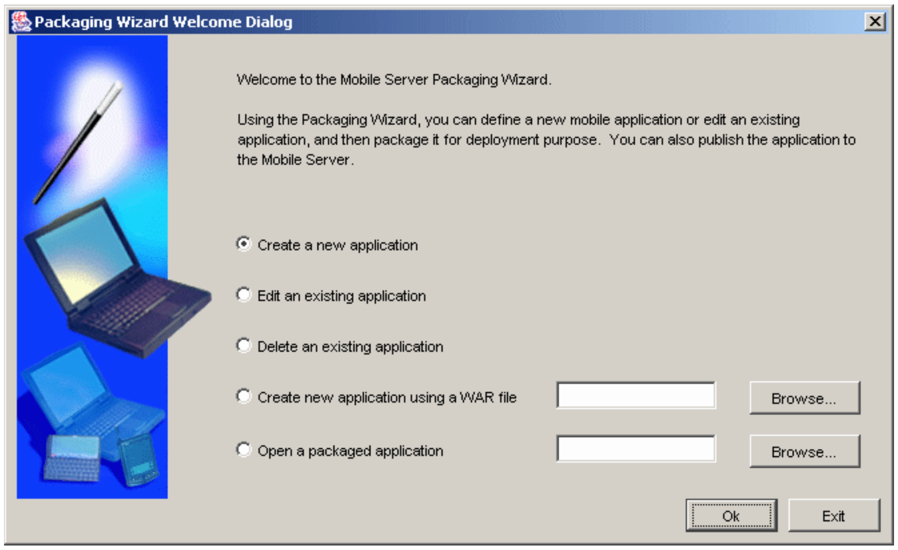
   ```
   cd <ORACLE_HOME>\mobile\sdk\bin
   ```

   ```
   wtgpack -d
   ```

   The Packaging Wizard appears and provides you with the option to create a new application, edit an existing application, delete an existing application, or open a packaged application, as displayed in Figure 19–1.

   Note: Deleting an existing application merely deletes the application from the XML file and does not remove the application from the Mobile Server.

Figure 19–1 Make a Selection Dialog


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

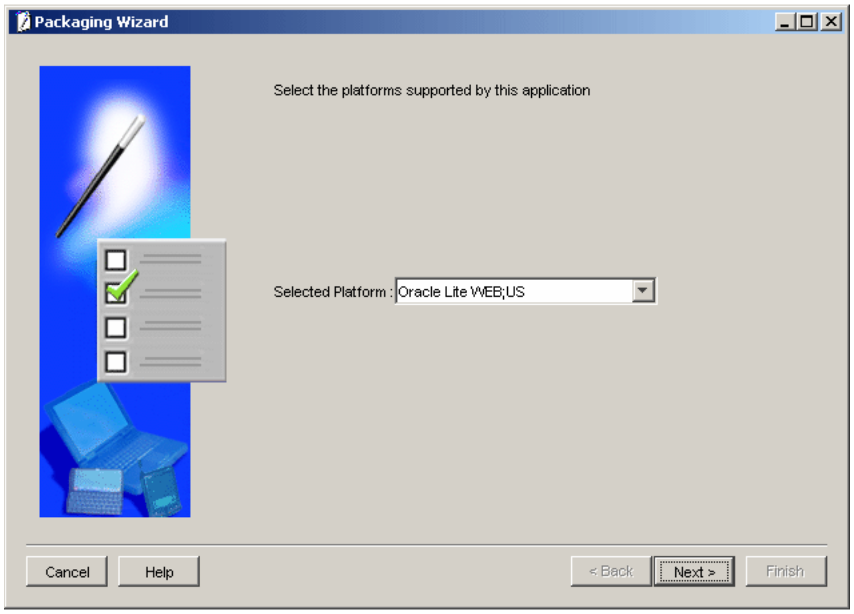
Delete an existing application

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 19–2 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 19–2 Selecting a Platform

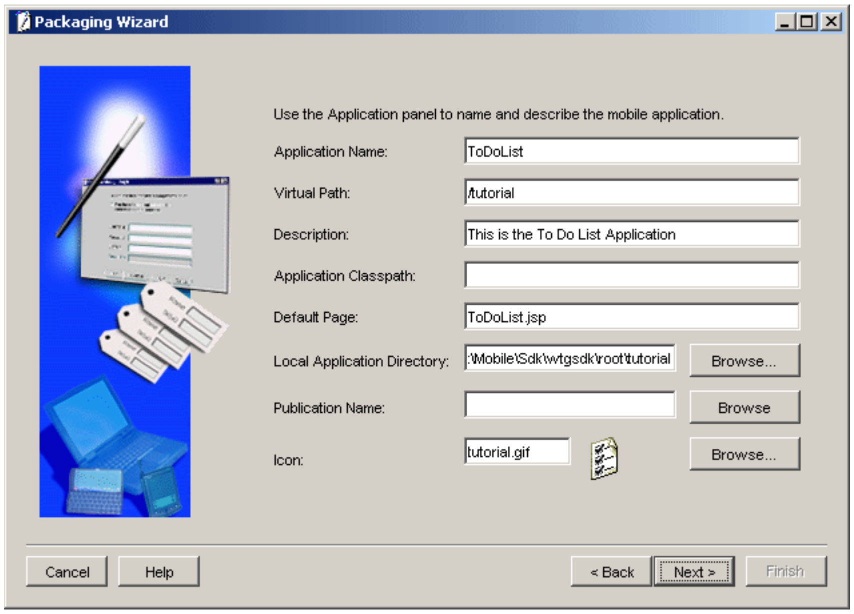

Packaging Wizard

Select the platforms supported by this application

Selected Platform : Oracle Lite WEB;US

4. As Figure 19–3 displays, the Application panel appears. Use the Application panel to modify To Do List application settings. As Table 19–4 describes, enter the specified values in the corresponding fields.

Figure 19–3 Application Panel



Use the Application panel to name and describe the mobile application.

Application Name:	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

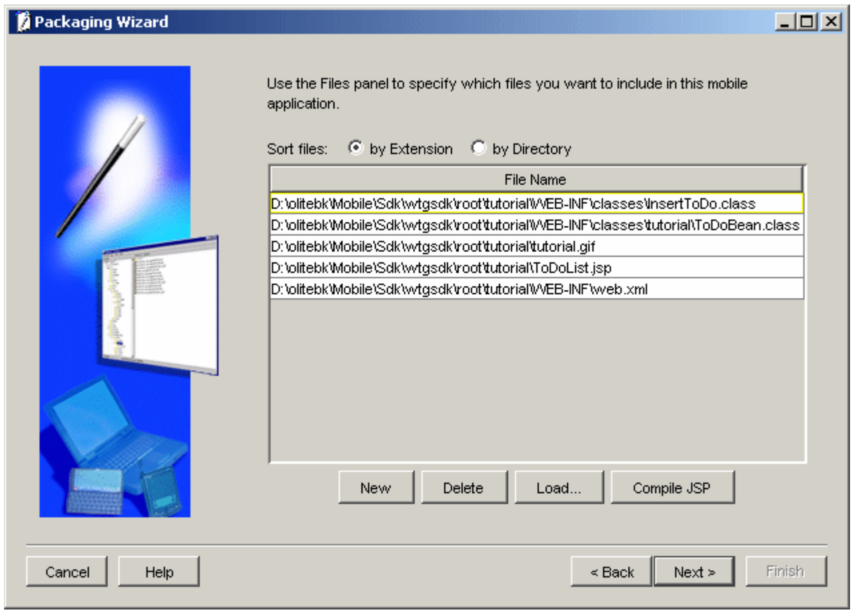
Table 19–4 The To Do List Application Values

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	(Leave this field blank)
Icon	`tutorial.gif`

5. Click **Next**. As Figure 19–4 displays, the Files panel appears. Using the Files panel, you can select files that are part of the application. The Packaging Wizard uploads the selected files from the local application directory to the application repository on the Mobile Server.

   The Files panel identifies files that the Packaging Wizard uploads from the local application directory to the application repository on the Mobile Server.

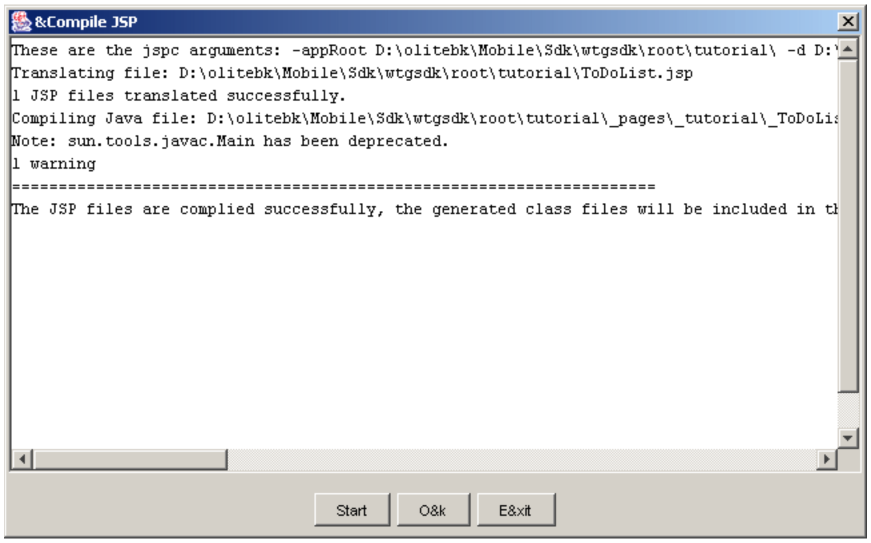
Figure 19–4 Uploading Application Files


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
New
Delete
Load...
Compile JSP
Cancel
Help
< Back
Next >
Finish

```
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 19–5 displays, the following confirmation page appears.

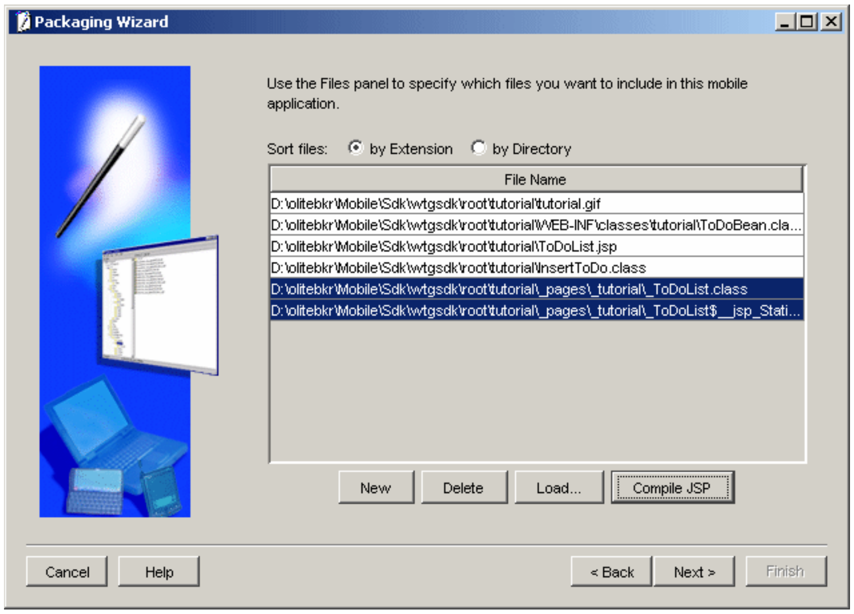
Figure 19–5 JSP Compilation Completion Message



```
These are the jspc arguments: -appRoot D:\olitebk\Mobile\Sdk\wtgsdk\root\tutorial\ -d D:\
Translating file: D:\olitebk\Mobile\Sdk\wtgsdk\root\tutorial\ToDoList.jsp
1 JSP files translated successfully.
Compiling Java file: D:\olitebk\Mobile\Sdk\wtgsdk\root\tutorial\_pages\_tutorial\_ToDoLi
Note: sun.tools.javac.Main has been deprecated.
1 warning
====================================================================
The JSP files are complied successfully, the generated class files will be included in t
```

7. As Figure 19–6 displays, the generated files are automatically added to the list of application files.

Figure 19–6 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
D:\olitebkr\Mobile\Sdk\wtgsdk\root\tutorial\tutorial.gif
D:\olitebkr\Mobile\Sdk\wtgsdk\root\tutorial\WEB-INF\classes\tutorial\ToDoBean.cla...
D:\olitebkr\Mobile\Sdk\wtgsdk\root\tutorial\ToDoList.jsp
D:\olitebkr\Mobile\Sdk\wtgsdk\root\tutorial\InsertToDo.class
D:\olitebkr\Mobile\Sdk\wtgsdk\root\tutorial_pages_tutorial_ToDoList.class
D:\olitebkr\Mobile\Sdk\wtgsdk\root\tutorial_pages_tutorial_ToDoList$__jsp_Stati...

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.

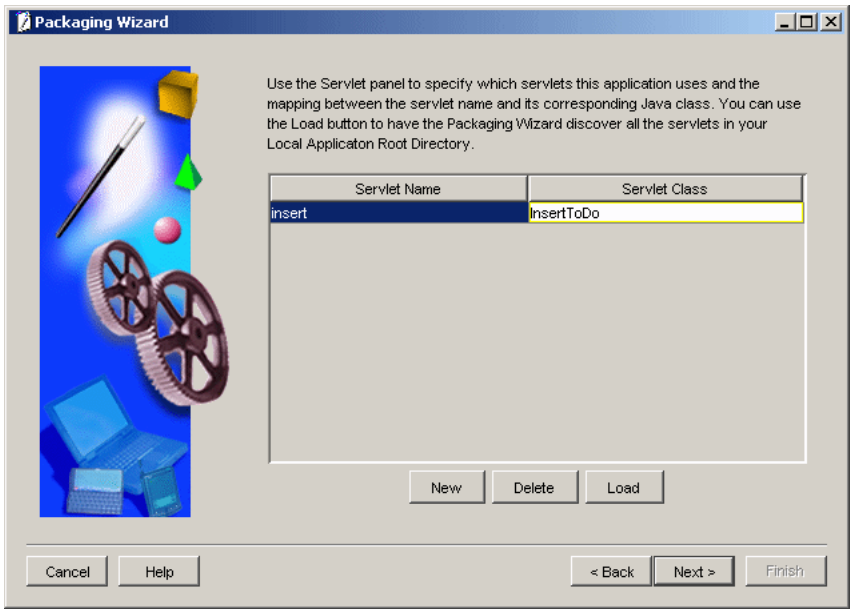
As Figure 19–7 displays, you can view the To Do List application's servlet in the Servlets panel. Since the To Do List application contains only one servlet, the Servlets panel displays a single line.

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.

Note: Ensure that you change the servlet name.

Figure 19–7 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. At this stage, this tutorial does not use the other tabs that are displayed in the Packaging Wizard. Click **Next** till you arrive at the last panel, and click **Finish**.

19.2.4 Step 4: Conducting a Trial Run

In this step, you will conduct a trial run of the To Do List application by starting the Mobile Client Web Server on the development computer. You will then access the To Do List application by launching your Web browser and connecting to the application's URL.

19.2.4.1 The Mobile Development Kit for Web-to-Go Web Server

The Mobile Client Web Server loads the To Do List application information and the Java servlet that you specified in the Packaging Wizard. Once started, you can access the Mobile Client Web Server from any Web browser, by specifying the URL of the computer it resides on. The default port for the Mobile Client Web Server is 7070. You can configure the port used by the Mobile Client Web Server by changing the port entry in the `webtogo.ora` file. This file is located in the following directory.

```
<ORACLE_HOME>\mobile\sdk\bin\webtogo.ora
```

For more information on how to edit the `webtogo.ora` file, see Section 11.3, "Editing the `webtogo.ora` file," in the *Oracle Database Lite Administration and Deployment Guide*.

For additional information regarding configuration parameters in the `webtogo.ora` file, see Appendix B, "Mobile Server Configuration Parameters," in the *Oracle Database Lite Administration and Deployment Guide*.

19.2.4.2 Required Action

Run the To Do List application by performing the following steps.

1. Start the Mobile Client Web Server.

   Using the Command Prompt, enter the following.

```
cd <ORACLE_HOME>\mobile\sdk\bin
```

```
wtgdebug.exe
```

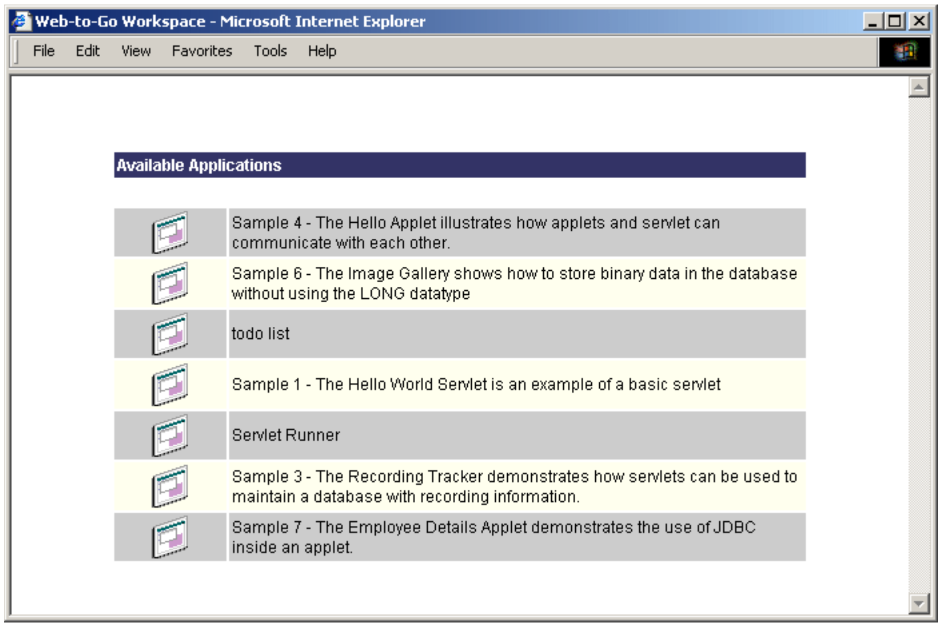
The Mobile Client Web Server starts and reports which servlets are loaded. If your servlets contain any `System.out.println()` statements, the messages appear in this window.

2. Start your Web browser and connect to the following URL.

```
http://<your_machine>:7070
```

As Figure 19–8 displays, the browser displays the list of applications currently known to the Web-to-Go system.

Figure 19–8 Available Applications Page


Web-to-Go Workspace - Microsoft Internet Explorer
File Edit View Favorites Tools Help

Available Applications

Sample 4 - The Hello Applet illustrates how applets and servlet can communicate with each other.

Sample 6 - The Image Gallery shows how to store binary data in the database without using the LONG datatype

todo list

Sample 1 - The Hello World Servlet is an example of a basic servlet

Servlet Runner

Sample 3 - The Recording Tracker demonstrates how servlets can be used to maintain a database with recording information.

Sample 7 - The Employee Details Applet demonstrates the use of JDBC inside an applet.

Table 19–5 describes the Available Applications page.

Table 19–5 List of Available Applications Description

Application	Description
Sample 4	The Hello Applet illustrates how applets and servlets can communicate with each other. The application is located in the following directory. `<ORACLE_HOME>\mobile\sdk\wtgsdk\root\sample4`
Sample 6	The Image Gallery shows how to store binary data in the database without using the `LONG` datatype. The application is located in the following directory. `<ORACLE_HOME>\mobile\sdk\wtgsdk\root\sample6`
To Do List	The application that you have added to the Mobile Client Web Server in Step 4.
Sample 1	The Hello World servlet is an example of a basic servlet. The application is located in the following directory. `<ORACLE_HOME>\mobile\sdk\wtgsdk\root\sample1`

Table 19–5 (Cont.) List of Available Applications Description

Application	Description
ServletRunner	The default application containing all published servlets that are not assigned to an application.
Sample 3	The Recording Tracker demonstrates how servlets can be used to maintain a database with recording information. The application is located in the following directory. `<ORACLE_HOME>\mobile\sdk\wtgsdk\root\sample3`
Sample 7	The Employee Data Applet demonstrates the use of JDBC inside an applet. The application is located in the following directory. `<ORACLE_HOME>\mobile\sdk\wtgsdk\root\sample7`

3. Click the **To Do List** application. A new browser window displays the following information.

   - The list of incomplete To Do items.
   - A simple HTML form that you can use to create new To Do items.

   All incomplete To Do items are preceded by the letter **'X'**. When you click **'X'**, the To Do List application flags the item as complete and removes the item from the list.

19.3 Packaging the Application

This section describes how to package the application and prepare it for publishing to the Mobile Server. In this section, you will perform the following tasks.

- Step 1: Defining the Application
- Step 2: Specifying Database Details
- Step 3: Defining the Snapshot
- Step 4: Defining Sequences
- Step 5: Creating SQL Files for the Application

19.3.1 Step 1: Defining the Application

In this step, you select and describe the To Do List application using the Packaging Wizard.

19.3.1.1 The Packaging Wizard

Using the Packaging Wizard, you can create or modify a Web-to-Go application and publish it to the Mobile Server. In this tutorial, you will use the Packaging Wizard to complete Steps 4 through 8 of the development phase.

19.3.1.2 Required Action

Select and describe the To Do List application by launching the Packaging Wizard in regular mode.

1. Using the Command Prompt, enter the following.

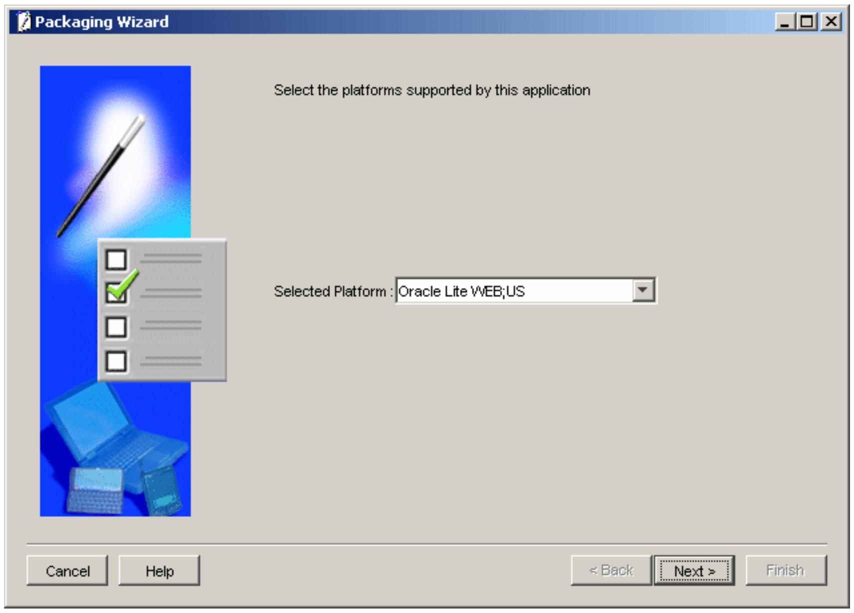
   a. `cd <ORACLE_HOME>\mobile\sdk\bin`

b. `wtgpack`

The Packaging Wizard appears.

2. Choose **Edit an existing application** and select the To Do List application from the list displayed.
3. Click **OK**. The Platforms panel appears. As Figure 19–9 displays, the Platforms panel contains the same information that you entered in Section 19.2.3, "Step 3: Defining the Application and Registering the Servlet".

Figure 19–9 Selecting a Platform

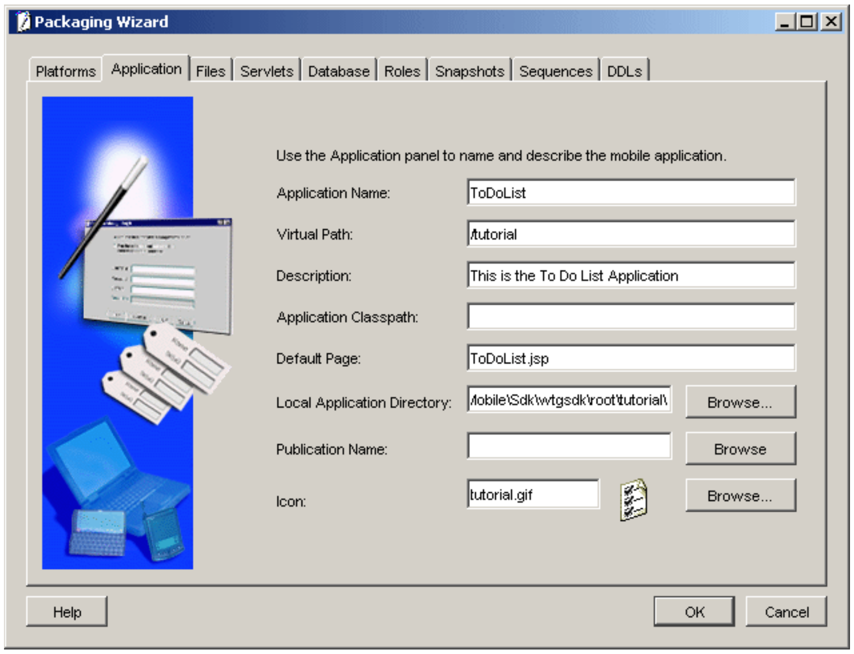


Select the platforms supported by this application

Selected Platform : Oracle Lite WEB;US

4. Click the **Application** tab. As Figure 19–10 displays, the Application tab contains the same information that you entered in Section 19.2.3, "Step 3: Defining the Application and Registering the Servlet".

Figure 19–10 Application Description 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: /lobile\Sdk\wtgsdk\root\tutorial\

Publication Name:

Icon: tutorial.gif

5. Describe the To Do List application by performing the following steps.

   a. As Table 19–6 describes, verify that the specified values in the following fields are correct.

Table 19–6 Application Panel Description

Field	Value
Application Name	`ToDoList`
Virtual Path	`/tutorial`
Description	This is the To Do List Application
Application Classpath	
Default Page	`ToDoList.jsp`
Local Application Directory	<*ORACLE_HOME*>`\mobile\sdk\wtgsdk\root\tutorial`
Publication Name	
Icon	`tutorial.gif`

   b. Click the **Files** tab. The Packaging Wizard automatically includes all files to the application.

   c. Click the **Servlets** tab. The Servlets tab appears.

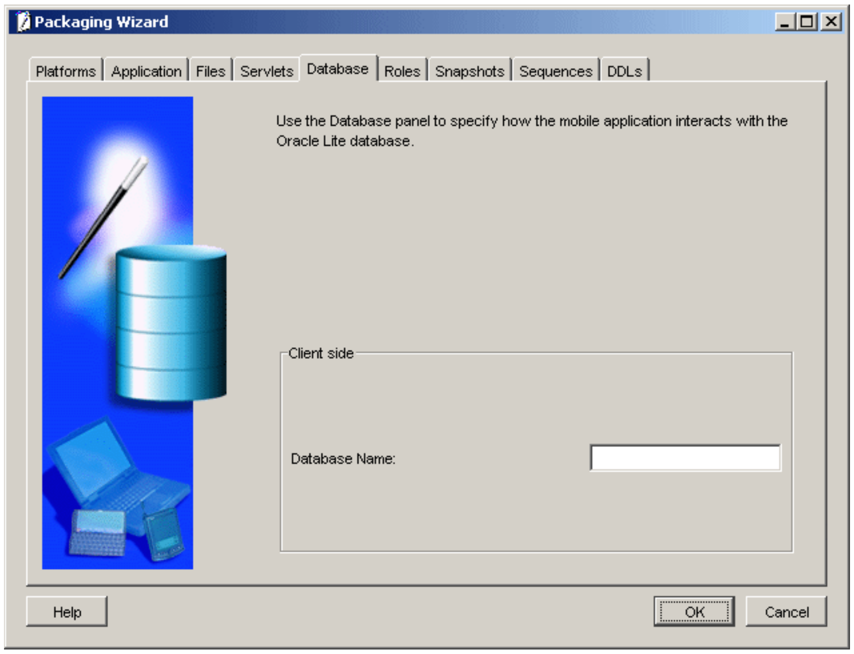
   d. Click the **Database** tab. The Database tab appears.

19.3.2 Step 2: Specifying Database Details

In this step, you will specify the Client Side Database Name.

Figure 19–11 displays the Database tab.

Figure 19–11 Database Tab



Use the Database panel to specify how the mobile application interacts with the Oracle Lite database.

Client side

Database Name:

The Database Name refers to the database file and the corresponding DSN that will be created for this application on the Mobile Client for Web-to-Go.

19.3.2.1 Required Action

Enter **todo** as the Client Side Database Name.

Click the **Snapshots** tab.

Note: This tutorial skips Roles because the tutorial application does not use any special roles.

The Roles tab enables the developer to define roles for the Web-to-Go application. In general, the developer must build application roles into the Web-to-Go application, because they do not occur automatically. For more information on how to build application roles, see Chapter 7, "Developing Mobile Web Applications", Section 7.2.2, "Application Roles".

19.3.3 Step 3: Defining the Snapshot

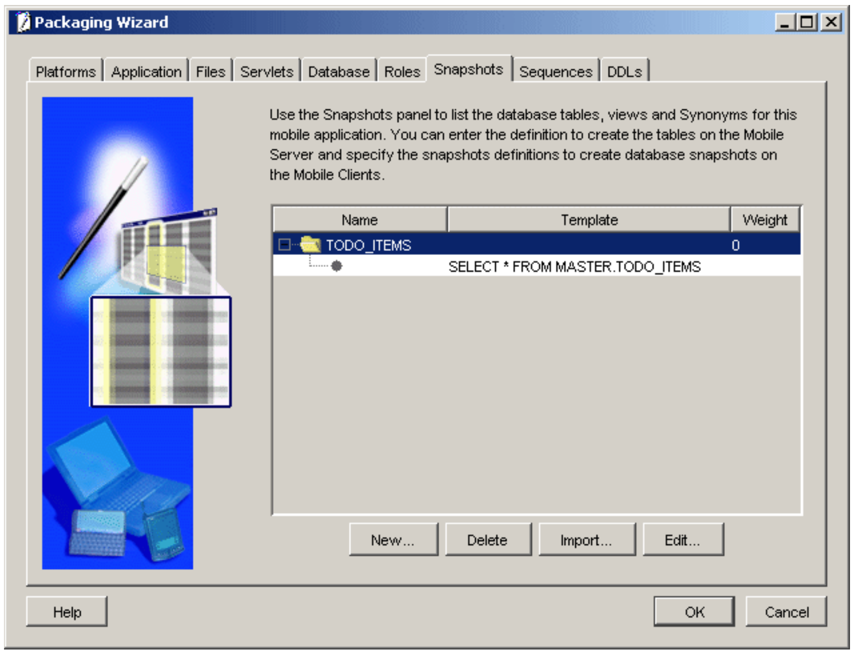
In this step, you will deploy the To Do List application's database schema objects using the Packaging Wizard.

19.3.3.1 The Snapshots Tab

The Snapshots tab defines database tables for which you will create snapshots. Using the Packaging Wizard, you can import the table definitions from the development database. These definitions can then be used to define the snapshots for the Mobile application.

Figure 19–12 displays the Snapshots tab.

Figure 19–12 Snapshots Tab


Use the Snapshots panel to list the database tables, views and Synonyms 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.

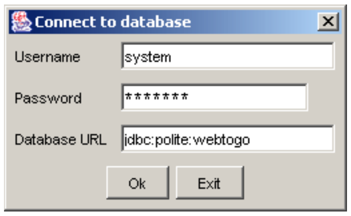
Name	Template	Weight
TODO_ITEMS		0
	SELECT * FROM MASTER.TODO_ITEMS	

19.3.3.2 Required Action

In the Snapshots tab, import the table definition from the development database by performing the following steps.

1. Click **Import**. As Figure 19–13 displays, the Connect to Database dialog appears. As Table 19–7 describes, enter the following information in the corresponding fields.

Figure 19–13 Connect to Database Dialog


Connect to database

Username	system
Password	*******
Database URL	jdbc:polite:webtogo

Table 19–7 Connect to Database Dialog Description

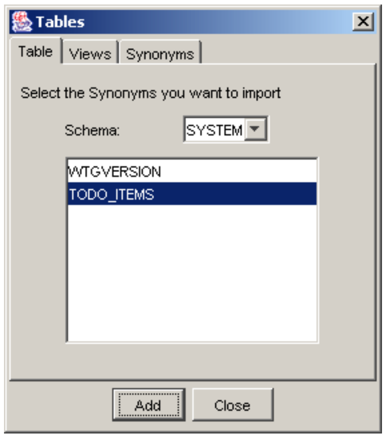
Field	Value
User Name	system
Password	Enter your database password
URL	`jdbc:polite:webtogo`

Note: Importing a table definition within the Packaging Wizard caches the JDBC Connection information. You cannot re-import a table definition using a different `Connect String` as the same connection information is used and cannot be modified.

---

2. Click **OK**. As Figure 19–14 displays, the Tables dialog appears and displays a list of available tables.

Figure 19–14 Tables Dialog



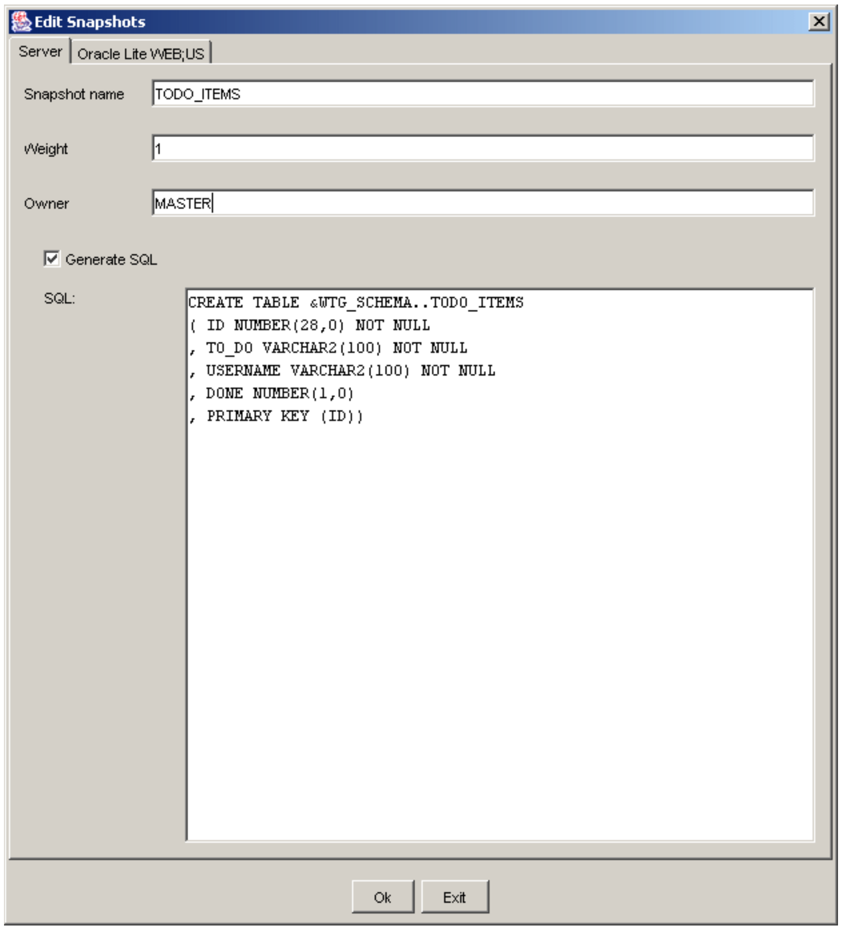
Select the Synonyms you want to import

Schema: SYSTEM

WTGVERSION
TODO_ITEMS

3. Select the `TODO_ITEMS` table, click **Add**, and click **Close**. The `TODO_ITEMS` snapshot appears in the Tables dialog. To view the SQL statement for the snapshots template, double-click `TODO_ITEMS`.
4. Select the SQL statement and click **Edit**. As Figure 19–15 displays, the Edit Snapshots panel appears.

Figure 19–15 Edit Snapshots Panel - Server Tab



Snapshot name: TODO_ITEMS

Weight: 1

Owner: MASTER

[✓] Generate SQL

SQL:

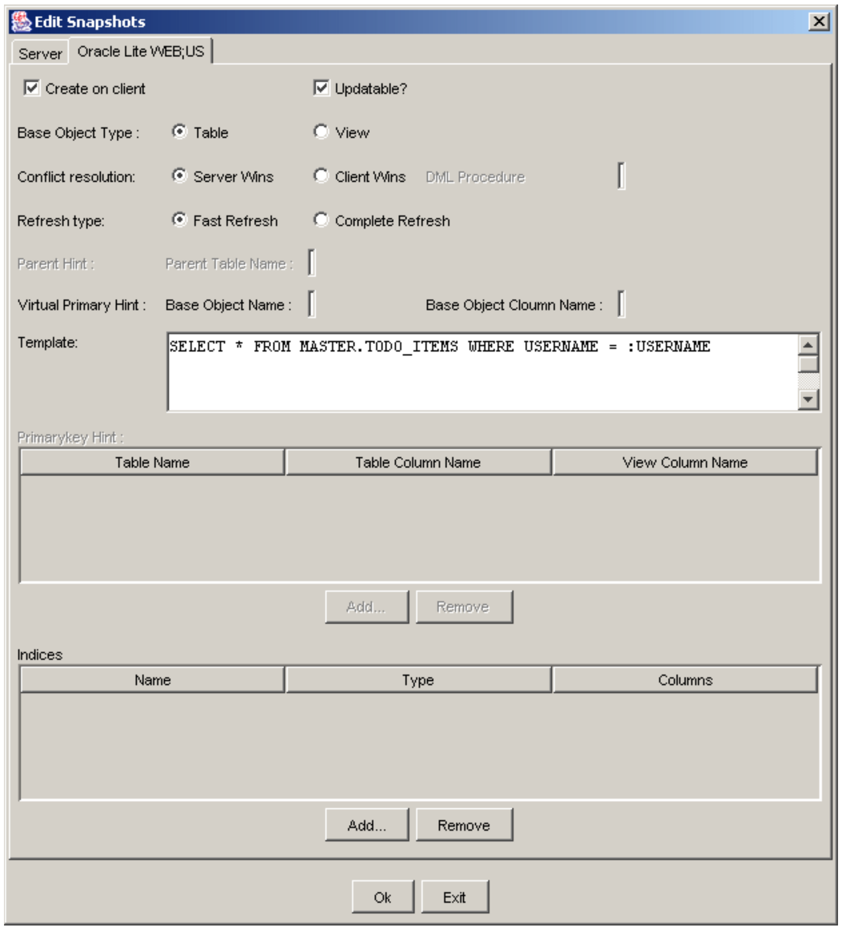
```
CREATE TABLE &WTG_SCHEMA..TODO_ITEMS
( ID NUMBER(28,0) NOT NULL
, TO_DO VARCHAR2(100) NOT NULL
, USERNAME VARCHAR2(100) NOT NULL
, DONE NUMBER(1,0)
, PRIMARY KEY (ID))
```

5. Change the weight to 1. This parameter controls the order in which snapshots are refreshed on the client.
6. Change the Owner to **master**. The Packaging Wizard seeks your confirmation to change the owner for all the templates. Click **OK**.
7. Click the Oracle Lite WEB;US tab. Select the **Create on client** box. Re-enter the SQL statement in the Template field as given below, and click **OK**.

   ```
   SELECT * FROM MASTER.TODO_ITEMS WHERE USERNAME = :USERNAME
   ```

   Figure 19–16 displays the SQL statement in the Template field.

Figure 19–16 Edit Snapshots Panel



[✓] 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

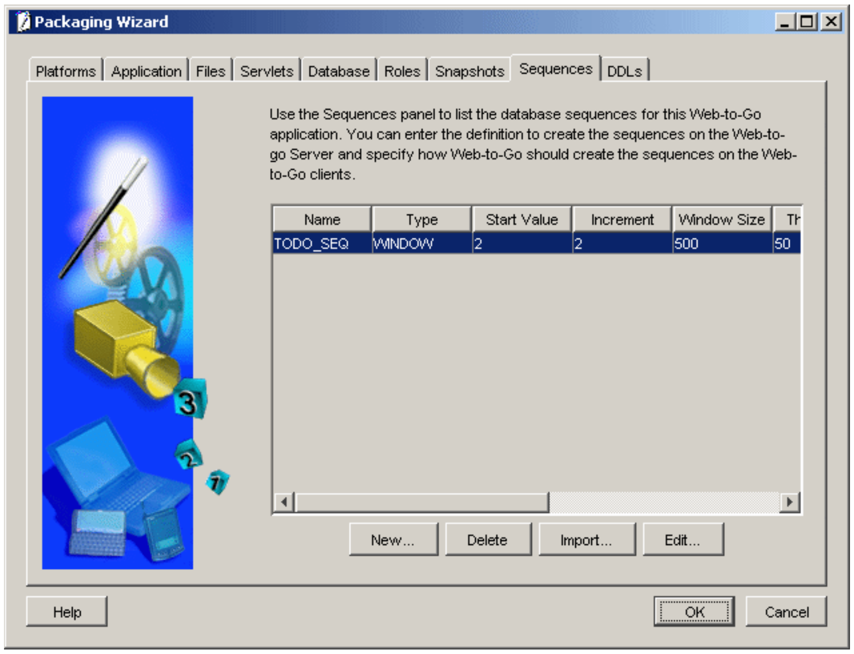
Name	Type	Columns

19.3.4 Step 4: Defining Sequences

The Sequences panel defines sequences that Web-to-Go creates for your client's applications in offline mode. In this step, you create a new definition of the TODO_SEQ sequence which the To Do List application uses in offline mode. Later on, you will create the actual sequences in the Oracle database. During synchronization, Web-to-Go automatically creates a local copy of the TODO_SEQ sequence on your client.

1. Click the Sequences tab. The Sequences panel appears as displayed in Figure 19–17. Using the Sequences tab, you can list database sequences for Web-to-Go applications. To specify how Web-to-Go creates sequences on the Mobile Client for Web-to-Go, you can include sequence definitions. These sequences must exist in the database prior to performing this step.

Figure 19–17 Sequences Tab

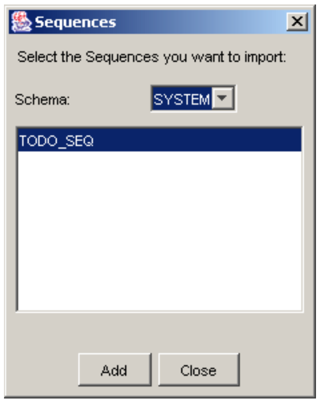


Use the Sequences panel to list the database sequences for this Web-to-Go application. You can enter the definition to create the sequences on the Web-to-go Server and specify how Web-to-Go should create the sequences on the Web-to-Go clients.

Name	Type	Start Value	Increment	Window Size	Th
TODO_SEQ	WINDOW	2	2	500	50

2. Click **Import**. As Figure 19–18 displays, the Sequences dialog appears displaying a list of available sequences.

Figure 19–18 Sequences Dialog



Select the Sequences you want to import:

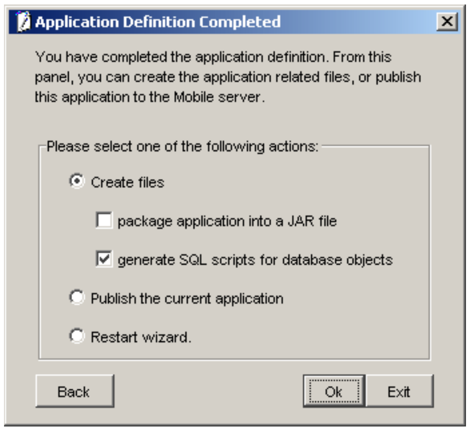
Schema: SYSTEM

TODO_SEQ

3. Select the **TODO_SEQ** sequence. Click **Add** and click **Close**.

4. Click **OK**. The Application Definition Completed panel appears, as displayed in Figure 19–19.

Figure 19–19 Application Definition Completed Dialog


Application Definition Completed

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.

Note: This tutorial application does not use DDLs and therefore skips the DDLs tab.

19.3.5 Step 5: Creating SQL Files for the Application

Using the Application Definition Completed panel, you can create SQL files for the To Do List application.

19.3.5.1 Required Action

Select the **Create files** option and select the **Generate SQL scripts for database objects** box. Click **OK**.

This action generates SQL scripts for database objects.

The Packaging Wizard places the specified files in the following directory.

`<ORACLE_HOME>\mobile\sdk\wtgsdk\root\tutorial\sql`

Table 19–8 describes the SQL scripts.

Table 19–8 SQL Scripts for Database Objects

File	Description
`ToDoList.sql`	The master script that calls other SQL scripts.
`tables.sql`	The script that creates all SQL tables.
`sequences.sql`	The script that creates the Sequences.
`ddls.sql`	This file is empty because no DLLs are defined.

19.3.6 Step 6: Package the Application

Using the Application Definition Completed panel, you can package the To Do List application into a JAR file.

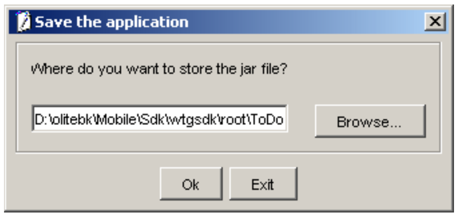
19.3.6.1 Required Action

The Application Definition Completed Dialog remains open for you to initiate application packaging.

1. Select the **Create files** option and select the **Package Application into a JAR file** box. Ensure that you select the **Generate SQL scripts for database objects** box.
2. At this stage, the Save the Application dialog prompts you for the name of the JAR file, as Figure 19–20 displays. The default location is given below.

   ```
   <ORACLE_HOME>\Mobile\Sdk\wtgsdk\root\ToDoList.jar
   ```

Figure 19–20 Save the Application Dialog


Save the application

Where do you want to store the jar file?

D:\olitebk\Mobile\Sdk\wtgsdk\root\ToDo

After choosing the JAR file, the jar file is created and contains the application files and definition.

You have now completed all development tasks that are required for packaging your application. Your application is packaged.

19.4 Publishing the Application

After packaging your application, you are ready to publish it. The following sections describe the steps for publishing the application.

19.4.1 Step1: Create the Table Owner Account

In this step, you will 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.

```
sqlplus system/manager@<MOBILESERVER_JDBC_URL>

create user master identified by master;

grant connect, resource to master;
```

19.4.2 Step 2: Create the Database Objects in the Oracle Database

In this step, you create database objects of the To Do List application in the Oracle database.

19.4.2.1 Required Action

Run the SQL master script and enter the following using the Command Prompt.

```
cd <ORACLE_HOME>\mobile\sdk\wtgsdk\root\tutorial\sql

sqlplus master/master@<MOBILESERVER_JDBC_URL> @ToDoList.sql
```

This script performs the following actions on the Oracle database.

- Creates the `TODO_ITEMS` table.
- Creates the `TODO_SEQUENCE` sequence.

19.4.3 Step 3: Start the Mobile Server

In this step, you start the Mobile Server.

19.4.3.1 Required Action

To start the Mobile Server, 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, enter the following command.

   ```
   runmobileserver
   ```

19.4.4 Step 4: Log on to the Mobile Server and Start the Mobile Manager

In this step, you will log on to the Mobile Server and start the Mobile Manager.

19.4.4.1 Required Action

To start the Mobile Manager, perform the following steps.

1. Start your Web browser and connect to the Mobile Server by enter the following URL.

   ```
   http://<mobile_server>/webtogo
   ```

   Note: Replace the `<mobile_server>` variable with the host name of your Mobile Server.

2. Log on 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.
4. Click your Mobile Server link.

19.4.5 Step 5: Upload the Application

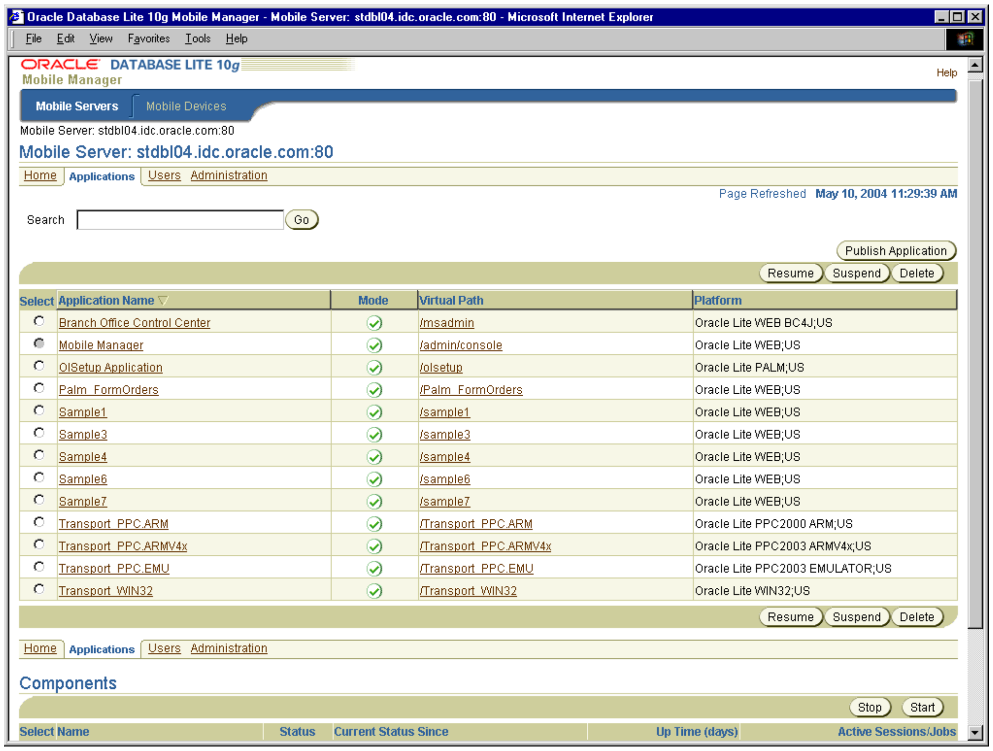
In this step, you upload the JAR file containing the To Do List application.

19.4.5.1 Required Action

To upload an application to the Mobile Server, perform the following steps.

1. Click the **Applications** link. As Figure 19–21 displays, the Applications page appears.

Figure 19–21 Applications Page



Mobile Server: stdbl04.idc.oracle.com:80

Mobile Server: stdbl04.idc.oracle.com:80

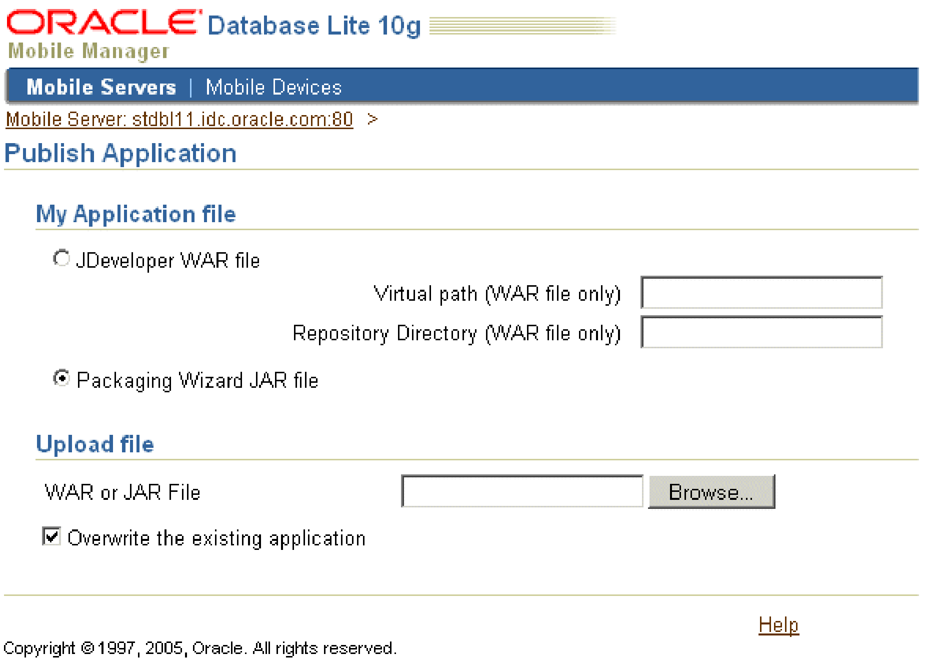
Page Refreshed May 10, 2004 11:29:39 AM

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

Select Name	Status	Current Status Since	Up Time (days)	Active Sessions/Jobs

2. Click **Publish Application**. As Figure 19–22 displays, the Publish Application page appears.

Figure 19–22 Publish Application Page



Mobile Server: stdbl11.idc.oracle.com:80 >

Publish Application

My Application file

○ JDeveloper WAR file

Virtual path (WAR file only)

Repository Directory (WAR file only)

● Packaging Wizard JAR file

Upload file

WAR or JAR File

☑ Overwrite the existing application

Copyright © 1997, 2005, Oracle. All rights reserved.

3. Select the Packaging Wizard JAR File option.

4. Using the **Browse** button, locate the jar file which you created in Section 19.3.6, "Step 6: Package the Application". The default location of the jar file is given below.

   ```
   <ORACLE_HOME>\Mobile\sdk\wtgsdk\root\ToDoList.jar
   ```

5. To publish the application, click **Publish**.

> **Note:** You will set the application properties in the following Section 19.5.3, "Step 3: Setting Application Properties".

19.5 Administering the Application

This section describes how to administer the application that you created and deployed. In this section, you will perform the following tasks.

- Section 19.5.1, "Step 1: Starting the Mobile Manager"
- Section 19.5.2, "Step 2: Using the Mobile Manager to Create a New User"
- Section 19.5.3, "Step 3: Setting Application Properties"
- Section 19.5.4, "Step 4: Granting User Access to the Application"
- Section 19.5.5, "Step 5: Defining Snapshot Template Values for the User"

For more information about Mobile Manager tasks described in this tutorial, see the *Oracle Database Lite Administration and Deployment Guide*.

19.5.1 Step 1: Starting the Mobile Manager

The Mobile Manager is a Web-based application that enables you to administer Mobile Server applications.

19.5.1.1 Required Action

To start the Mobile Manager, perform the following steps.

1. Start your Web browser and connect to the Mobile Server by entering the following URL.

   ```
   http://<mobile_server>/webtogo
   ```

> **Note:** Replace the `<mobile_server>` variable with the host name of your Mobile Server.

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. Click your Mobile Server link. Your Mobile Server home page appears.

19.5.2 Step 2: Using the Mobile Manager to Create a New User

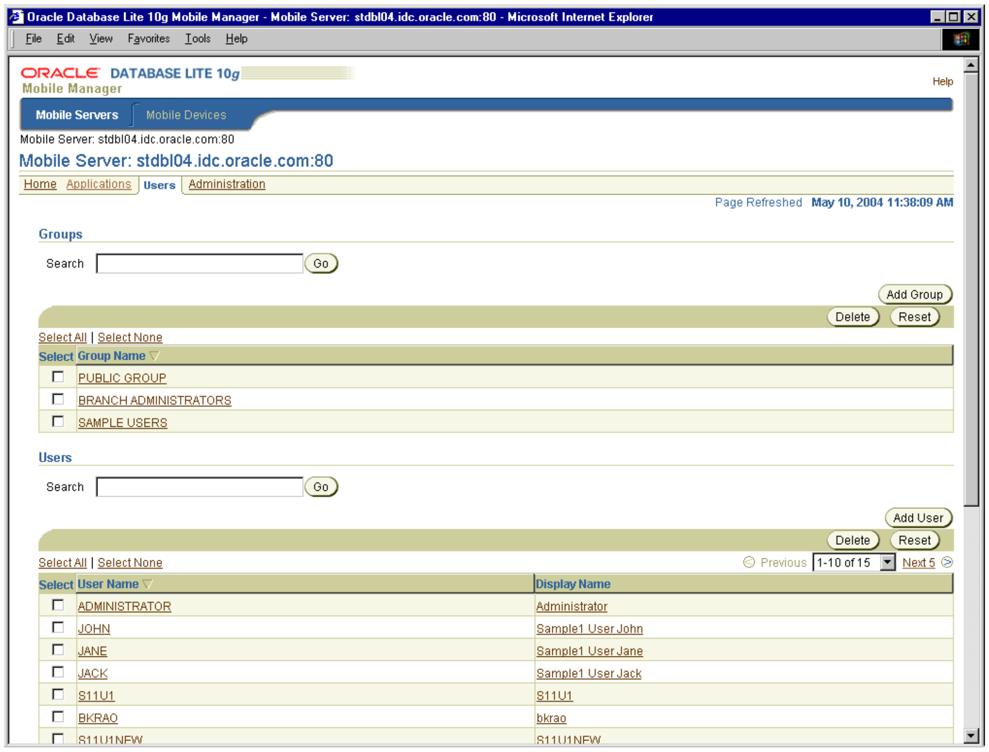
In this step, you will create a new user.

19.5.2.1 Required Action

To create a new Mobile Server user, perform the following steps.

1. On the Mobile Manager home page, click the **Users** link. As Figure 19–23 displays, the Users page appears.

Figure 19–23 Users Page



Oracle Database Lite 10g Mobile Manager - Mobile Server: stdbl04.idc.oracle.com:80 - Microsoft Internet Explorer

Mobile Server: stdbl04.idc.oracle.com:80

Mobile Server: stdbl04.idc.oracle.com:80

Page Refreshed May 10, 2004 11:38:09 AM

Groups

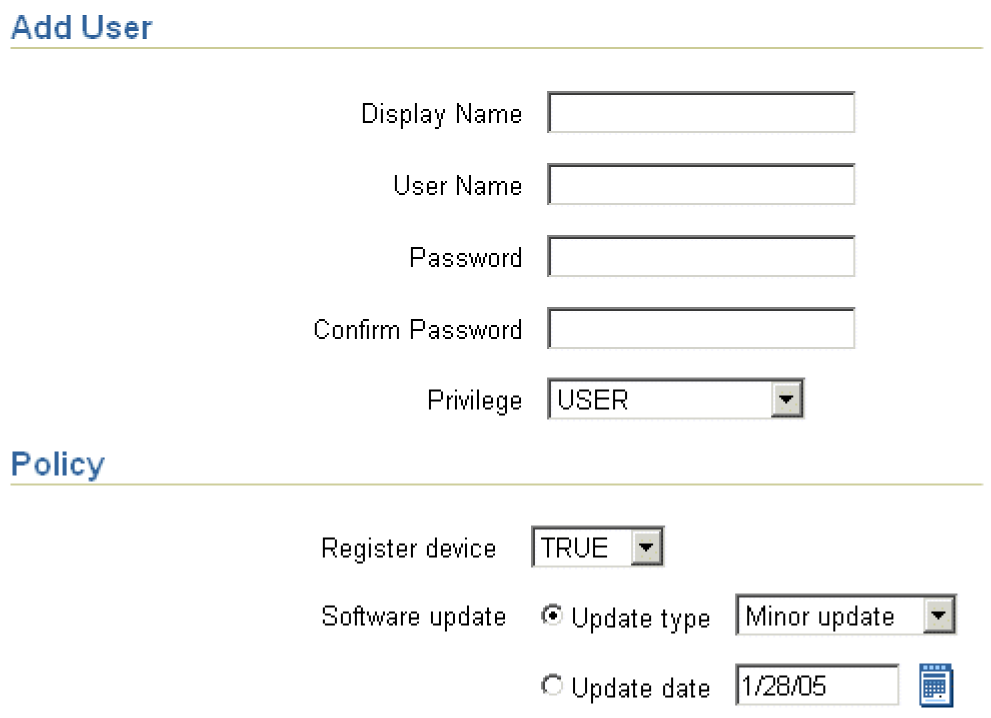
Select	Group Name
☐	PUBLIC GROUP
☐	BRANCH ADMINISTRATORS
☐	SAMPLE USERS

Users

Select	User Name	Display Name
☐	ADMINISTRATOR	Administrator
☐	JOHN	Sample1 User John
☐	JANE	Sample1 User Jane
☐	JACK	Sample1 User Jack
☐	S11U1	S11U1
☐	BKRAO	bkrao
☐	S11U1NEW	S11U1NEW

2. Click **Add User**. As Figure 19–24 displays, the Add User page appears.

Figure 19–24 Add User Page



Add User

Display Name	
User Name	
Password	
Confirm Password	
Privilege	USER

Policy

Register device	TRUE	
Software update	● Update type	Minor update
	○ Update date	1/28/05

3. As described in Table 19–9, enter the following information in the Add User page and click **Save**.

Table 19–9 Add User Page Description

Field	Value
Display Name	`tutorial`
User Name	`tutorial`
Password	`tutorial`
Password Confirm	`tutorial`
Privilege	User

19.5.3 Step 3: Setting Application Properties

In this step, you will set the To Do List application's properties.

19.5.3.1 Required Action

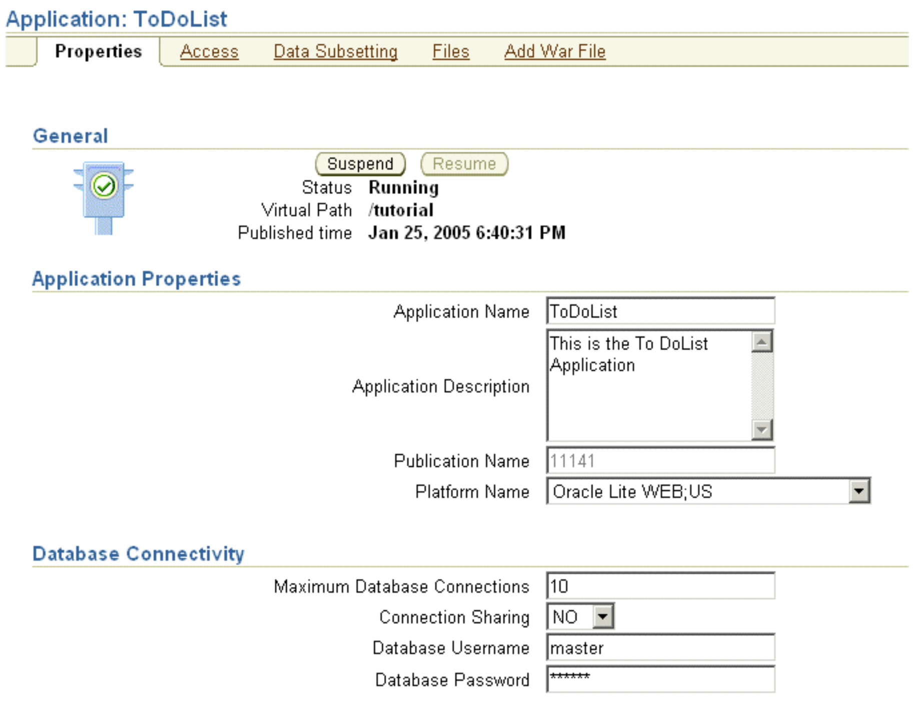
To set the To Do List application's properties, perform the following steps.

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 19–25 displays, the Application Properties page lists application properties and database connectivity details.

Figure 19–25 Application Properties Page



Status	Running
Virtual Path	/tutorial
Published time	Jan 25, 2005 6:40:31 PM
Application Name	ToDoList
Application Description	This is the To DoList Application
Publication Name	11141
Platform Name	Oracle Lite WEB;US
Maximum Database Connections	10
Connection Sharing	NO
Database Username	master
Database Password	******

4. In the **Database Password** field type `master`. This is the default password for the Web-to-Go demo schema. Click **Apply**. The Mobile Manager displays a confirmation message.

19.5.4 Step 4: Granting User Access to the Application

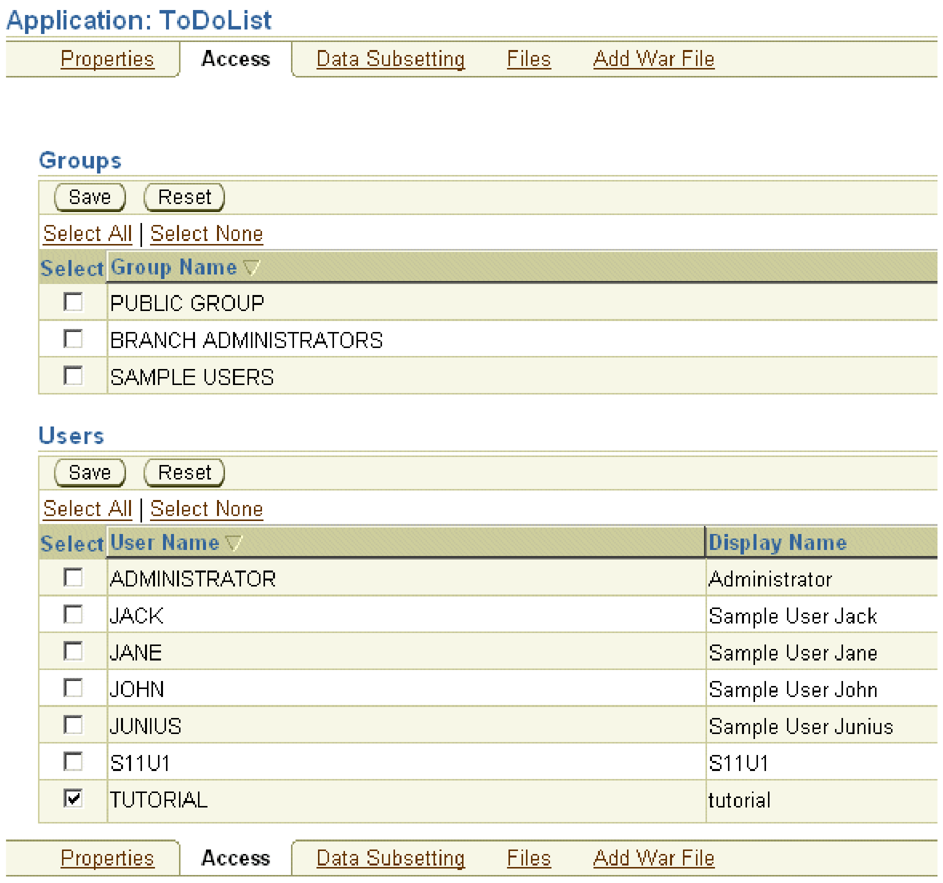
In this step, you grant the user `TUTORIAL` access to the To Do List application.

19.5.4.1 Required Action

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 19–26 displays, the Access page lists groups and users that are associated with the application. The check boxes on this page indicate whether or not the user or group has access to the application.

Figure 19–26 Access Page



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.

19.5.5 Step 5: Defining Snapshot Template Values for the User

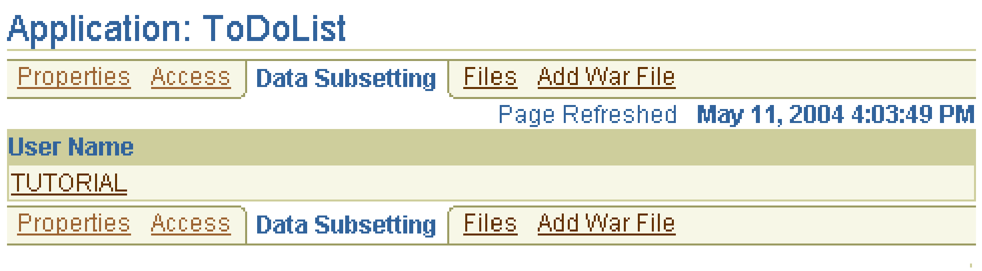
In this step, you will define the snapshot template variable for the user, `TUTORIAL`. Each Mobile Client for Web-to-Go downloads the same application data when it synchronizes. In some cases, you may want to specify the data your application downloads for each user. You can accomplish this by modifying the user's snapshot template variable.

19.5.5.1 Required Action

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 19–27 displays, the Data Subsetting page appears.

Figure 19–27 Data Subsetting Page


Application: ToDoList

Properties Access **Data Subsetting** Files Add War File

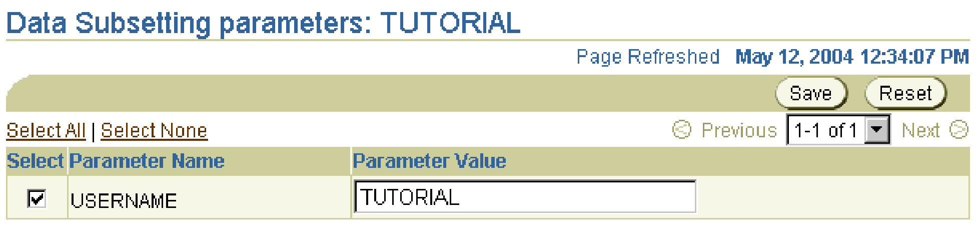
Page Refreshed **May 11, 2004 4:03:49 PM**

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 19–28 displays, the Data Subsetting Parameters page appears.

Figure 19–28 Data Subsetting Parameters Page


Data Subsetting parameters: TUTORIAL

Page Refreshed **May 12, 2004 12:34:07 PM**

Select	Parameter Name	Parameter Value
✓	USERNAME	TUTORIAL

3. Select the Parameter Name and enter the value `tutorial`. Click **Save**.

For more information about Snapshots, refer the *Oracle Database Lite Administration and Deployment Guide*.

19.6 Running the Application on the Mobile Client for Web-to-Go

This section describes how to use the application that you created and tested in the Development section, deployed in the Deployment section, and then administered in the Administration section. In this section, you will perform the following tasks:

- Section 19.6.1, "Step 1: Installing the Mobile Client for Web-to-Go"
- Section 19.6.2, "Step 2: Logging into the Mobile Client for Web-to-Go"
- Section 19.6.3, "Step 3: Synchronizing the Mobile Client for Web-to-Go"

Note: You must install the application and test it on a separate machine from the Mobile Server.

19.6.1 Step 1: Installing the Mobile Client for Web-to-Go

This section describes how to 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.

19.6.1.1 Required Action

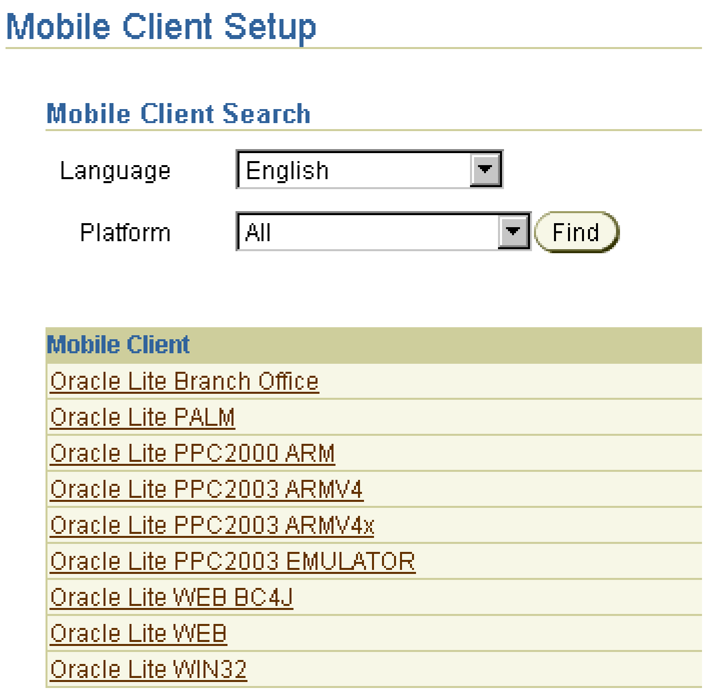
To install the Mobile Client for Web-to-Go, perform the following actions.

1. Start your Web browser and connect to the Mobile Server by entering the following URL.

   ```
   http://<mobile_server>/webtogo/setup
   ```

2. As Figure 19–29 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 19–29 Mobile Client Setup Page



Mobile Client Setup

Mobile Client Search

Language	English
Platform	All

Mobile Client
Oracle Lite Branch Office
Oracle Lite PALM
Oracle Lite PPC2000 ARM
Oracle Lite PPC2003 ARMV4
Oracle Lite PPC2003 ARMV4x
Oracle Lite PPC2003 EMULATOR
Oracle Lite WEB BC4J
Oracle Lite WEB
Oracle Lite WIN32

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.

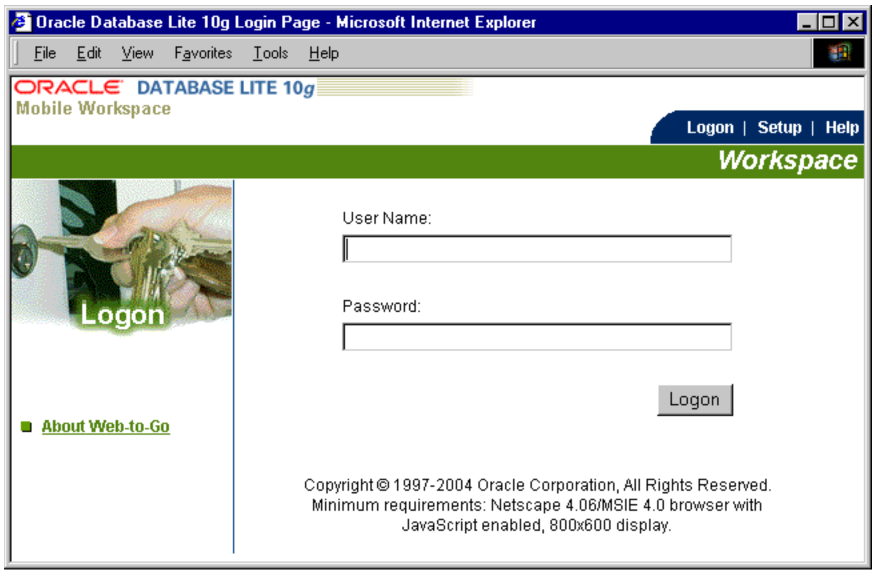
3. If you are using Netscape, choose a location to save the setup program and click **OK**. In Windows Explorer, double-click `setup.exe` to run the setup program.

   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 19–30 displays.

Figure 19–30 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.

19.6.2 Step 2: Logging into the Mobile Client for Web-to-Go

In this step, you will complete the Mobile Client for Web-to-Go setup process.

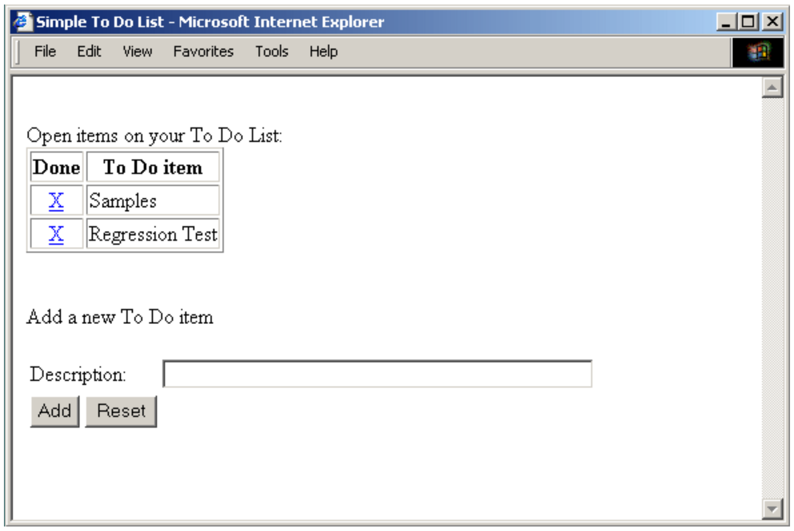
19.6.2.1 Required Action

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 19–31 displays, Web-to-Go launches the To Do List application in your browser.

Figure 19–31 The To Do List Application


Simple To Do List - Microsoft Internet Explorer
File Edit View Favorites Tools Help

Open items on your To Do List:

Done	To Do item
X	Samples
X	Regression Test

Add a new To Do item

Description:

7. Enter a new To Do item and save it in the database. Click **Add**.
8. Exit the application by closing the browser window. This action returns you to the workspace.

19.6.3 Step 3: Synchronizing the Mobile Client for Web-to-Go

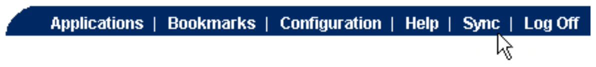
In this step, you synchronize the Mobile Client for Web-to-Go.

19.6.3.1 Required Action

To synchronize the Mobile Client for Web-to-Go with the Mobile Server, perform the following steps.

1. As Figure 19–32 displays, click the Sync tab in the upper right corner of the workspace.

Figure 19–32 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.

20 Building Offline Mobile Web Applications Using BC4J: A Tutorial

This document enables you to create, deploy, and use a BC4J application, using a tutorial. Topics include:

20.1 Overview

The Oracle BC4J (Business Components for Java) is a part of the Oracle JDeveloper IDE (Integrated Development Environment), and provides Java developers with tools to create and manage reusable Java components.

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 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, BC4J automatically generates database oriented components, so that you can focus on the business logic instead of on database related components.

The sample BC4J application used in this tutorial maintains employee details and stores all items in a relational database.

20.1.1 Before You Start

Before you start developing business components in Java, ensure that the computer you are using for your development meets the requirements specified in this section.

Table 20–1 lists configuration and installation requirements for the development computer.

Table 20–1 Development Computer Requirements

Requirement	Description
Windows NT/2000/XP User Login	The Windows NT/2000/XP login user must have Administrator privileges on the development computer.
Installed Java Components	Java Development Kit 1.4.2 or higher.
Installed Oracle Components	Mobile Server or Mobile Development Kit (Oracle Database Lite CD-ROM)
	Oracle 8.1.7 or higher
	Oracle9*i* JDeveloper, Release 9.0.3.

Note: The BC4J tutorial is shipped with the Mobile Development Kit as a JAR file named `OracleLite_BC4J_Tutorial.jar`. The file is located in the directory <*ORACLE_HOME*>`\mobile\sdk\wtgsdk\src`. You can use this JAR file to publish the BC4J tutorial to the Mobile Server and then continue with the rest of the tutorial by following the steps given in Section 20.7, "Deploying the Sample Application". If you want to develop the same application (as packaged in `OracleLite_BC4J_Tutorial.jar)`, follow the steps from Section 20.2, "Developing the Application" on.

20.2 Developing the Application

This section enables you to develop the BC4J application for Oracle Database Lite in phases.

To develop the BC4J application, you must perform the following tasks.

1. Section 20.2.1, "Creating the Database Connection"
2. Section 20.2.2, "Creating the BC4J Component"
3. Section 20.2.3, "Configuring the BC4J Component to Use the WTGJdbc Connection"
4. Section 20.2.4, "Building and Deploying the BC4J Component as a Simple Archive"
5. Section 20.2.5, "Writing the JSP Application to Access the BC4J Component"
6. Section 20.2.6, "Deploying the JSP Application as a Simple Archive"

20.2.1 Creating the Database Connection

For developing an application using BC4J, you must create two database connections, as follows:

- A connection to the back-end Oracle database that is used only within BC4J to test the application while developing within BC4J.
- The permanent connection used between the back-end Oracle database and the Oracle Database Lite, which is used by the application after deployment for synchronization between the two databases.

During development and testing within BC4J, use the first connection. After you are done developing the application and are ready to deploy it, change the connection from to the second connection before deploying the application.

The following sections provide details on how to create each type of connection:

- Section 20.2.1.1, "Create the Connection to the Back-End Oracle Database"
- Section 20.2.1.2, "Create the Connection to the Oracle Lite Database"

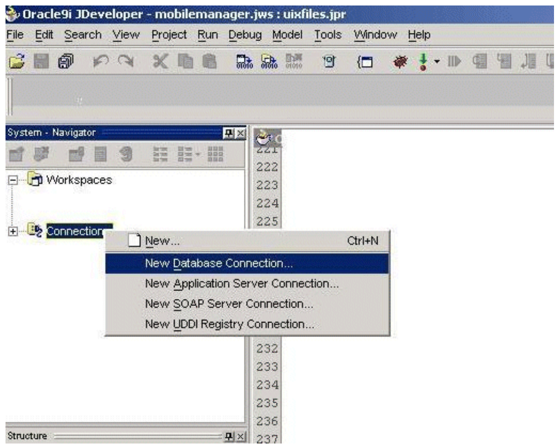
20.2.1.1 Create the Connection to the Back-End Oracle Database

You need to create a connection to the back-end Oracle database, which is used only within BC4J to test the application while developing within BC4J. For this example, we create the `tutorialConn` connection to connect to the back-end Oracle database using the `oracle.jdbc.driver.OracleDriver` for the BC4J development process.

To create the `tutorialConn` connection, perform the following steps.

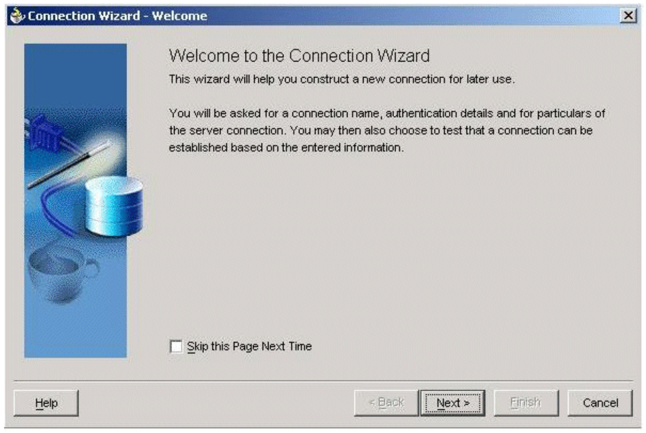
1. In the JDeveloper **System Navigator** panel and as displayed in Figure 20–1, right-click the **Connections** node and choose the **New Database Connection** option.

Figure 20–1 Choosing a New Database Connection


Oracle9i JDeveloper - mobilemanager.jws : uixfiles.jpr
File Edit Search View Project Run Debug Model Tools Window Help
System - Navigator
Workspaces
Connections
New... Ctrl+N
New Database Connection...
New Application Server Connection...
New SOAP Server Connection...
New UDDI Registry Connection...
Structure

2. The Connection Wizard Welcome panel appears, which is displayed in Figure 20–2. Click **Next**.

Figure 20–2 Welcome Panel - Connection Wizard



Connection Wizard - Welcome

Welcome to the Connection Wizard

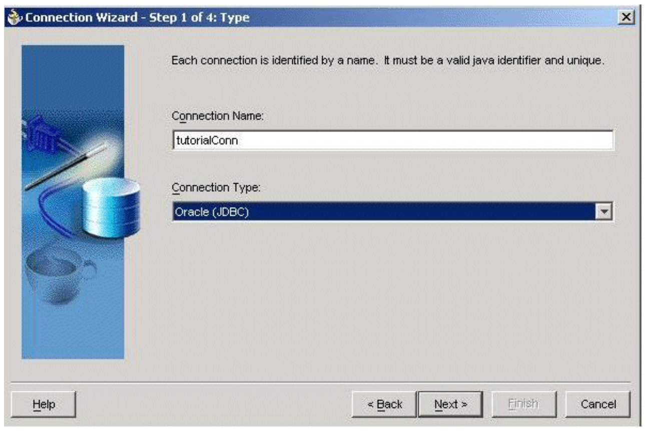
This wizard will help you construct a new connection for later use.

You will be asked for a connection name, authentication details and for particulars of the server connection. You may then also choose to test that a connection can be established based on the entered information.

Skip this Page Next Time

3. The Connection Wizard - Step 1 of 4: Type panel appears, as displayed in Figure 20–3. Create a connection named `tutorialConn` and choose **`Oracle (JDBC)`** from the Connection Type list. Click **Next**.

Figure 20–3 Connection Wizard - Step 1 of 4: Type



Connection Wizard - Step 1 of 4: Type

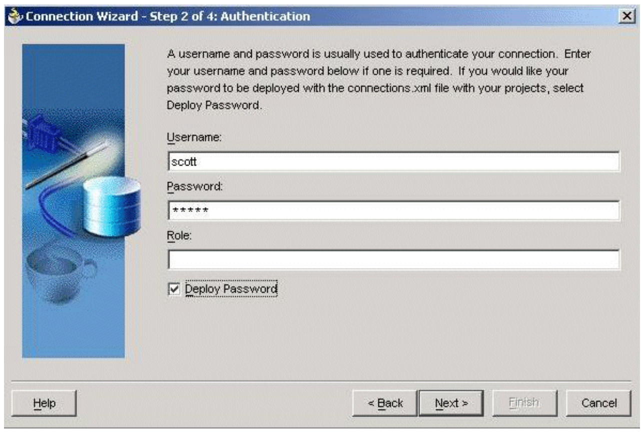
Each connection is identified by a name. It must be a valid java identifier and unique.

Connection Name: tutorialConn

Connection Type: Oracle (JDBC)

4. The Connection Wizard - Step 2 of 4: Authentication panel appears, as displayed in Figure 20–4. Enter `scott` as the user name and `tiger` as the password. Select the Deploy Password box. Click **Next**.

Figure 20–4 Connection Wizard - Step 2 of 4: Authentication Panel


Connection Wizard - Step 2 of 4: Authentication

A username and password is usually used to authenticate your connection. Enter your username and password below if one is required. If you would like your password to be deployed with the connections.xml file with your projects, select Deploy Password.

Username: scott

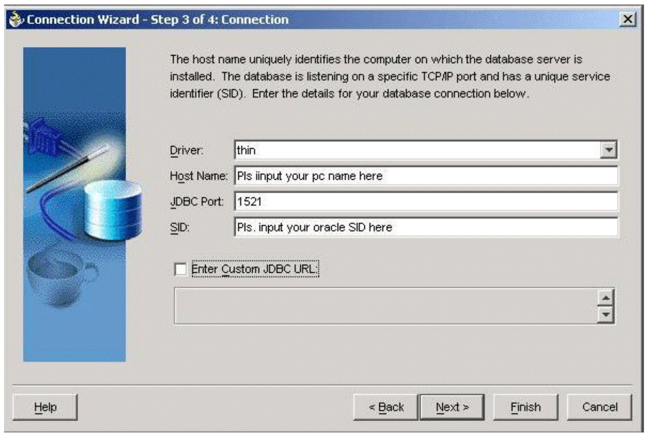
Password: *****

Role:

☑ Deploy Password

5. The Connection Wizard - Step 3 of 4: Connection panel appears, as displayed in Figure 20–5. Choose the **`thin`** option from the Driver list and enter your PC Host Name, JDBC Port number, and the database SID in the corresponding fields. Do not select the Enter Custom JDBC URL box. Click **Next**.

Figure 20–5 Connection Wizard - Step 3 of 4: Connection Panel


Connection Wizard - Step 3 of 4: Connection

The host name uniquely identifies the computer on which the database server is installed. The database is listening on a specific TCP/IP port and has a unique service identifier (SID). Enter the details for your database connection below.

Driver: thin

Host Name: Pls iinput your pc name here

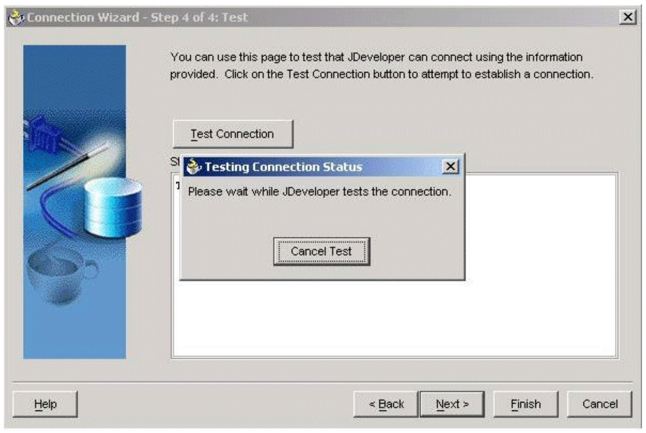
JDBC Port: 1521

SID: Pls. input your oracle SID here

☐ Enter Custom JDBC URL:

6. The Connection Wizard - Step 4 of 4: Test panel appears, as displayed in Figure 20–6. Click **Test Connection**. The Connection Wizard displays a connection status message. Click **Finish**.

Figure 20–6 Connection Wizard - Step 4 of 4: Test Panel

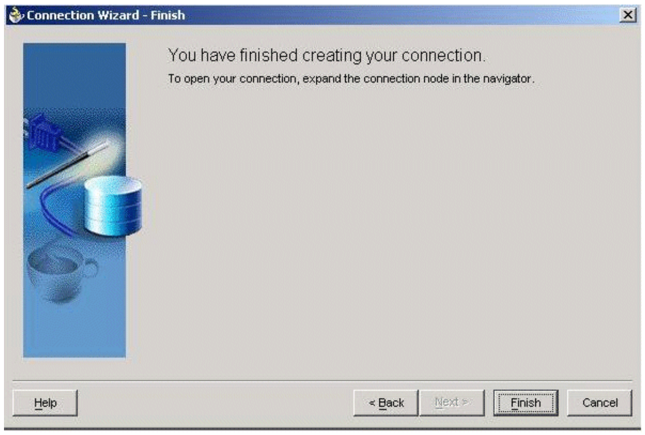

Connection Wizard - Step 4 of 4: Test
Test Connection
Testing Connection Status
Cancel Test
Help
< Back
Next >
Finish
Cancel

You can use this page to test that JDeveloper can connect using the information provided. Click on the Test Connection button to attempt to establish a connection.

Please wait while JDeveloper tests the connection.

You have finished creating the `tutorialConn`, as displayed in Figure 20–7.

Figure 20–7 Connection Wizard - Finish Panel

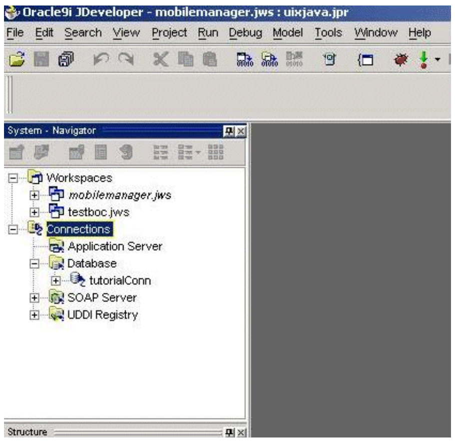


You have finished creating your connection.

To open your connection, expand the connection node in the navigator.

7. In JDeveloper, the `tutorialConn` icon appears in the System Navigator window under the Connections node, as displayed in Figure 20–8.

Figure 20–8 Tutorial Connection Icon in the System Navigator



In summary, Table 20–2 shows the values that you provide for this connection in the Connection Wizard.

Table 20–2 TutorialConn - Connection Wizard Description

Field Name	Value
Connection Name	`tutorialConn`
User name	`scott`
Password	`tiger`
Select a JDBC Driver	Thin
SID	Your Oracle database SID

20.2.1.2 Create the Connection to the Oracle Lite Database

Create the permanent connection used by the application between the back-end Oracle database and the Oracle Database Lite. This connection is used after the application is deployed for synchronization between the two databases—the back-end Oracle database and the local Oracle Lite database.

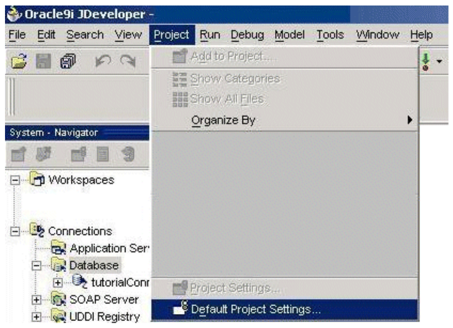
For this example, create the `WTGJdbc` connection, which uses the `oracle.lite.web.WTGJdbcDriver`.

Note: 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. Configure the project settings and include the Oracle Database Lite user library named `webtogo.jar`. Start JDeveloper and click the **Project** menu. As displayed in Figure 20–9, select the **Default Project Settings** option. Then select **Development->Configuration->Libraries** in the left side pane. Click **New Button**.

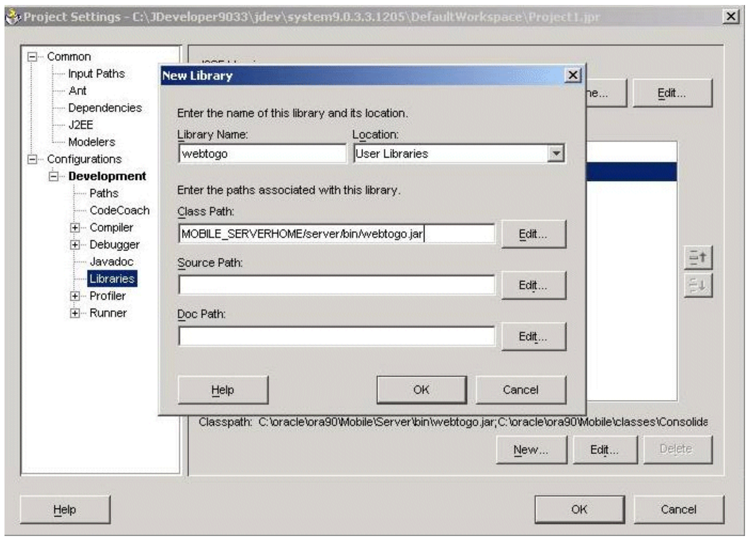
Figure 20–9 Choosing Default Project Settings



2. In the Project Settings panel, add a new library and name the new library as `webtogo`. Enter the `CLASSPATH` as given below and displayed in Figure 20–10. Click **OK**.

```
<mobile_serverhome>/server/bin/webtogo.jar
```

Figure 20–10 Adding a New Library and Classpath



Enter the name of this library and its location.

Library Name: webtogo

Location: User Libraries

Enter the paths associated with this library.

Class Path: MOBILE_SERVERHOME/server/bin/webtogo.jar

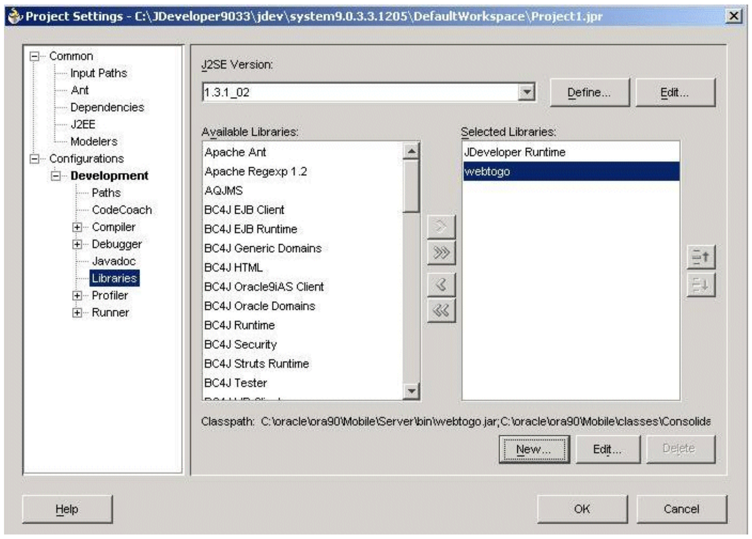
Source Path:

Doc Path:

Classpath: C:\oracle\ora90\Mobile\Server\bin\webtogo.jar;C:\oracle\ora90\Mobile\classes\Consolidε

3. Move the `webtogo` library from the Available Libraries list to the Selected Libraries list, as displayed in Figure 20–11. Click **OK**.

Figure 20–11 Moving the Webtogo Library to the Selected Libraries List



J2SE Version: 1.3.1_02

Available Libraries: Apache Ant, Apache Regexp 1.2, AQJMS, BC4J EJB Client, BC4J EJB Runtime, BC4J Generic Domains, BC4J HTML, BC4J Oracle9iAS Client, BC4J Oracle Domains, BC4J Runtime, BC4J Security, BC4J Struts Runtime, BC4J Tester

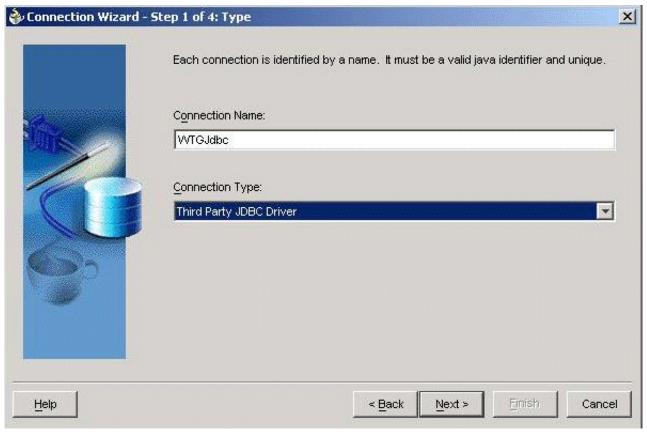
Selected Libraries: JDeveloper Runtime, webtogo

Classpath: C:\oracle\ora90\Mobile\Server\bin\webtogo.jar;C:\oracle\ora90\Mobile\classes\Consolide

4. After configuring project settings as mentioned in this step, create the `WTGJdbc` connection using the same method that you used to create `tutorialConnection`.

   To create the `WTGJdbc` connection, start JDeveloper and right-click the Connection object and choose **New Database Connection Option**. As displayed in Figure 20–12, the Connection Wizard - Step 1 of 4: Type panel appears. Enter `WTGJdbc` as the Connection Name and choose **`Third Party JDBC Driver`** as the JDBC Connection Type. Click **Next**.

Figure 20–12 Connection Wizard - Step 1 of 4: Type Panel



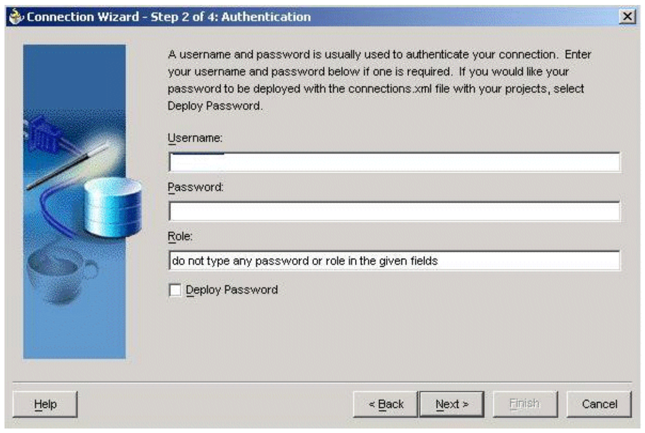
Each connection is identified by a name. It must be a valid java identifier and unique.

Connection Name: WTGJdbc

Connection Type: Third Party JDBC Driver

5. The Connection Wizard - Step 2 of 4: Authentication panel appears, as displayed in Figure 20–13. Do not enter any values in this panel. Click **Next**.

Figure 20–13 Connection Wizard - Step 2 of 4: Authentication Panel



A username and password is usually used to authenticate your connection. Enter your username and password below if one is required. If you would like your password to be deployed with the connections.xml file with your projects, select Deploy Password.

Username:

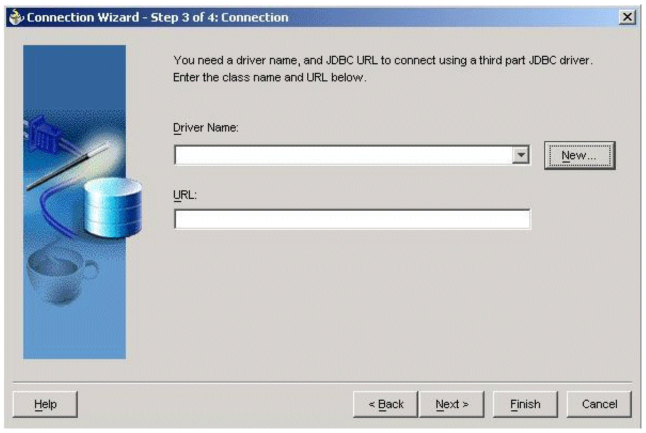
Password:

Role: do not type any password or role in the given fields

☐ Deploy Password

6. The Connection Wizard - Step 3 of 4: Connection panel appears, as displayed in Figure 20–14. Click **New**.

Figure 20–14 Connection Wizard - Step 3 of 4: Connection Panel



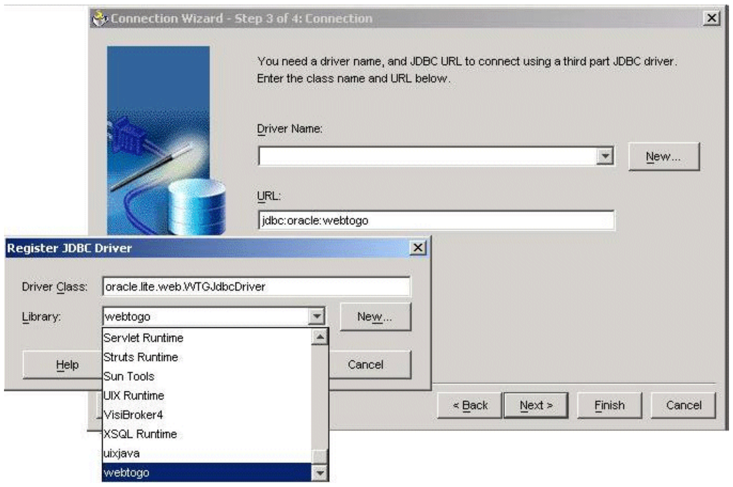
You need a driver name, and JDBC URL to connect using a third part JDBC driver.
Enter the class name and URL below.

7. The Register JDBC Driver dialog appears, as Figure 20–15 displays. Enter `oracle.lite.web.WTGJdbcDriver` as the Driver Class. Choose `webtogo` from the Library list and click **OK**. Enter the following URL.

   ```
   jdbc:oracle:webtogo
   ```

   Click **Next**.

Figure 20–15 JDBC Driver Dialog



You need a driver name, and JDBC URL to connect using a third part JDBC driver.
Enter the class name and URL below.

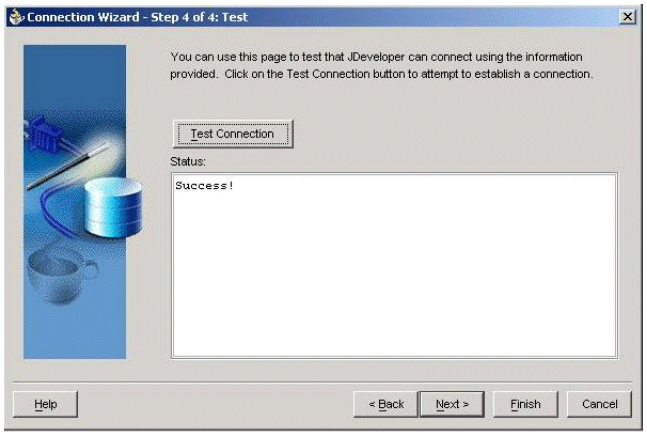
URL: jdbc:oracle:webtogo

Driver Class: oracle.lite.web.WTGJdbcDriver

Library: webtogo

8. The Connection Wizard - Step 4 of 4: Test panel appears, as Figure 20–16 displays. To test your `WTGJdbc` connection, click **Test Connection**. The Status box displays that the `WTGJdbc` connection has been created successfully. Click **Finish**.

Figure 20–16 Connection Wizard - Step 4 of 4: Test Panel



You can use this page to test that JDeveloper can connect using the information provided. Click on the Test Connection button to attempt to establish a connection.

```
Success!
```

In summary, Table 20–3 describes the values that are entered in the Connection Wizard to create the `WTGJdbc` connection:

Table 20–3 WTGJdbc Connection - Connection Wizard Description

Field Name	Values
Connection Name	`WTGJdbc`
Select a JDBC Driver	Third Party JDBC Driver
Class Name	`oracle.lite.web.WTGJdbcDriver`
Datasource URL	`jdbc:oracle:webtogo`

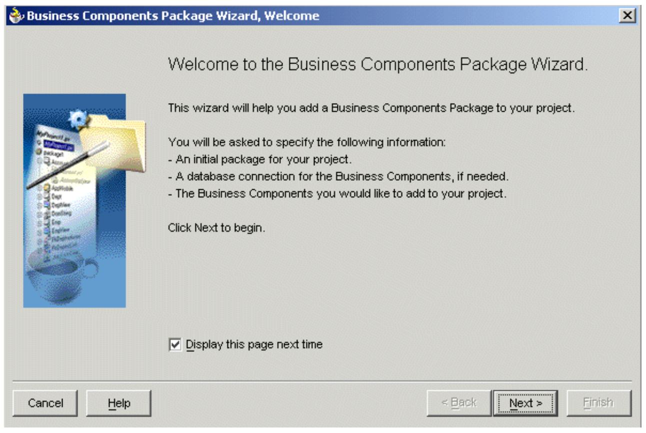
Note: In the Connection Wizard, enter values as specified in Table 20–2 and Table 20–3 only. Retain all other values as default values.

20.2.2 Creating the BC4J Component

Using JDeveloper, create the BC4J component named `tutorialapp`, as follows:

1. In JDeveloper, select **New** from the **File** menu. In the New dialog box that appears, the options—Projects in the left panel under General Categories and Empty Project in the right panel are pre-selected as defaults. Select **Workspace** to enable the OK button and click **OK**. JDeveloper creates a new empty project named `Project.jpr`.
2. Rename `Project.jpr` to `tutorialapp.jpr`, which creates a new project by that name. Select the project and then select **File->Rename**.
3. Right click `tutorialapp.jpr` in the JDeveloper workspace. Select **New Business Components Package**. As shown in Figure 20–17, the "Business Components Package Wizard, Welcome" dialog appears. Click **Next**.

Figure 20–17 The Business Components Package Wizard, Welcome Dialog


Business Components Package Wizard, Welcome

Welcome to the Business Components Package Wizard.

This wizard will help you add a Business Components Package to your project.

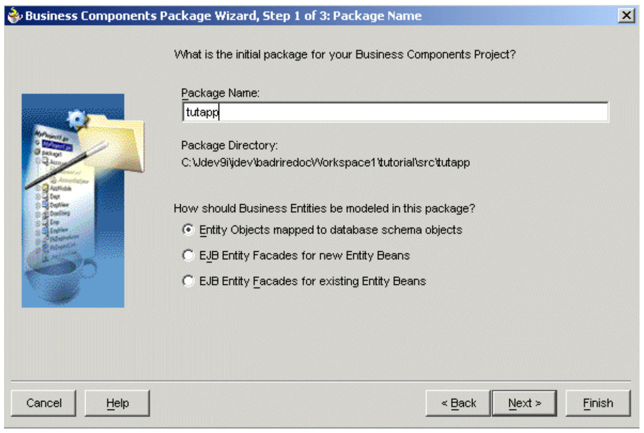
You will be asked to specify the following information:
- An initial package for your project.
- A database connection for the Business Components, if needed.
- The Business Components you would like to add to your project.

Click Next to begin.

☑ Display this page next time

4. The "Business Components Package Wizard, Step 1 of 3: Package Name" dialog appears, as illustrated in Figure 20–18. In the Package Name field, enter `tutapp`. Click **Next**.

Figure 20–18 Business Components Package Wizard Step 1 of 3: Package Name


Business Components Package Wizard, Step 1 of 3: Package Name

What is the initial package for your Business Components Project?

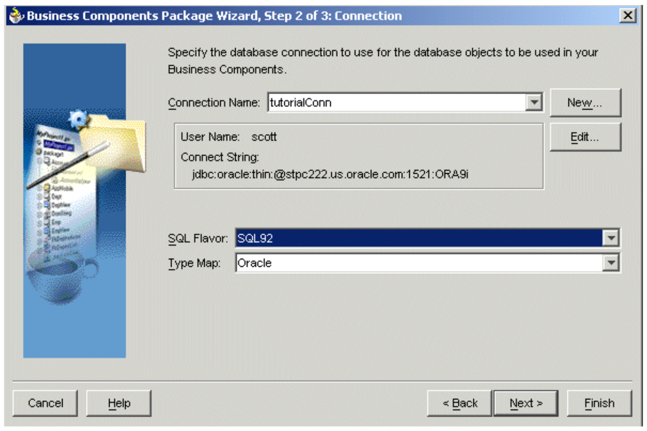
Package Name: tutapp

Package Directory:
C:\Jdev9i\jdev\badriredocWorkspace1\tutorial\src\tutapp

How should Business Entities be modeled in this package?
- ● Entity Objects mapped to database schema objects
- ○ EJB Entity Facades for new Entity Beans
- ○ EJB Entity Facades for existing Entity Beans

5. The "Business Components Package Wizard, Step 2 of 3: Connection" dialog appears, as depicted in Figure 20–19.

Figure 20–19 Business Components Package Wizard Step 2 of 3: Connection



Specify the database connection to use for the database objects to be used in your Business Components.

Connection Name: tutorialConn

User Name: scott

Connect String:
jdbc:oracle:thin:@stpc222.us.oracle.com:1521:ORA9i

SQL Flavor: SQL92

Type Map: Oracle

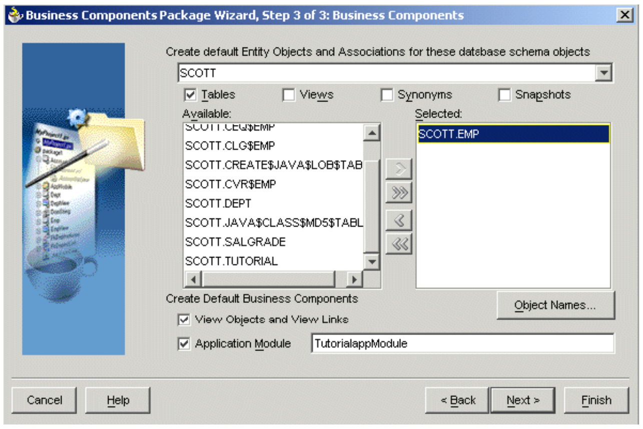
Select the values that are listed in Table 20–4 and click **Next**.

Table 20–4 Values for Business Components Package Wizard, Step 2 of 3: Connection

Field	Description
Connection Name	`tutorialConn`
SQL Flavor	SQL92
Type Map	Oracle

6. In the "Business Components Project Wizard, Step 3 of 3: Business Components" dialog, select `EMP` from the list displayed in the left panel and move it to the "Selected" list, as illustrated by the example in Figure 20–20. Click **Finish**.

Figure 20–20 Business Components Package Wizard, Step 3 of 3: Business Components Dialog


Business Components Package Wizard, Step 3 of 3: Business Components
Object Names...
Cancel
Help
< Back
Next >
Finish

Create default Entity Objects and Associations for these database schema objects

SCOTT

[✓] Tables [ ] Views [ ] Synonyms [ ] Snapshots

Available:
SCOTT.CEQ$EMP
SCOTT.CLG$EMP
SCOTT.CREATE$JAVA$LOB$TAB
SCOTT.CVR$EMP
SCOTT.DEPT
SCOTT.JAVA$CLASS$MD5$TABL
SCOTT.SALGRADE
SCOTT.TUTORIAL

Selected:
SCOTT.EMP

Create Default Business Components

[✓] View Objects and View Links

[✓] Application Module TutorialappModule

7. JDeveloper creates the `TutorialAppModule` BC4J component.

20.2.3 Configuring the BC4J Component to Use the WTGJdbc Connection

After you have completed developing and testing your application, switch the connection to use the `WTGJdbc` connection, which is the connection between the

Oracle database and Oracle Database Lite. To configure the BC4J component to use the `WTGJdbc` connection, perform the following steps.

1. Right-click on the `TutorialAppModule` and click the **Configurations...** option. The Configuration Manager appears.
2. In the Oracle Business Component Configuration dialog, click **Edit**. Choose `WTGJdbc` as the JDBC connection.
3. Click **OK**. The BC4J component is now configured to use the `WTGJdbc` connection.
4. Click **OK** in the Configuration Manager window.

20.2.4 Building and Deploying the BC4J Component as a Simple Archive

To build and deploy the BC4J component as a simple archive, perform the following steps.

1. Right-click the `tutorialapp.jpr` file and select the **Create Business Components Deployment Profiles** option. The Business Component Deployment Wizard appears.
2. Select the **Simple Archive Files** option from the list displayed and move it to the **Selected** list. Click **Next**. By default, this option is already selected.
3. The "Business Component Deployment Wizard Step 2 of 2: Simple Archive Files" appears. Under the 'Selected Platform - Simple Archive Files' section, accept the default Profile name.
4. Click **Next**. Click **Finish**. The `tutorialapp.bcdeploy` file is created under `tutorial.jpr`.
5. Right-click the file `tutorialapp.bcdeploy` and select **Deploy**. JDeveloper creates two jar files—`tutorialappCSCommon.jar` and `tutorialappCSMT.jar`.

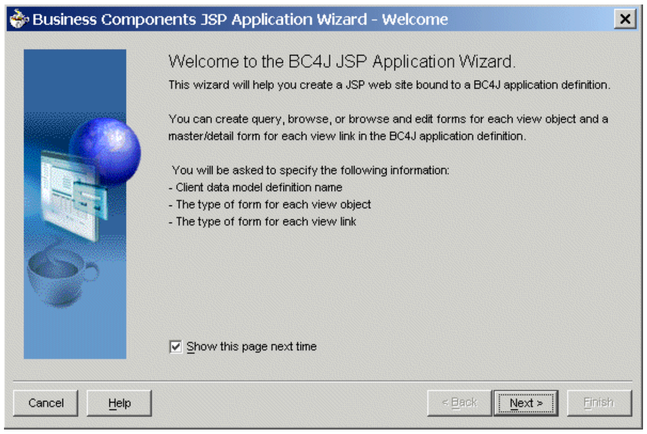
Note: To check the location of the `.jar` files that you created, check the Deployment Log window in the JDeveloper UI.

20.2.5 Writing the JSP Application to Access the BC4J Component

To write the JSP application that will access the BC4J component, perform the following steps.

1. In JDeveloper, select File->New Options. Select **Empty Project** in the right-hand pane. Click **OK**. The system automatically creates a new empty project called `MyProject.jpr`.
2. Under the File menu, select **Rename...** and rename `MyProject.jpr` to `tutorialclientapp.jpr`.
3. Click the `tutorialclientapp.jpr` file in the JDeveloper workspace. Select **File->New**. Click **Web Tier** and select **JSP for Business Components**. Click **OK**. The Business Components JSP Application Wizard appears. Click **Next**.

Figure 20–21 Business Components JSP Application Wizard - Welcome Dialog



Welcome to the BC4J JSP Application Wizard.

This wizard will help you create a JSP web site bound to a BC4J application definition.

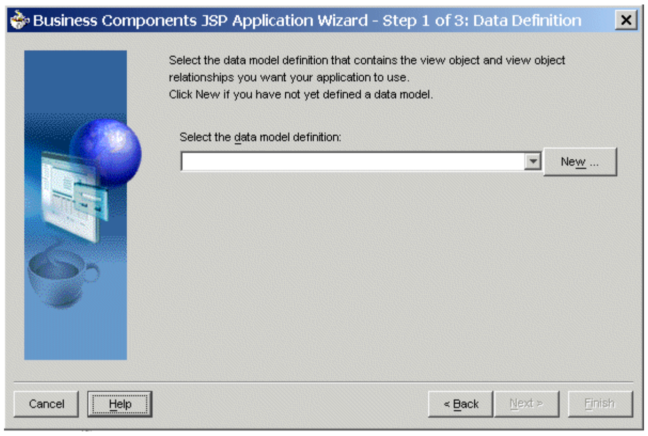
You can create query, browse, or browse and edit forms for each view object and a master/detail form for each view link in the BC4J application definition.

You will be asked to specify the following information:
- Client data model definition name
- The type of form for each view object
- The type of form for each view link

☑ Show this page next time

4. Click **New**. The 'Business Components JSP Application Wizard' appears. Click **Next**. The wizard displays the 'Business Components JSP Application Wizard - Step 1 of 3:Data Definition' dialog.

Figure 20–22 Business Components JSP Application Wizard - Step 1 of 3: Data Definition Dialog

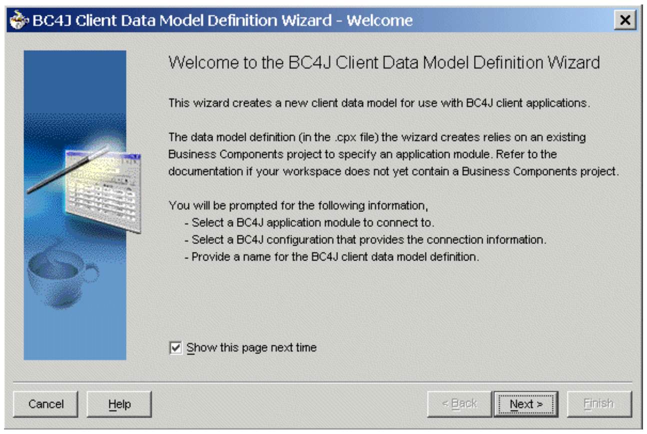

Business Components JSP Application Wizard - Step 1 of 3: Data Definition
New ...
Cancel
Help
< Back
Next >
Finish

Select the data model definition that contains the view object and view object relationships you want your application to use.
Click New if you have not yet defined a data model.

Select the data model definition:

5. Click **New** in the 'Business Components JSP Application Wizard - Step 1 of 3:Data Definition' dialog. The 'BC4J Client Data Model Definition Wizard' appears. Click **Next**.

Figure 20–23 BC4J Client Data Model Definition Wizard - Welcome Dialog


BC4J Client Data Model Definition Wizard - Welcome

Welcome to the BC4J Client Data Model Definition Wizard

This wizard creates a new client data model for use with BC4J client applications.

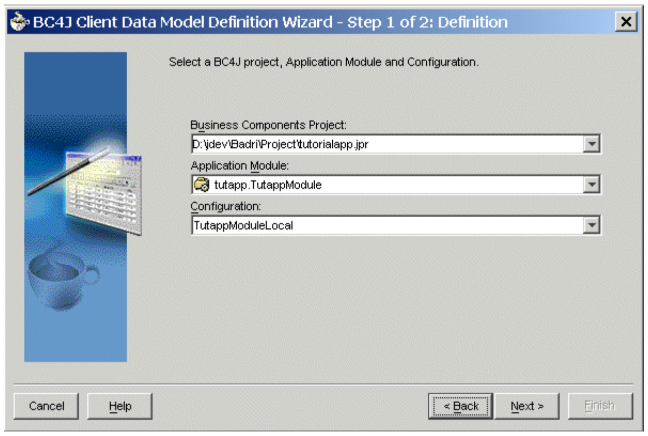
The data model definition (in the .cpx file) the wizard creates relies on an existing Business Components project to specify an application module. Refer to the documentation if your workspace does not yet contain a Business Components project.

You will be prompted for the following information,
- Select a BC4J application module to connect to.
- Select a BC4J configuration that provides the connection information.
- Provide a name for the BC4J client data model definition.

☑ Show this page next time

6. The 'BC4J Client Data Model Definition Wizard: Step 1 of 2: Definition' appears.

Figure 20–24 BC4J Client Data Model Definition Wizard, Step 1 of 2: Definition Dialog


BC4J Client Data Model Definition Wizard - Step 1 of 2: Definition

Select a BC4J project, Application Module and Configuration.

Business Components Project: D:\jdev\Badri\Project\tutorialapp.jpr

Application Module: tutapp.TutappModule

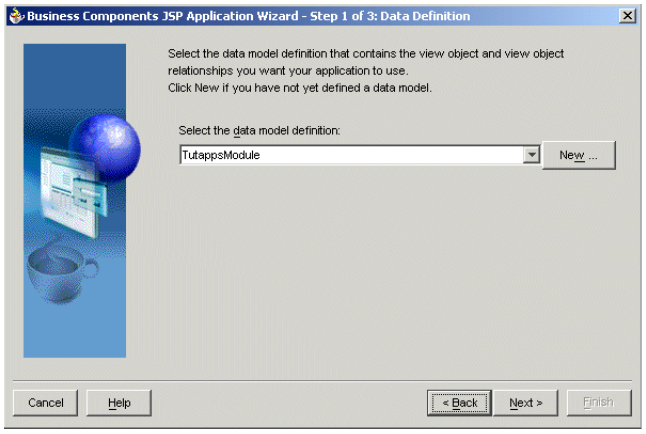
Configuration: TutappModuleLocal

7. Verify the default values and click **Next**. `TutappModule` appears as the default definition name in the 'BC4J Client Data Model Definition Wizard - Step 2 of 2: Definition Name' dialog. Click **Next**.

8. Click **Finish**. The 'Business Components JSP Application Wizard' dialog appears. Click **Next**.

9. In the 'Business Components JSP Application Wizard - Step 1 of 3: Data Definition' dialog, select `TutappModule` as the data model definition, as displayed in Figure 20–25. Click **Next**.

Figure 20–25 Business Components JSP Application Wizard - Step 1 of 3: Data Definition Dialog


Business Components JSP Application Wizard - Step 1 of 3: Data Definition

Select the data model definition that contains the view object and view object relationships you want your application to use.
Click New if you have not yet defined a data model.

Select the data model definition: TutappsModule

10. Accept the default selections and click **Next** in the two dialog boxes that appear: 'Business Components JSP Application Wizard - Step 2 of 3: View Object Forms' and 'Business Components JSP Application Wizard - Step 3 of 3: View Link Form'. The 'Summary' window appears. Click **Finish**.

20.2.6 Deploying the JSP Application as a Simple Archive

To deploy the JSP application as a simple archive, perform the following steps.

1. In JDeveloper, click the `tutorialclientapp.jpr` file and select the file named `tutappclient_jpr_war.deploy`. Right click on the `tutorialclient_jpr_war.deploy` file.
2. Select 'Deploy to WAR file'. The file `tutappclient_jpr_war.war` is created. To track the deployment location, check the 'Deployment Log' text area in JDeveloper.

20.3 Packaging the JSP Application

To package the JSP application, perform the following steps.

1. Create a sub-directory called `bc4jtutapp` under the following location:

   ```
   <Mobile_ServerHome>\Mobile\Sdk\wtgsdk\root
   ```

2. Unzip the `tutappclient_jpr_war.war` file into the `bc4jtutapp` directory.
3. Edit all the JSP files to delete the following.

   ```
   charset=windows-1252
   ```

   as shown below:

   ```
   <%@page language="Java"errorpage="errorpage.jsp"
   ContentType="text/html;charset=windows-1252"%>\
   ```

4. Edit the `web.xml` file and insert the following tag at the end just before closing `</web-app>`.

   ```
   <filter>
    <filter-name>CheckSessionFilter</filter-name>
    <filter-class>oracle.lite.web.CheckSessionFilter</filter-class>
   </filter>
   ```

```
<filter-mapping>
 <filter-name>CheckSessionFilter</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
```

5. Using the Command Prompt window, run the Packaging Wizard and provide the screen inputs that are listed and described in Table 20–5.

Table 20–5 Packaging Wizard Input Details

Screen	Input	Details
Platform	Web-To-Go	N/A
Application	Application Name	BC4J Oracle Database Lite Tutorial Application
Application	Virtual Path	`/bc4jtutorial`
Application	Description	Oracle Lite Tutorial Application
Application	Application Classpath	no input
Application	Default Page	`main.html`
Application	Local Application Directory	`<`*`ORACLE_HOME`*`>\Mobile\Sdk\wtgsdk\root\bc4jtutapp`
Files	The Packaging Wizard loads all files in a directory under the Local Application Directory.	N/A

Table 20–6 lists the servlet names and their corresponding classes that are created in the Packaging Wizard by default.

Table 20–6 Servlet Names and Classes

Screen	Servlet Name	Servlet Class
Servlet	`EMDServlet`	`oracle.jbo.server.emd.EMDServlet`
Servlet	`ImageServlet`	`oracle.cabo.image.servlet.ImageServlet`
Servlet	`TecateServlet`	`oracle.cabo.image.servlet.TecateServlet`
Servlet	`BajaServlet`	`oracle.cabo.servlet.BajaServlet`
Servlet	`OrdPlayMediaServlet`	`oracle.ord.html.OrdPlayMediaServlet`

Table 20–7 lists server side and client side database values that you must specify in the Packaging Wizard.

Table 20–7 Database Values

Screen	Input	Details
Database	Server side Database User Name	`scott`
Database	Number of Connections	0
Database	Share Connections	Do not select this check box
Database	Client side Database Name	Client DB

6. Under the Snapshots section, click **Import**. You can now connect to the Oracle Database by providing the following values in the "Connect to Database" dialog.

Table 20–8 lists values that you must specify in the Connect to Database dialog.

Table 20–8 Connect to Database Dialog Description

Field	Description
User Name	`scott`
Password	`tiger`
Database URL	`jdbc:oracle:thin:@DatabaseHostMachineName:port:SID`

7. After specifying the Database Connection values, select `Emp` from the list of tables.
8. Click **Edit** and change the weight from 0 to 1.
9. Retain the default values for Roles, Sequences, DDLs, and Registry fields.
10. Package the application into a JAR file.

20.4 Publishing and Configuring the JSP Application from the Mobile Manager

To 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, you must specify the appropriate port number.

3. Login into the Mobile Server using the administrator user name 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.

20.5 Testing the BC4J Application

Perform the following to test your 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 "Oracle Lite Tutorial Application".
7. Test the application by executing the BC4J application on the Webt-to-Go Mobile client, as described in Section 20.6, "Running the BC4J Application on the Mobile Client for Web-to-Go".

20.6 Running the BC4J Application on the Mobile Client for Web-to-Go

To execute the BC4J application on the Mobile Client for Web-to-Go, perform the following steps.

1. Using the following URL, check the Oracle database server IP address setup.

   ```
   http://Server_IP_Address/setup
   ```

2. Download and install the Mobile Client for Web-to-Go with BC4J support.
3. Using the following URL, check the local host in the client machine.

   ```
   http://localhostname
   ```

4. Log in to the client machine using `tutorial` as both the user name and password.
5. After the client machine synchronizes the application and data from the server, click the 'Oracle Lite Tutorial Application' link to test the application on the client machine.

20.7 Deploying the Sample Application

To deploy the sample application, perform the following steps.

1. Log in to the database as a system user. If the `SCOTT` schema does not exist already, execute the `bc4j.sql` script.
2. Publish the `OracleLite_BC4J_Tutorial.jar` file. It is found under the following directory.

   ```
   <mobile_server_home>\mobile\Sdk\wtgsdk\src\bc4jtutorial>
   ```

   Using the Mobile Manager, publish the above `.jar` file into the Mobile Server and enter the following virtual path.

   ```
   /bc4jtutorial
   ```

3. Click **Mobile Manager** and click **Applications**.
4. Click the 'Oracle Lite BC4J Application' link. The Properties page appears.
5. Enter `tiger` as the database password and click **Save**.
6. Navigate back to the Mobile Manager home page and click the **Users** link. In case the user `tutorial` doesn't exist already, add this user, as described in Section 20.5, "Testing the BC4J Application".
7. Click **Applications** and select 'Oracle Lite BC4J Application.' To provide access to the 'Oracle Lite BC4J Application', click **Access** and assign the access privilege to the user named `tutorial`.
8. Using the following URL, browse the client machine with BC4J support.

   ```
   http://servername:port/webtogo/setup
   ```

9. Download and install the Mobile Client for Web-to-Go.
10. If not started already, start the Mobile Client for Web-to-Go.
11. Log in to the Mobile Client for Web-to-Go with `tutorial` as the user name and password.

12. Upon completion of the Synchronization process, the system displays the 'Oracle Lite BC4J Application' link.
13. Click the 'Oracle Lite BC4J Application' link to access the BC4J tutorial application.

21

Building Offline Mobile Applications for Win32: A Tutorial

This document guides you through the Mobile application development process for the Win32 platform through a tutorial. Topics include:

- Section 21.1, "Overview"
- Section 21.2, "Developing Offline Mobile Applications for Win32"

21.1 Overview

To demonstrate the steps involved in building offline Mobile applications for the Win32 platform, this tutorial presents a simplified Mobile field service example.

21.2 Developing Offline Mobile Applications for Win32

Let us assume that you have a `TASK` table on the server that contains information about tasks that must be accomplished by your Mobile field service technicians for a day. Listed below is the `TASK` table structure. Each row in the `TASK` table describes work to be done at a customer site.

- `TASK(ID number(4) primary key`
- `Description varchar(40) not null`
- `CustName varchar(30) not null`
- `CustPhone varchar(12)`
- `CustStAddr varchar(40) not null`
- `CustCity varchar(40) not null`
- `Notes varchar(100)`

Let us also assume that you have three service technicians, Tom, Dick, and Harry. You want to assign all the tasks in the City of Cupertino to Tom, those in the City of Mountain View to Dick, and those in the City of Palo Alto to Harry. You envision your application to work as follows:

Each service technician has a laptop that he uses to obtain his task list in the morning. He will perform the task during the day and will update the Notes column of a task with information about its status or what he has done. At the end of his work day, the service technician uploads his changes to the server.

We will assume the following environment for your application.

- The Mobile Server is installed on the machine called `mserver`.

- The test Oracle database that is used to store the application data and the Mobile Server Repository is installed on the machine `oradbserver` with the listener on port 1521. The Oracle database instance name is `orcl`. We will assume that you can log in to the database with the user name `master` and password `master`. You can substitute any user for `master` so long as the user has the right privileges.
- You have already installed the Mobile Development Kit on your development machine.

Our implementation plan is as follows. The exact sequence of commands for each step is given later.

1. Create the `TASK` table in the `oradbserver` and insert some rows into it. This step is not needed if you already have a database that contains a table similar to `TASK`.
2. Use the Packaging Wizard to define an application called Mobile Field Service. Create one publication item based on the `TASK` table for the application. Publish the application (which has no application files) to the Mobile Server.
3. 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.
4. 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.
5. 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.
6. Synchronize the changes you made in the snapshot with the server database by running the Mobile Sync application again.
7. 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.
8. 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.
9. 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
10. Use the Packaging Wizard to update the application to include the above program.

The Mobile Server is now ready for real life testing.

21.2.1 Command Sequence

The following sections describe the command sequence.

21.2.1.1 Step 1. Create TASK Table on the Server Database

We will use the Oracle9*i* thin JDBC driver to connect to the Oracle database running in the `oradbserver` machine. Ensure that the thin JDBC driver (<*ORACLE_HOME*>`\jdbc\lib\classes12.zip`) file is included in your `CLASSPATH` environment variable. We will 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\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
```

21.2.1.2 Step 2. Define a Publication Item and Publish the Application

We will now use the Packaging Wizard to create a publication item for your application.

To use the Packaging Wizard, type the following command at the Command Prompt.

```
d:\> wtgpack
```

The Packaging Wizard appears.

Select the 'Create a new application' option and click **OK**.

In the next panel, select the 'Oracle Lite WIN32:US' platform from the list of 'Available Platforms'. This action prompts the Packaging Wizard to create a Windows 32 application. Click **Next**.

The next screen is for entering application information. We will call our application "Mobile Field Service". We will publish it in the `/MFS` directory on the Mobile Server. All our application files will be stored in the directory `D:\MFSDEV\Win32` on the development machine. We need to use the Win32 sub-directory under the development directory for the Windows 32 application. The *Oracle Database Lite Tools and Utilities Guide* discusses the directory naming convention used by the Packaging Wizard.

Enter the following information on the screen.

Application Name: `Mobile Field Service`

Virtual Path: `/MFS`

Description: `Field Service Task Assignment`

Local Application Directory: `D:\MFSDEV` (note: you don't specify the Win32 subdirectory here)

Click **Next**.

The next screen allows you to include any files such as the executable and image files that the application will need. It will read the `D:\MFSDEV\Win32` directory and will display all the files found there. For now, we only want to create snapshots and so we will not include any files yet. Click **Next**.

The next panel is used to enter the database name that is for the client database. Enter the following:

Client Side

Database Name: `MFS`

Click **Next**.

The next screen enables you to define publication items that will become snapshots on the client database. We will create the publication based on the `TASK` table that we have defined on the server. We do this by importing the table into the Packaging Wizard. Click the "Import" button towards the bottom of the screen. You will be prompted to enter the server login information. Enter the following:

User Name: `master`

Password: `master`

Database URL: `jdbc:oracle:thin:@oradbserver:1521:orcl`

Click the **OK** button.

A dialog appears listing all the tables that are available. Select the `TASK` table and click the **Add** button. Click the **Close** button. The `TASK` table is now listed in the "snapshots" table. Select the row for the `TASK` table and click the **Edit** button.

The next screen enables you to enter the subsetting query for the snapshot. We will create an updatable snapshot that can be refreshed incrementally (fast refresh). If there is a conflict in updates, we want the server changes to win. So we enter the following information.

Set the value of Weight to 1 and then click the tab "Win32".

To explain how table weight works. Table weight is an integer property of association between publications and publication items. The Mobile Server uses table weight to determine the order in which to apply client Operations to master tables within each publication as listed below.

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.
4. The value assigned must be an integer between 1 and 1023.

Table weight is applied to publication items within a specific publication. For example, a publication can have more than one publication item of weight "2", which would have `INSERT` operations performed for any publication item of a lower weight within the same publication.

Continuing with the steps to package and publish the application, in the next screen after setting the value of Weight to 1 and clicking the tab "Win 32", enter the following:

Create on Client: `check`

Updatable?: `check`

Conflict resolution: `select the "Server Wins" option`

Refresh type: `select the "Fast Refresh" option`

Template: `select * from master.task where CustCity = :city`

Click **OK**. This brings you back to the previous screen. The template query contains a variable (subscription parameter) named "city". Later, when you provision the application to a user, you will be prompted to enter the value for it.

Click **Finish**. A dialog appears. Select "Publish the current application" option and click **OK**. A dialog appears prompting you to enter information about the Mobile Server.

Enter the following:

Mobile Server URL: `mserver`

Mobile Server User Name: `Administrator`

Mobile Server Password: `admin`

Repository path: `/MFS`

Click **OK**. If you get the message "Application Published Successfully", click the **OK** button and then click the **EXIT** button. You have successfully published an application that has no files and one publication item.

21.2.1.3 Step 3. Create Users and Subscriptions

To create users on the Mobile Server, you use the Mobile Manager. To use the Mobile Manager, you must log in to the Mobile Server as `administrator`. To log in to the Mobile Server, perform the following actions.

1. Using a browser, browse the Mobile Server page by entering the following URL.

   `http://<mobile_server>/webtogo`

   (For historical reasons, the term "webtogo" instead of "Mobileserver" is used in the URL.)

   The Logon page appears. Enter the "administrator" as the User Name and "admin" as the password.

   Click the **Logon** button.

2. The Mobile Server farms page appears. Click your Mobile Server's link. Your Mobile Server's home page appears. To display your applications, click the Applications link. Click the Application Name link for which you will add users. On the Users page, click the Add User button. The Add User page appears.

   You will use this screen to create users Tom, Dick and Harry. We will only show how to create user Tom in the following commands.

3. Enter the following information.

   Display Name: `Tom Jones`

   User Name: `Tom`

   Password: `tomjones`

   Password Confirm: `tomjones`

   System Privilege: `User`

4. Click the **Save** button. The Mobile Manager displays a confirmation message. Click **OK**.
5. To provide access to these users, click the Access link. The Access page lists existing applications. Select the Mobile Field Service application by checking the "Access" box for it. Click the **Save** button. A message box appears. Click the **OK** button on the message box. You have just granted user Tom the permission to execute the Mobile Field Service application.
6. To create a data subset in your database during application installation, you will now define subscriptions for these users. Click the Data Subsetting link. The Data Subsetting parameters page appears. For the Mobile Field Service application, we have only one publication item and it has only one subscription parameter called "city". Enter the value "Cupertino" (without the quotes) for the value of "city" and click the **Save** button. The Mobile Manager displays a confirmation message. Click the **OK** button.

You have successfully created the user Tom, granted him the privilege to execute the application "Mobile Field Service", and assigned him all the tasks in the City of Cupertino.

Repeat the above steps for users Dick and Harry.

21.2.1.4 Step 4. Install the Oracle Database Lite 10*g* Client and the Mobile Field Service Application and Data

In a production system, Mobile users such as Tom, Dick, and Harry would visit the setup page of the Mobile Server and download the Oracle Database Lite 10*g* Windows 32 client. They will then run the Mobile Sync application to download the Mobile Field Service application and the corresponding data subsets. After downloading the application and data, they will use the Mobile Field Service application and occasionally run Mobile Sync to synchronize the data with the server.

However, we are still in the development process and the developer has not yet developed the real Mobile Field Service application. So far, the installation and initial synchronization will only create a client Oracle Database Lite database that has a snapshot called `TASK` in it.

The developer will install and perform the initial synchronization as user Tom to retrieve an Oracle Database Lite database with a snapshot in it. He will then test the synchronization process before he develops the application.

On the machine where you installed the Mobile Development Kit, browse the setup page of the Mobile Server. The URL is `http://<your_mobile_server/webtogo/setup`. The setup page displays a list of supported platforms. Download the Mobile Client for Win32 by clicking on the appropriate link. To install the client, choose a directory, say `D:\MFS`. Browse the directory and familiarize yourself with its structure. Start the Commend Prompt and enter the following:

```
C:>D:

D:>cd MFS\Mobile\bin

D:\MFS\Mobile\bin>msync
```

This will run the Mobile Sync application, downloaded as part of the application installation. (You can also run the Mobile Sync application in your `\sdk\bin` directory.) A 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).

21.2.1.5 Step 5. Browse the TASK Snapshot and Update a Row

Start the Command Line and enter the following:

```
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.

21.2.1.6 Step 6. Synchronize the Change with the Server

Before you synchronize your change with the server, you must ensure that the MGP process is running. To ensure that the MGP process is running, follow the directions given in Step 3 and log on to the Mobile Server as administrator. Navigate to your Mobile Server home page and click the Applications link. Click the Job Scheduler link in the bottom section of this page and click the MGP Data Synchronization link. Click the MGP/Apply Compose Cycles link and schedule the MGP process on the MGP/Apply Compose Cycles page.

On your development machine, run the Mobile Sync application as described in Step 4. When the synchronization is successfully completed, your changes will be reflected in the server database.

21.2.1.7 Step 7. Check your changes on the server and modify a server record

Connect to the server database and issue the following SQL statements:

```
D:> msql master/master@jdbc:oracle:thin:@oradbserver:1521:orcl

SQL> select * from task;
```

You will see your changes reflected in the table. Now we will make a change in this table.

```
SQL> update task set description = 'Garbage Disposal Leaking',
Notes= 'Urgent: house is getting flooded' where id = 2;
```

```
1 row(s) updated
SQL> commit;
Commit complete
SQL> exit
```

21.2.1.8 Step 8. Synchronize again to get the server changes

On your development machine, run the Mobile Sync application as described in Step 4. When the synchronization is successfully completed, perform the following:

```
D:>MFS\Mobile\bin>msql system/manager@jdbc:polite:tom_mfs
SQL> select * from task;
```

You will see two rows displayed. The second row displays the changes that you made on the server.

21.2.1.9 Step 9. 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\mfs
```

(The`\src` directory contains the source and the makefile for it.)

This example is very simple and does not use any UI widgets. It 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 publish this program to the Mobile Server, copy the `mfs.exe` file into the directory `D:\MFSDEV\Win32`.

21.2.1.10 Step 10. Republish the Application with the Application Program

Use the Packaging Wizard to republish the application. From the Command Line, enter the following:

```
D:>wtgpack
```

On the first screen, select the "Edit an existing application" option. From the drop down list, select "Mobile Field Service" and click the **OK** button.

In the next screen, click the **Files** tab. You should see the **MFS.exe** file listed in the "File Name" window. Click **OK**.

In the next screen, select the "Publish the current application" option and click OK. You will be prompted to enter the login information for the Mobile Server. Click the OK button after entering the information. You will then see a message box warning you that the application already exists on the Mobile Server and whether you want to overwrite it. Click the YES button.

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.

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
```

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.

22 Building Offline Mobile Applications for Windows CE: A Tutorial

This document describes how to build a Visual Basic.NET (Visual Studio.NET 2003) application using the Oracle Database Lite 10*g* ADO.NET interface for Pocket PC. It enables you to implement offline Mobile applications for the Pocket PC using Oracle Database Lite 10*g*. It provides you with the complete framework to build, deploy, and manage offline Mobile applications. Oracle Database Lite 10*g* supports various application models for the Pocket PC by supporting industry standard interfaces such as ODBC, JDBC, and ADO.NET. Topics include:

22.1 Overview

This document guides you through the entire offline Mobile application implementation process using a sample Pocket PC application. The tutorial enables you to create, deploy, administer, and use a Pocket PC Windows CE application.

The sample Pocket PC application is based on typical activities of delivery personnel in the Transportation and Logistics industry. The day-to-day operations of such personnel involve package pick-up and delivery. A delivery person collects the complete delivery package list and the package delivery destination information for the day, before he leaves the dispatch center on his Pocket PC. As the truck driver also carries information related to package pick-up and delivery with him, the delivery person can work offline and update the package pick-up and delivery status on his Pocket PC. Later, he can synchronize his updated information with the central server running in the dispatch center over any wireless network.

22.1.1 Before You Start

This tutorial assumes that the Mobile Server is installed on the same desktop that is used for Pocket PC application development. Before starting the offline Mobile application development process, you must ensure that the development computer and the client device meet the requirements specified below.

22.1.1.1 Application Development Computer Requirements

You must configure and install the following components on the development computer.

Table 22–1 lists the configuration and installation requirements for the Mobile application development computer.

Table 22–1 Application Development Computer Requirements

Requirement	Description
Windows NT/2000/XP User Login	The login user on the Windows NT/2000 development computer must have "Administrator" privileges.
Installed Java Components	Java Development Kit 1.4.2 or higher.
Installed Oracle Database Lite 10*g* Components	Oracle Database 8.1.7 or higher. The Mobile Server (Oracle Database Lite 10*g* CD-ROM). The Mobile Development Kit (Oracle Database Lite 10*g* CD-ROM).
Installed Pocket PC Components	Microsoft Active Sync 3.7.1 or higher. Microsoft eMbedded Visual Toolkit 3.0

22.1.1.2 Client Device Requirements

You must connect the client device to the desktop and install the Oracle Database Lite 10*g* client for Pocket PC on the device. For more information on how to install the Mobile Client on the device, see Section 22.5.1, "Installing the Oracle Database Lite Mobile Client for Pocket PC".

22.2 Developing the Application

This section explains how to develop and test the Pocket PC Transport application using the Mobile Development Kit for Pocket PC. The Pocket PC Transport application is written in Visual Basic.NET (Visual Studio.NET 2003).

To develop and test the Pocket PC Transport application, perform the following tasks.

1. Create database objects in the Oracle database.
2. Write the application code.
3. Compile the application.

22.2.1 Creating Database Objects in the Oracle Server

During deployment, the Mobile Server automatically creates the Oracle Database Lite 10*g* database in the client device along with the requisite tables and data. To publish the application, users must create database objects in the Oracle database.

22.2.1.1 The Pocket PC Transport Application Database Objects

The Pocket PC Transport application uses the following database objects.

1. Packages Table
2. Routes Table
3. Trucks Table

Table 22–2 lists the columns of packages that enable you to store all information about the package.

Table 22–2 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 22–3 lists the columns of routes that enable you to store all information about a route.

Table 22–3 Routes Table

Column	Description
`ROUTE_NO`	Route Number (Primary Key)
`ROUTE_NM`	Route Name
`EST_TIME`	Estimated Time

Table 22–4 lists columns of trucks that enable you to store all information about the availability status and destination information for a truck.

Table 22–4 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

1. The `master` schema is available in the Oracle Database Server. If the `master` schema is not available, enter the following command using the Command Prompt window.

Note: Ensure that the `CLASSPATH` includes `classes12.jar` or `classes12.zip`.

```
> msql system/manager@jdbc:oracle:thin:@<HOST>:<PORT>:<Service_Name>
SQL> create user master identified by master;
SQL> grant connect,resource to master;
```

The variable `<HOST>` refers to the machine name where the Oracle database is installed.

The variable `<PORT>` refers to the Oracle database listener port.

2. Enter the following commands to create database objects in the Oracle Database Server.

```
> cd ORACLE_HOME\Mobile\Sdk\samples\ado.net\Transport

> msql master/master@jdbc:oracle:thin:@<HOST>:<PORT>:
      <Service_Name> @create.sql
```

Note: While entering the above command to create database objects, you must include a mandatory space between `"<Service_Name>"` and `"@create.sql"`.

22.2.2 Writing the Application Code

The Pocket PC Transport application's Visual Basic.NET (Visual Studio.NET 2003) is readily available with the sample application. The following section explains the code written for the Transport application and is presented below.

22.2.2.1 Transport Module (Transport.vb)

To open a database connection, you must declare a connection object. In this tutorial, the connection object 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 a DSN named `TRANSPORT`. This DSN name is mentioned in the Packaging Wizard. For more information refer, Section 22.3.2, "Defining the Application Connection to the Oracle Database Server".

22.2.2.2 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.

22.2.2.3 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.

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.

```
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 OracleAdapter's `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)
```

22.2.2.4 Create Package (frmNew.vb)

This form allows users to create a new package entry in Oracle Database Lite. During the form's `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.

```
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()
```

22.2.3 Compiling 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 2003).

22.2.3.1 Creating CAB Files

To build CAB files for the Transport Tutorial application, right click on the 'Transport' project tree view object on the 'Solution Explorer' window of Visual Studio.NET 2003. Choose the 'Build CAB File' object from the popup menu.

To create the CAB file, select the **Application Install Wizard...** submenu from the **Remote Tools** option under the **Tools** menu in the Visual Basic.NET (Visual Studio.NET 2003) IDE.

1. Open the Project file "`.ebp`" of the application by entering the following value.

   `<`*`ORACLE_HOME`*`>\Mobile\Sdk\samples\ado.net\Transport`

2. Enter the directory name of the "`.vb`" file that you created and saved in the previous section.
3. Enter a directory name to store the CAB files. For example: "`C:\Transportinstall`.
4. Select the required processor for which you want to create a CAB file. For example, ARM 1100.
5. The Application Install Wizard displays default Active X Controls. Accept the default controls and click **Next**.
6. The next window prompts you to include additional files such as images to the application. The current application has two image files namely, `ipaq.bmp` and `Olite.bmp`. Both files are not system files. Click **Next**.
7. Enter "`Transport`" as the value for all fields in the Application Install Wizard except in the "Company Name" field. Enter "`Oracle`" as the value for the "Company Name" field.
8. Click **Create Install**.

The Application Install Wizard creates CAB files for the selected processors and saves them under the "`C:\Transportinstall\CD1`" directory.

To skip the steps in this section for creating a CAB file, a **cab.zip** file is provided in the following directory.

```
<ORACLE_HOME>\Mobile\Sdk\samples\ado.net\Transport
```

22.2.3.2 Installing the Application from the CAB File

You can download and install the application on the device after packaging and publishing the application. The following sections describe how to package and publish the application.

22.3 Packaging and Publishing the Application

This section describes how to package the application and prepare it for publishing into the Mobile Server. To package and publish the application, you must perform the following tasks.

1. Define the application using the Packaging Wizard.
2. Define the application connection to the Oracle Database Server.
3. Define the snapshot.
4. Publish the application.

22.3.1 Defining the Application Using the Packaging Wizard

Using the Packaging Wizard, you can select and describe the Transport application.

22.3.1.1 Creating a New Application

Using the Mobile Server's Packaging Wizard, you can create or modify a Pocket PC application and publish the Pocket PC application into the Mobile Server. For more information on how to use the Packaging Wizard, see the *Oracle Database Lite Tools and Utilities Guide*.

You can select and describe the Pocket PC Transport application by launching the Packaging Wizard in regular mode.

To launch the Packaging Wizard in regular mode, perform the following steps.

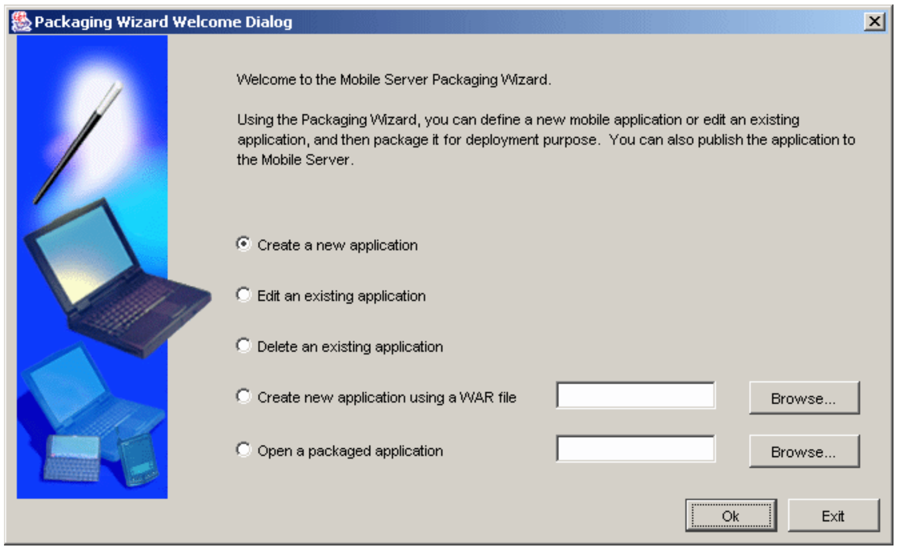
1. Using the Command Prompt, enter the following.

   ```
   cd ORACLE_HOME\mobile\sdk\bin
   ```

   ```
   wtgpack
   ```

   As Figure 22–1 displays, the Packaging Wizard displays the Welcome panel. Select the **Create a new application** option and click **OK**.

Figure 22–1 Welcome Dialog



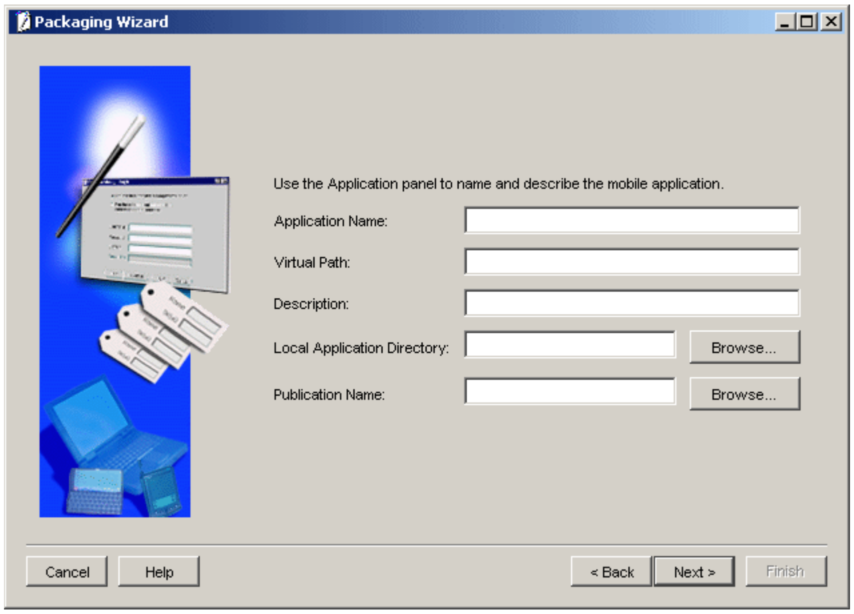
Welcome to the Mobile Server Packaging Wizard.

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 WinCE from the list displayed and click **Next**.
3. The Application panel appears. As Table 22–5 describes, enter the Pocket PC Transport application settings. Figure 22–2 displays the Applications panel.

Figure 22–2 Applications Panel



Use the Application panel to name and describe the mobile application.

Application Name:
Virtual Path:
Description:
Local Application Directory:
Publication Name:

Table 22–5 The Pocket PC Transport Application Settings

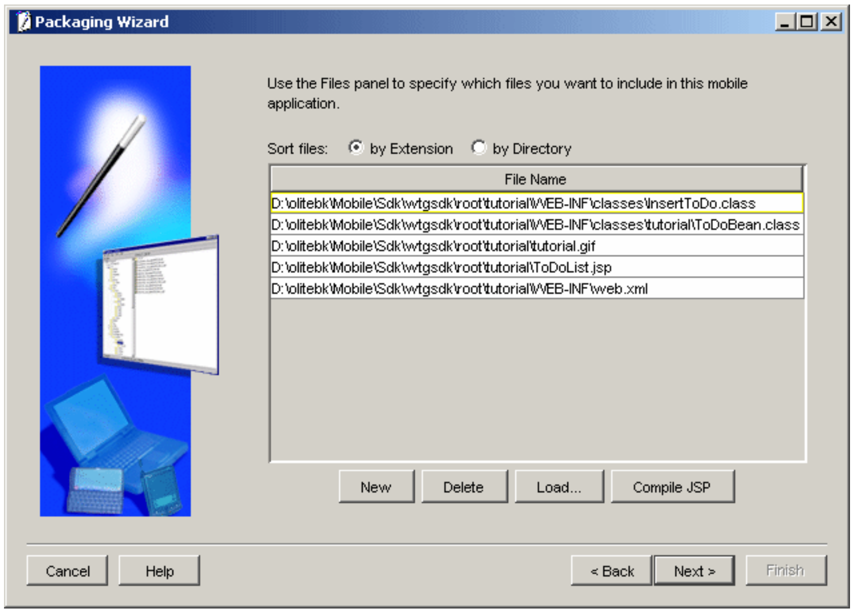
Field	Value
Application Name	Transport
Virtual Path	/Transport

Table 22–5 (Cont.) The Pocket PC Transport Application Settings

Field	Value
Description	Transport and Logistics Management
Local Application Directory	`<ORACLE_HOME>\Mobile\Sdk\samples\ado.net\Transport`
Publication Name	Leave this field blank.

4. Click **Next**. As Figure 22–3 displays, the Files panel appears.

Figure 22–3 Files Panel



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

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 from the directory in which you saved the CAB file, using the Application Install Wizard.

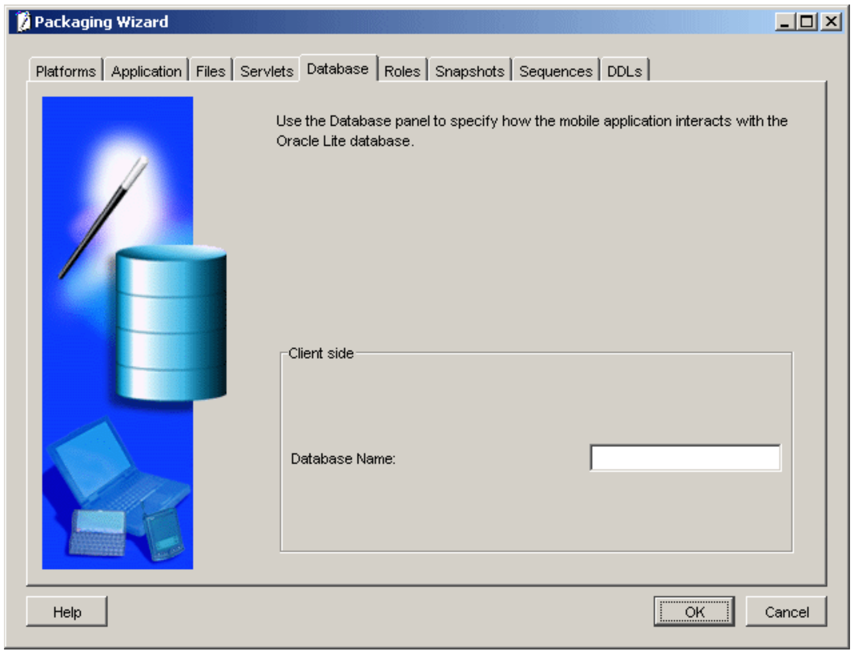
For example, in this tutorial, you must select the `Transport_PPC.ARM.CAB` because your target device is Pocket PC with the ARM chipset.

22.3.2 Defining the Application Connection to the Oracle Database Server

After selecting the appropriate CAB file, you must define the application connection details to the Oracle Database Server.

On the Files panel, click **Next**. As Figure 22–4 displays, the Database panel appears. It enables you to define the Transport application's connection information to the Oracle Database Server.

Figure 22–4 Database Panel



Use the Database panel to specify how the mobile application interacts with the Oracle Lite database.

Client side

Database Name:

The Client Side Database Name field refers to the Data Source Name (DSN) for the Oracle Database Lite database file, which is automatically created on the device. In this filed, enter the value 'transport'.

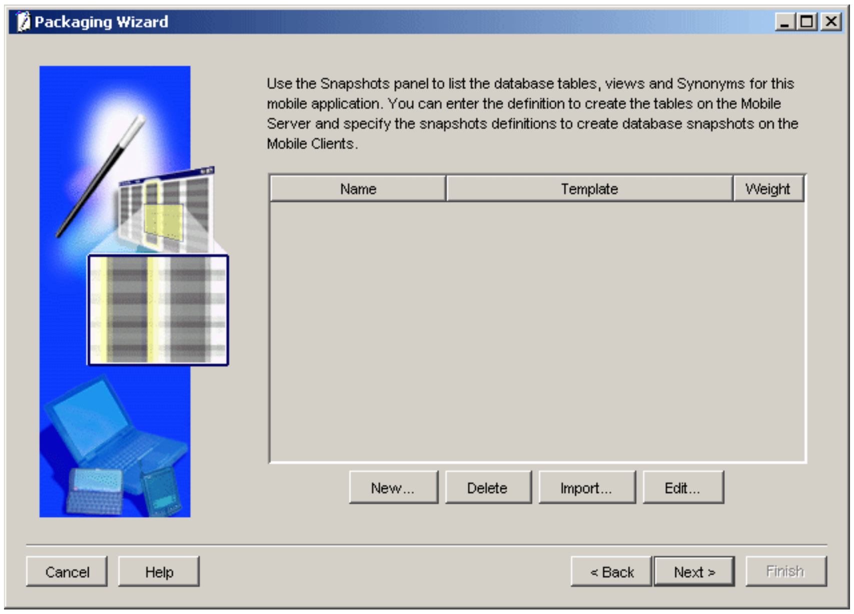
22.3.3 Defining Snapshots

After specifying the application connection details, you must define the snapshots used by your Mobile application. The Snapshots panel defines database tables that contain your Mobile application data and is used for periodic synchronization. It enables you to define the synchronization logic for the Transport application. The Packaging Wizard also enables you to import table definitions from the Oracle Database Server.

To define snapshots for the Transport application, perform the following steps.

1. On the Database panel, click **Next**. As Figure 22–5 displays, the Snapshots panel appears. To import the table definition from the Oracle Database Server, click **Import**. The Connect To Database dialog appears. Enter values as specified in Table 22–6.

Figure 22–5 Snapshots Panel



Use the Snapshots panel to list the database tables, views and Synonyms 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.

Table 22–6 Connect to Database Dialog Description

Field	Description	Value
User Name	Schema name (database user name) which has the database object	master
Password	Password of the schema owner	master
URL	`jdbc:oracle:thin:@<HOST>:<PORT>:<Service_Name>`	`jdbc:oracle:thin:@ssinghan-pc:1521:webtogo`

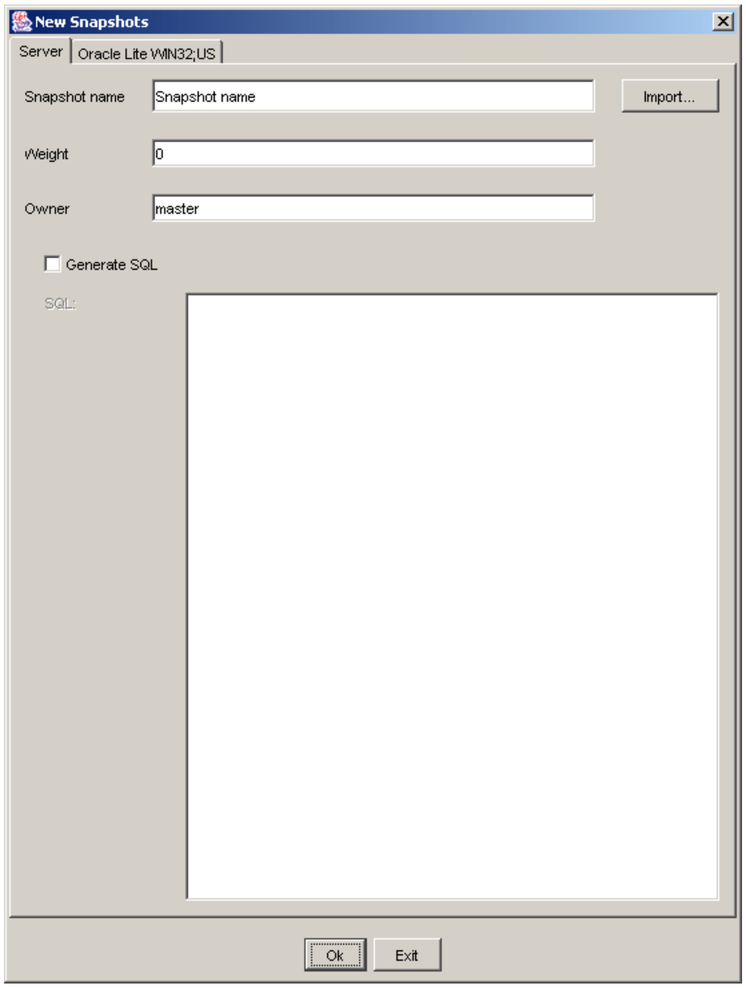
Note: If you do not have the database object on the Oracle Server, you can still create one using the **New** button on the Snapshots panel.

2. Click **OK**. The Tables dialog appears and displays a list of available tables.

   Select the Packages, Trucks, and Routes tables. Click **Add** and click **Close**. The Snapshots panel displays the chosen database tables.

3. Select the Packages table and click **Edit**. As Figure 22–6 displays, the Edit Snapshots panel appears.

Figure 22–6 Edit Snapshots Panel



New Snapshots

Server | Oracle Lite WIN32;US

Snapshot name: Snapshot name

Weight: 0

Owner: master

☐ Generate SQL

SQL:

Note: Ensure that the Create on Client box is selected. If the Create on Client box is not selected, the corresponding snapshot is not created on the Oracle Database Lite client.

---

4. To control the order in which the snapshots are refreshed on the client, you must change the weight for the Packages table to 1. Clear the Generate SQL box, as you have already created database objects in the Oracle Database Server and hence, do not need to create SQL for creating the database.

5. Click the **WinCE** tab. You must ensure that the Create on Client box is selected, and the Template field displays the following SQL statement.

```
SELECT * FROM MASTER.PACKAGES
```

Note: To update the snapshot on a client, you must ensure that the Updatable? box is checked. If the Updatable? box is not checked, the data synchronization will always be unidirectional from the Oracle Database, and all changes made from the device will be lost.

---

6. Click **OK**.
7. In the Snapshots panel, select the Routes table and click **Edit**. The Edit Snapshot dialog appears.
8. To control the order in which snapshots are refreshed on the client, change the weight for the Routes table to 2. Clear the General SQL box, as you have already created database objects in the Oracle Database Server and do not need to create SQL for creating the database.

   Note: As we do not update the Routes and Trucks tables in this tutorial, users must clear the Updatable? box, but ensure that the Create on Client box is selected.

9. Ensure that the Create on Client box is selected, and the Template field displays the following SQL statement.

   ```
   SELECT * FROM MASTER.ROUTES
   ```

10. Repeat steps 8 through 10 for the `TRUCKS` table. Use 3 as the value for weight.
11. Click **Next**. The DDLs panel dialog appears.
12. Click **Finish**. The Application Definition Completed dialog appears.

22.3.4 Publishing the Application

Using the Application Definition Completed dialog, you can package and publish the Pocket PC 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 22–7 describes, enter the specified values.

Table 22–7 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.	`/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.

22.4 Administering 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. Start the Mobile Server.
2. Launch the Mobile Manager.
3. Create a new user.
4. Set application properties.
5. Grant user access to the application.
6. Start MGP.

For more information on the Mobile Manager see the *Oracle Database Lite Administration and Deployment Guide*.

22.4.1 Starting the Mobile Server

To start the Mobile Server in standalone mode, enter the following command using the Command Prompt.

```
> runmobileserver
```

22.4.2 Launching 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.

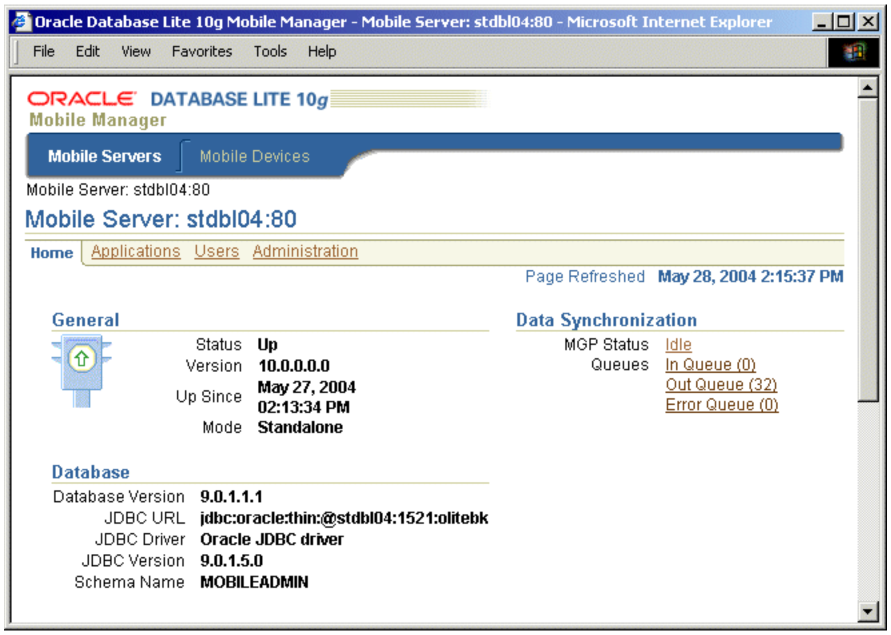
   ```
   http://<mobile_server>/webtogo
   ```

   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 22–7 displays the Mobile Server home page.

Figure 22–7 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

22.4.3 Creating 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 22–8.
4. Click **Save**. The Mobile Manager displays a confirmation message.
5. Click **OK**.

Table 22–8 lists the values that you must enter in the **Add User** page.

Table 22–8 The Add User Page Description

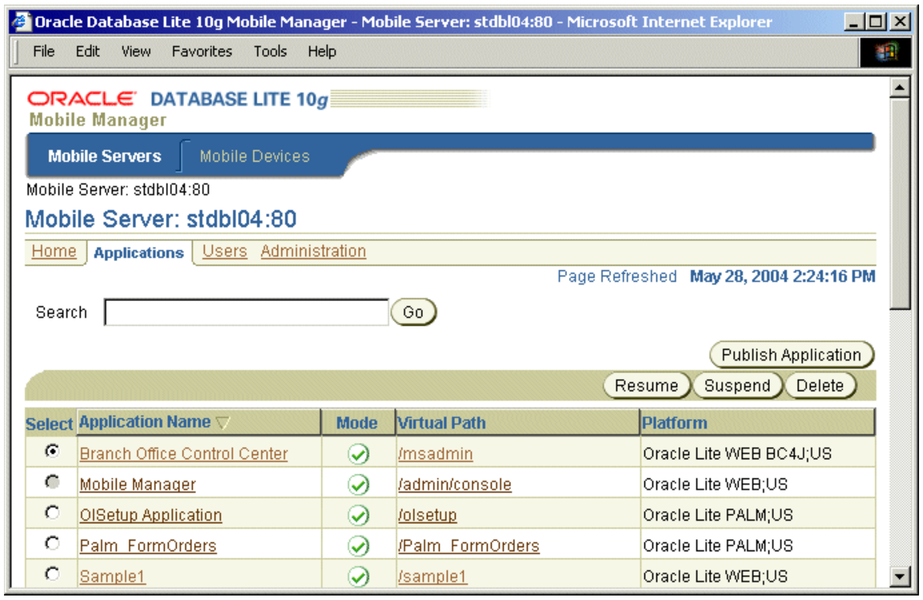
Field	Value
Display Name	bob
User Name	bob
Password	bobhope
Password Confirm	Re-enter the password for confirmation
System Privilege	Select the "User" option

22.4.4 Setting the Application Properties

To set the Pocket PC Transport Application's properties, perform the following steps.

1. In the Mobile Manager, click the **Applications** tab. As Figure 22–8 displays, The **Applications** page appears. You can search the list of available applications by application name.

Figure 22–8 Applications Page



Mobile Server: stdbl04:80

Mobile Server: stdbl04:80

Page Refreshed May 28, 2004 2:24:16 PM

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 PPC2000 ARM; US**. In the **Database Password** field, enter "`master`". This is the default password for the "`master`" user schema of the Oracle Server Database.

4. Click **Save**.

22.4.5 Granting User Access to the Application

To grant user access to the Transport application, perform the following steps.

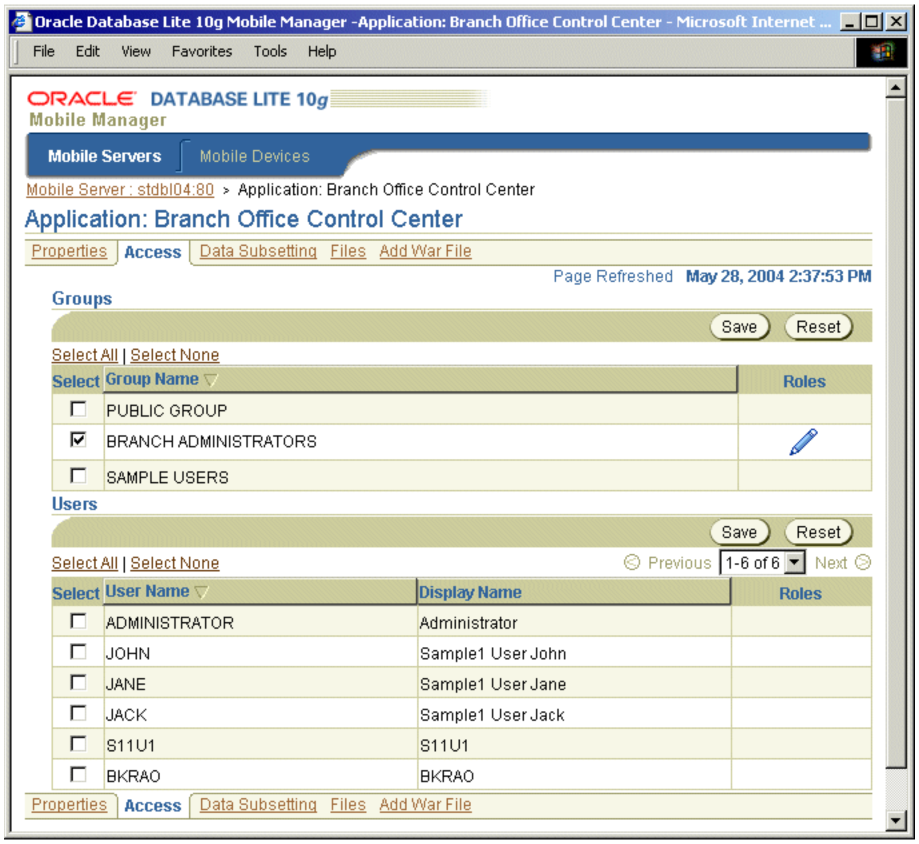
1. In the Transport application page, click the **Access** link. As Figure 22–9 displays, the Access page lists application users and application groups. To grant access to a user or a group of users to the Transport application, select the corresponding boxes.

   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 22–9 displays the Access page of the Transport application.

Figure 22–9 Access Page


Oracle Database Lite 10g Mobile Manager -Application: Branch Office Control Center - Microsoft Internet ...
File Edit View Favorites Tools Help
ORACLE DATABASE LITE 10g
Mobile Manager
Mobile Servers
Mobile Devices
Mobile Server : stdbl04:80 > Application: Branch Office Control Center
Properties Access Data Subsetting Files Add War File
Save
Reset
Select All | Select None
Previous 1-6 of 6 Next

Application: Branch Office Control Center

Page Refreshed May 28, 2004 2:37:53 PM

Groups

Select	Group Name	Roles
☐	PUBLIC GROUP	
☑	BRANCH ADMINISTRATORS	✎
☐	SAMPLE USERS	

Users

Select	User Name	Display Name	Roles
☐	ADMINISTRATOR	Administrator	
☐	JOHN	Sample1 User John	
☐	JANE	Sample1 User Jane	
☐	JACK	Sample1 User Jack	
☐	S11U1	S11U1	
☐	BKRAO	BKRAO	

22.4.6 Starting the Message Generator and Processor (MGP)

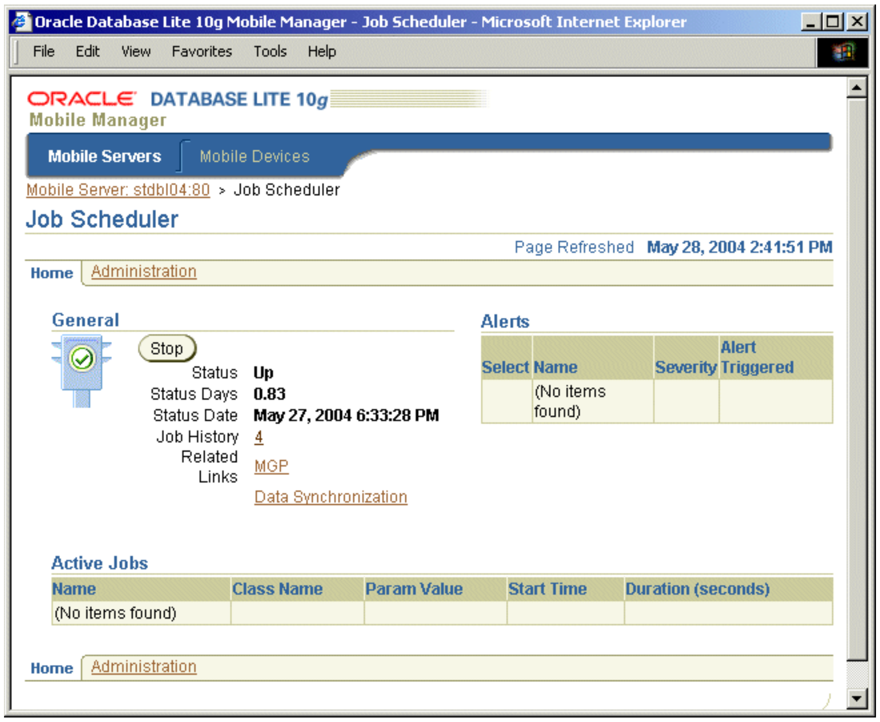
In the Oracle Database Lite 10*g* Asynchronous replication model, a client does not wait for the server to prepare the payload. A payload contains data that will be synchronized. The Mobile Server prepares the payload for all Mobile clients asynchronously by running the MGP process in the background at all times. Hence, when a Mobile Client initiates the synchronization process, the Mobile Server uploads the client payload into an in-queue and picks up the payload for the client from the corresponding out-queue. The MGP processes payloads in the in-queues and out-queues and performs database operations with the Oracle Server in the background.

To start the MGP, perform the following steps.

1. Navigate to the Mobile Manager Home page and click **Jobs** in the **Components** list. The **Job Scheduler** page appears.
2. Click the **Start** button.

   Figure 22–10 displays the **Job Scheduler** page.

Figure 22–10 Job Scheduler Page



Mobile Server: stdbl04:80 > Job Scheduler

Job Scheduler

Page Refreshed May 28, 2004 2:41:51 PM

General

Status	Up
Status Days	0.83
Status Date	May 27, 2004 6:33:28 PM
Job History	4

Alerts

Select	Name	Severity	Alert Triggered
	(No items found)		

Active Jobs

Name	Class Name	Param Value	Start Time	Duration (seconds)
(No items found)				

22.5 Running the Application on the Pocket PC

This section describes how to run the application after creating, testing, deploying, and administering the application. To run the application, perform the following tasks.

1. Install the Oracle Database Lite Mobile Client for Pocket PC.
2. Install and synchronize the Transport application.

22.5.1 Installing 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
   ```

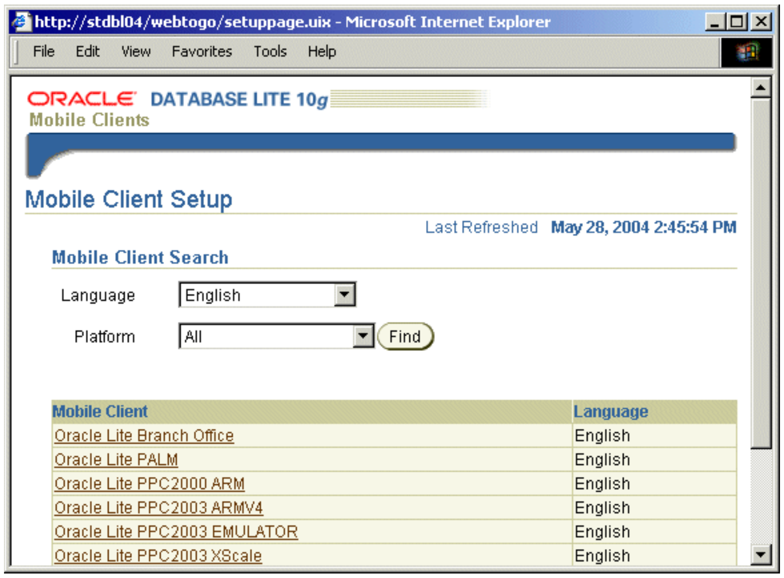
   Note: You must replace the `<Mobile_Server>` variable with the host name or IP address of your Mobile Server.

   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 PPC2000 ARM** to access the setup program for the Pocket PC device with the ARM chipset.

   Figure 22–11 displays the Mobile Client Setup page.

Figure 22–11 Mobile Client Setup Page



http://stdbl04/webtogo/setuppage.uix - Microsoft Internet Explorer

File Edit View Favorites Tools Help

ORACLE DATABASE LITE 10g
Mobile Clients

Mobile Client Setup

Last Refreshed May 28, 2004 2:45:54 PM

Mobile Client Search

Language: English
Platform: All
Find

Mobile Client	Language
Oracle Lite Branch Office	English
Oracle Lite PALM	English
Oracle Lite PPC2000 ARM	English
Oracle Lite PPC2003 ARMV4	English
Oracle Lite PPC2003 EMULATOR	English
Oracle Lite PPC2003 XScale	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. Use the default directory `C:\mobileclient\olite`, and 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's 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's installation starts on the device. Once completed, the **Mobile Client for Pocket PC** is installed on your device under the **\ORACE** directory.

22.5.2 Installing and Synchronizing 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.

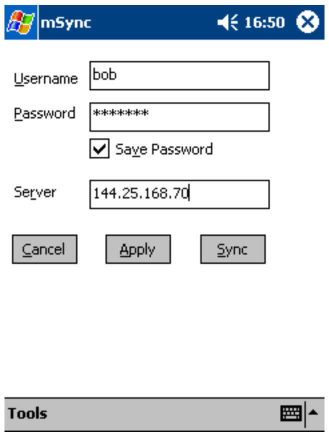
2. The msync dialog appears. To download the Transport application and snapshots for user BOB, enter data as described in Table 22–9.

Table 22–9 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 22–12 displays the **msync** dialog on the Pocket PC.

Figure 22–12 Running msync on Pocket PC



Username: bob
Password: *******
[✓] Save Password
Server: 144.25.168.70

3. To save these values, tap **Apply**.
4. To synchronize your application and data to the device, tap the **Sync** button.

> **Note:** Ensure that the device is connected to the desktop or the network and that the Mobile Server is running.

5. After the synchronization process is complete, a `transport.odb` file is created under the **\OraCE** directory and the **Transport** application is installed on the Pocket PC automatically.
6. Using the **Start** menu on the device, locate the **Transport** application in the **Programs** menu.
7. To run the **Transport** application, tap the **Transport** icon.

Part III

Appendices

Part III contains the following appendices:

A Optimizing SQL Queries

This document provides tips on improving the performance of your SQL queries. Topics include:

- Section A.1, "Optimizing Single-Table Queries"
- Section A.2, "Optimizing Join Queries"
- Section A.3, "Optimizing with Order By and Group By Clauses"

The tip examples use the database schema listed in Table A–1:

Table A–1 Database Schema Examples

Tables	Columns	Primary Keys	Foreign Keys
LOCATION	LOC#	LOC#	
	LOC_NAME		
EMP	SS#	SS#	
	NAME		
	JOB_TITLE		
	WORKS_IN		WORKS_IN references DEPT (DEPT#)
DEPT	DEPT#	DEPT#	
	NAME		
	BUDGET		
	LOC		LOC references LOCATION (LOC#)
	MGR		MGR references EMP (SS#)

A.1 Optimizing Single-Table Queries

To improve the performance of a query that selects rows of a table based on a specific column value, create an index on that column. For example, the following query performs better if the `NAME` column of the `EMP` table has an index.

```
SELECT *
FROM EMP
WHERE NAME = 'Smith';
```

An index may ruin performance if selecting more than 10% of the rows of the indexing columns is poor. For example, an index on `JOB_TITLE` may not be a good choice even if the query is as follows.

```
SELECT *
FROM EMP
```

```
WHERE JOB_TITLE='CLERK'
```

A.2 Optimizing Join Queries

The following can improve the performance of a join query (a query with more than one table reference in the FROM clause).

A.2.1 Create an Index on the Join Column(s) of the Inner Table

In the following example, the inner table of the join query is DEPT and the join column of DEPT is `DEPT#`. An index on `DEPT.DEPT#` improves the performance of the query. In this example, since `DEPT#` is the primary key of DEPT, an index is implicitly created for it. The optimizer will detect the presence of the index and decide to use DEPT as the inner table. In case there is also an index on EMP.WORKS_IN column the optimizer evaluates the cost of both orders of execution; DEPT followed by EMP (where EMP is the inner table) and EMP followed by DEPT (where DEPT is the inner table) and picks the least expensive execution plan.

```
SELECT e.SS#, e.NAME, d.BUDGET
FROM EMP e, DEPT d
WHERE e.WORKS_IN = DEPT.DEPT#
AND e.JOB_TITLE = 'Manager';
```

A.2.2 Bypassing the Query Optimizer

Normally optimizer picks the best execution plan, an optimal order of tables to be joined. In case the optimizer is not producing a good execution plan you can control the order of execution using the HINTS feature SQL. For more information see the *Oracle Database Lite SQL Reference*.

For example, if you want to select the name of each department along with the name of its manager, you can write the query in one of two ways. In the first example which follows, the hint `/*+ordered*/` says to do the join in the order the tables appear in the `FROM` clause.

```
SELECT /*+ordered*/ d.NAME, e.NAME
FROM DEPT d, EMP e
WHERE d.MGR = e.SS#
```

or:

```
SELECT /*+ordered*/ d.NAME, e.NAME
FROM EMP e, DEPT d
WHERE d.MGR = e.SS#
```

Suppose that there are 10 departments and 1000 employees, and that the inner table in each query has an index on the join column. In the first query, the first table produces 10 qualifying rows (in this case, the whole table). In the second query, the first table produces 1000 qualifying rows. The first query will access the EMP table 10 times and scan the DEPT table once. The second query will scan the EMP table once but will access the DEPT table 1000 times. Therefore the first query will perform much better. As a rule of thumb, tables should be arranged from smallest effective number of rows to largest effective number of rows. The effective row size of a table in a query is obtained by applying the logical conditions that are resolved entirely on that table.

In another example, consider a query to retrieve the social security numbers and names of employees in a given location, such as New York. According to the sample schema, the query would have three table references in the FROM clause. The three tables could be ordered in six different ways. Although the result is the same regardless of which order you choose, the performance could be quite different.

Suppose the effective row size of the LOCATION table is small, for example `select count(*) from LOCATION where LOC_NAME = 'New York'` is a small set. Based on the above rules, the LOCATION table should be the first table in the FROM clause. There should be an index on LOCATION.LOC_NAME. Since LOCATION must be joined with DEPT, DEPT should be the second table and there should be an index on the LOC column of DEPT. Similarly, the third table should be EMP and there should be an index on `EMP#`. You could write this query as:

```
SELECT /*+ordered*/ e.SS#, e.NAME
FROM LOCATION l, DEPT d, EMP e
WHERE l.LOC_NAME = 'New York' AND
l.LOC# = d.LOC AND
d.DEPT# = e.WORKS_IN;
```

A.3 Optimizing with Order By and Group By Clauses

Various performance improvements have been made so that SELECT statements run faster and consume less memory cache. Group by and Order by clauses attempt to avoid sorting if a suitable index is available.

A.3.1 IN Subquery Conversion

Converts IN subquery to a join when the select list in the subquery is uniquely indexed.

For example, the following IN subquery statement is converted to its corresponding join statement. This assumes that c1 is the primary key of table t2:

```
SELECT c2 FROM t1 WHERE
c2 IN (SELECT c1 FROM t2);
```

becomes:

```
SELECT c2 FROM t1, t2 WHERE t1.c2 = t2.c1;
```

A.3.2 ORDER BY Optimization with No GROUP BY

This eliminates the sorting step for an ORDER BY clause in a select statement if ALL of the following conditions are met:

1. All ORDER BY columns are in ascending order or in descending order.
2. Only columns appear in the ORDER BY clause. That is, no expressions are used in the ORDER BY clause.
3. ORDER BY columns are a prefix of some base table index.
4. The estimated cost of accessing by the index is less than the estimated cost of sorting the result set.

A.3.3 GROUP BY Optimization with No ORDER BY

This eliminates the sorting step for the grouping operation if GROUP BY columns are the prefix of some base table index.

A.3.4 ORDER BY Optimization with GROUP BY

When ORDER BY columns are the prefix of GROUP BY columns, and all columns are sorted in either ascending or in descending order, the sorting step for the query result is eliminated. If GROUP BY columns are the prefix of a base table index, the sorting step in the grouping operation is also eliminated.

A.3.5 Cache Subquery Results

If the optimizer determines that the number of rows returned by a subquery is small and the query is non-correlated, then the query result will be cached in memory for better performance. For example:

```
select * from t1 where
t1.c1 = (select sum(salary)
from t2 where t2.deptno = 100);
```

B Oracle Lite Database Utilities

This appendix describes how to use the following Oracle Lite database utilities for the Windows 32 and Windows CE platforms. Table B–1 lists all of the utility names:.

Table B–1 Database Tools and Utilities

Utility	Description
Section B.1, "The mSQL Tool"	Allows users to execute SQL statements against the Oracle Lite database.
Section B.2, "CREATEDB"	Use this to create your Oracle Lite database.
Section B.3, "REMOVEDB"	Use this to remove your Oracle Lite database.
Section B.4, "ENCRYPDB"	Use this to encrypt your Oracle Lite database.
Section B.5, "DECRYPDB"	Use this to decrypt your Oracle Lite database.
Section B.6, "MIGRATE"	Use this to migrate to Oracle Database Lite from a previous release.
Section B.7, "ODBC Administrator and the Oracle Database Lite ODBC Driver"	Use this to manage ODBC connections by creating data source names (DSNs) that associate the Oracle Database Lite ODBC Driver with the Oracle Database Lite that you want to access through the driver.
Section B.8, "ODBINFO"	Use this utility to find out the version number and volume ID of an Oracle Database Lite database.
Section B.9, "VALIDATEDB"	Use this to validate the structure of an Oracle Lite database and find any corruption of the database.
Section B.10, "Transferring Data Between a Database and an External File"	Use either the command-line tool or programmatic APIs to load data from an external file into a table in Oracle Database Lite, or to unload (dump) data from a table in Oracle Database Lite to an external file.
Support for Linguistic Sort	Allows databases to be created with linguistic sort capability enabled. See Section 2.11, "Support for Linguistic Sort" for more information.
dropjava	This is a command-line utility you can use to remove Java classes from Oracle Database Lite. For more information, see Section 11.3.1.4.1, "Using dropjava".
loadjava	This is a command-line utility you can use to load a Java class into Oracle Database Lite. For more information, see Section 11.3.1.1.1, "loadjava".

B.1 The mSQL Tool

Mobile SQL (mSQL) is a GUI-based application that runs on the client device (laptop, Palm OS, and Windows CE). It allows the user to execute SQL statements against the local database. It is both a developers tool and a code example. It allows users to access functionality provided by the ODBC and Oracle Database Lite OKAPI interfaces of the underlying Oracle Database Lite database engine.

The mSQL tool enables you to create, access, and manipulate Oracle Database Lite on Palm Computing platform devices. Using mSQL you can accomplish the following:

- Create databases
- View tables
- Execute SQL statements

The following sections describe how to use the mSQL tool on two platforms:

- Section B.1.1, "The mSQL Tool for Windows 32"
- Section B.1.2, "The mSQL Tool for Windows CE"

B.1.1 The mSQL Tool for Windows 32

The mSQL tool is an application that runs as a command-line interface. It allows the user to execute SQL statements against the local database. The mSQL tool accesses the database through JDBC. It is both a developers tool and a code example.

The following sections describe information relevant to database access, starting mSQL and mSQL commands:

- Section B.1.1.1, "Starting mSQL"
- Section B.1.1.2, "Populating your Database Using mSQL"
- Section B.1.1.3, "SET TERM {ON | OFF}"
- Section B.1.1.4, "SET TIMING {ON | OFF}"
- Section B.1.1.5, "SET VERIFY {ON | OFF}"

B.1.1.1 Starting mSQL

Start mSQL by opening the `*ORACLE_HOME*\Mobile\SDK\Bin` directory and double-clicking on the `msql.exe` file. This starts the command-line interface that accepts standard SQL commands. For more information, see the *Oracle Database Lite SQL Reference*.

B.1.1.2 Populating your Database Using mSQL

You can use SQL scripts to create tables and schema, and to insert data into tables. A SQL script is a text file, generally with a `.sql` extension, that contains SQL commands. You can run a SQL script from the mSQL prompt, as follows:

```
msql> @<ORACLE_HOME>\DBS\Poldemo.sql
```

You can also type:

```
msql> START <filename>
```

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

B.1.1.3 SET TERM {ON|OFF}

Controls the display of output generated by commands executed from a script. OFF suppresses the display so that you can spool output from a script without seeing the output on the screen. ON displays the output. TERM OFF does not affect output from commands you enter interactively.

B.1.1.4 SET TIMING {ON|OFF}

Controls the display of timing statistics. ON displays timing statistics on each SQL command. OFF suppresses timing of each command.

B.1.1.5 SET VERIFY {ON|OFF}

Controls whether to list the text of a SQL statement or PL/SQL command before and after replacing substitution variables with values. ON lists the text; OFF suppresses the listing.

B.1.2 The mSQL Tool for Windows CE

The following sections describe start mSQL and how to access the database using mSQL for the Windows CE platform:

- Section B.1.2.1, "Database Access"
- Section B.1.2.2, "Starting mSQL"
- Section B.1.2.3, "Manage Snapshots Using mSQL"

B.1.2.1 Database Access

The mSQL tool accesses the database through both the ODBC and OKAPI interface. Most functions are performed through ODBC, but functions that ODBC cannot handle are implemented using OKAPI function calls.

B.1.2.2 Starting mSQL

Start mSQL by opening the `<`*`ORACLE_HOME`*`>\Mobile\Sdk\WinCE`, selecting the folder representing the version Windows CE, and selecting the processor on your device. Double-click on the `msql.exe` file. This starts the GUI which accepts standard SQL commands. For more information, see the *Oracle Database Lite SQL Reference*.

B.1.2.3 Manage Snapshots Using mSQL

The Oracle Lite database format is the same on Windows 32 and Windows CE platforms. Manage your snapshots, as follows:

1. Create and test your snapshots on Windows 32 using the Windows 32 mSQL command-line utility.
2. Copy the database to the Windows CE platform.
3. Use the Windows CE mSQL tool to manipulate the database that is on your device.

The mSQL tool enables the user to execute SQL statements against the local database and access functionality provided by the interfaces of the underlying Oracle Lite database engine.

B.2 CREATEDB

Description

Utility for creating a database.

Syntax

```
CREATEDB DataSourceName DatabaseName Database_SysUser_Password [[[VolID] DATABASE_
SIZE] EXTENT_SIZE] [collation sequence]
```

Keywords and Parameters

`DataSourceName`

Data source name, used to look up the `ODBC.TXT` file for the default database directory.

Note: If you specify an invalid DSN, Oracle Database Lite ignores the DSN and creates the database in the current directory. To access this database through ODBC, you must create a DSN for the database that points to the directory in which the database resides. For instructions on adding a DSN, see Section B.7.1, "Adding a DSN Using the ODBC Administrator".

`DatabaseName`

Name of the database to be created. It can be a full path name or just the database name. If only the database name is given, the database is created under the Data Directory for the data source name specified in the `ODBC.TXT` file. The extension for the database name must always be `.ODB`. If a name without the `.ODB` is given, the `.ODB` is appended.

`DATABASE_SysUser_Password`

The database system user password.

`VolID`

When specified, the `VolID` is used as the database ID, instead of the database ID from the `POLITE.INI` file. The ID must be unique for each database. If you specify a volumn id, then you also specify the database and extent sizes. Thus, the createdb executable knows that the volume id, database size and extent size are being specified when three numbers are provided in a row.

Note: For the volume id, database size, and extent size, specify only the number; do not specify name=value. See the examples for more information.

`DATABASE_SIZE`

The database size in bytes. If you want to specify the database size, then you also must specify the volume id and extent size.

`EXTENT_SIZE`

An incremental amount of pages in a database file. When a database runs out of pages in the current file, it extends the file by this number of pages. If you want to specify the extent size, then you also must specify the volume id and database size.

`COLLATION_SEQUENCE`

This parameter is a string constant which creates the database as enabled for linguistic sorting when a value other than the default is used. A collation sequence specified here overrides a collation sequence set using the `NLS_SORT [collation_sequence]` parameter in the `POLITE.INI` file. The string can also be one of the options listed in Table B–2:

Table B–2 Collation Sequence Values

Collation Sequence	Description
`BINARY`	Default. Two strings are compared character by character and the characters are compared using their binary code value.
`FRENCH`	Two strings are compared according to the collation sequence of French. Supported by ISO 8859-1 or IBM-1252.
`GERMAN`	Two strings are compared according to the collation sequence of German. Supported by ISO 8859-1 or IBM-1252.
`CZECH`	Two strings are compared according to the collation sequence of Czech. Supported by ISO 8859-2 or IBM-1250.
`XCZECH`	Two strings are compared according to the collation sequence of Xczech. Supported by ISO 8859-2 or IBM-1250.

Note: There is no way to alter a collation sequence after the database is created.

Examples

Create the `db1` database with DSN of `polite` and password `manager`: `createdb polite db1 manager`

Create the `db2.odb` database with DSN `polite` and password `manager300`: `createdb polite c:\testdir\db2.odb manager300`

Create `polite` database with DSN polite, password of `manager`, and a collation sequence of `french`: `createdb polite polite manager french`

Create polite database with DSN `polite`, password `manager`, volume id of 199, database size of 1000, and extent size of 1:
`createdb polite polite manager 199 1000 1`

B.3 REMOVEDB

Description

Utility for deleting a database.

Syntax

REMOVEDB DataSourceName Database Name

Keywords and Parameters

DataSourceName
Data source name of the database you want to remove. The DSN can be a dummy argument such as none, in which case the database name must be a fully qualified filename.

DatabaseName
The name of the database to delete. It can be a full path name or just the database name. If only the database name is given, the database is deleted from the Data Directory for the data source name specified in the `ODBC.INI` file.

Examples

```
removedb polite db1
```

```
removedb none c:\testdir\db2.odb
```

B.4 ENCRYPDB

Description
Enables you to encrypt Oracle Database Lite with a password, which prevents unauthorized access to the database and encrypts the database, so that the data stored in the database files cannot be interpreted. To decrypt the database, see Section B.5, "DECRYPDB".

This tool is used by embedded applications to encrypt the database used by the application. To encrypt an Oracle Lite database used by a Mobile client, see the `ENCRYPT_DB` parameter in the `POLITE.INI` Appendix in the *Oracle Database Lite Administration and Deployment Guide*.

`ENCRYPDB` uses AES-128 encryption.

Syntax
`ENCRYPDB` *DSN* | `NONE` *DBName* [*New_Password* [*Old_Password*]]

Keywords and Parameters

- `DSN`—Data Source Name of Oracle Database Lite that you want to encrypt. If you specify `NONE`, `DBName` must be a fully qualified database name with the full path name (without the `.ODB` extension). If the `DSN` is a value other than `NONE`, then the name must appear as a data source name in the `ODBC.TXT` file.
- `DBName`—Name of the database to be encrypted. If DSN was specified as `NONE`, `DBName` must be entered with the full path name.
- `New_Password and Old_Password`—Optional, the password (or previously used password) for encrypting the database. This password can be 128 characters in length. If you do not enter a password, `ENCRYPDB` prompts you to enter one. Since both passwords are optional in the command line to invoke the utility, the command line could have three different forms:
    - No password given: If the database is already encrypted, then `ENCRYPDB` assumes that the user is trying to change the password of the database. It prompts the user for the old password once and new password twice, and encrypts the database using the new password. If the database is not already

encrypted, ENCRYPDB prompts for the new password twice and encrypts the database using this new password.

- One password given: This password is assumed to be the new password. If the database is already encrypted, ENCRYPDB prompts for the old password and encrypts the database using the new password.
- Both passwords given: ENCRYPDB assumes that the first password is the new password and the second is the old password.

Comments

If you call this utility from another program, the possible values returned are listed in Table B–3:

Table B–3 ENCRYPDB Return Codes

Return Code	Description
`EXIT_SUCCESS`	Success
`EXIT_USAGE`	Command line arguments are not properly used or are in error
`EXIT_PATH_TOO_LONG`	Path is too long
`EXIT_SYSCALL`	I/O error while making new encrypted copy on disk
`EXIT_BAD_PASSWD`	Incorrect password supplied

The default Oracle Database Lite (`POLITE.ODB`) is not encrypted. After encrypting an Oracle Database Lite, every user that attempts to establish a connection to the encrypted Oracle Database Lite must provide the valid password. If the password is not provided, Oracle Database Lite returns an error. An Oracle Database Lite database cannot be encrypted if there are any open connections to the database.

You should consider the following when encrypting and decrypting Oracle Database Lite:

- You cannot decrypt an encrypted database without the password. Make sure you back up your database in a secure place before you encrypt it. Another user of the same database can create a copy with a new user name for a user who loses their password, otherwise, there is no method to recover a database where the passwords are lost.
- After encrypting the database, you must include the password in the connect string to connect to the database.
- A password encrypts the entire database. It is not a user-specific password.
- Database encryption does not prevent a third party from removing an Oracle Lite Database. That is, `removedb` and `rmdb` remove a database without checking the password. Use tools that protect unauthorized users from manipulating your file system.
- ODBC applications that connect to an encrypted Oracle Database Lite database need to specify a valid password. It is customary to prompt for the password at runtime rather than to code it in the application. Most ODBC applications can use the `SQLDriverConnect` function with the `DRIVER=` option, rather than the `SQLConnect` function, if the applications require the Oracle Database Lite ODBC driver to prompt for the password at runtime.
- All sample applications provided with this release of Oracle Database Lite are designed to run against a database that is not encrypted.

- You can use DECRYPDB and ENCRYPDB (in this order) to change the password of a database. However, DECRYPDB creates an Oracle Database Lite database in plain text before ENCRYPDB encrypts it. This results in a database in plain text form, for a short period of time, and is not recommended.
- For encrypted databases, all user names and passwords are written to a file named DSN.OPW. Each user can then use the password as a "key" to unlock the .OPW file before the .ODB file is accessed. When you copy or back up the database, you should include the .OPW file.

B.5 DECRYPDB

Description

This tool allows you to decrypt an encrypted Oracle Lite database used with an embedded application. For more information, see Section B.4, "ENCRYPDB".

This tool is used by embedded applications to decrypt the database used by the application. To encrypt an Oracle Lite database used by a Mobile client, see the ENCRYPT_DB parameter in the POLITE.INI Appendix in the *Oracle Database Lite Administration and Deployment Guide*.

SYNTAX

DECRYPDB *DSN* | NONE *DBName* [*Password*]

Keywords and Parameters

`DSN`

Data Source Name of Oracle Database Lite that you want to decrypt. If you specify NONE, you must the enter the DBName with the full path name (without the .ODB extension).

`DBName`

Name of the database to be decrypted. If DSN was specified as NONE, the DBName must be entered with the full path name.

`Password`

Optional. The password used previously to encrypt Oracle Database Lite. If you do not enter the password, DECRYPDB prompts you to enter it.

Comments

An Oracle Database Lite database cannot be decrypted if there is any open connection to the database.

If you call this utility from another program, the possible values returned are listed in Table B–4:

Table B–4 DECRYPDB Return Codes

Return Code	Description
EXIT_SUCCESS	Success
EXIT_USAGE	Command line arguments are not properly used or are in error
EXIT_PATH_TOO_LONG	Path is too long
EXIT_SYSCALL	I/O error while making new decrypted copy on disk

Table B–4 (Cont.) DECRYPDB Return Codes

Return Code	Description
`EXIT_BAD_PASSWD`	Incorrect password supplied

For more information, see the comments in Section B.4, "ENCRYPDB".

B.6 MIGRATE

Description

Utility for migrating a database from a previous version of Oracle Lite to Oracle Database Lite 10*g*. Before you use this utility, you must install the current release of Oracle Database Lite. Also, if your database is encrypted, you must first decrypt it before using this utility.

Syntax

MIGRATE DSN DBName

where `DBName` can be the database name or the database path and name.

Keywords and Parameters

`DSN`

Data source name of the database to migrate. This is used to look up the default database directory in the `ODBC.INI` file for the database name given in `DBName`. If the `DSN` has the value `NONE` the `DBName` should be a complete pathname of the database file.

`DBName`

The database name, or the path and database name, to migrate. If only the database name is specified, the database file must exist in the directory specified in the `DataDirectory` parameter (under the data source name) in the `ODBC.INI` file.

Comments

As mentioned in this section, you must install Oracle Database Lite before you use this utility.

Any messages generated by the `MIGRATE` utility are displayed on the screen in the command window.

Using this utility allows you to compress empty space in your existing Oracle Database Lite database.

This utility does not support the migration of Java Stored Procedures.

Examples

```
MIGRATE polite db1
```

```
MIGRATE none c:\testdir\db1.odb
```

B.7 ODBC Administrator and the Oracle Database Lite ODBC Driver

A Data Source Name (DSN) associates the Oracle Database Lite ODBC Driver with the Oracle Database Lite database that you want to access through the driver. The Oracle

Database Lite installation process creates a default DSN, POLITE, for the Oracle Database Lite database. You can also create additional DSNs for the additional Oracle Database Lite databases that you create.

Microsoft provides the ODBC Administrator, a tool for managing the ODBC.INI file and associated registry entries in Windows 98/NT/2000/XP. The ODBC.INI file and the Windows registry store the DSN entries captured through the ODBC Administrator. Using the ODBC Administrator, you can relate a DSN to the Oracle Database Lite ODBC Driver.

Note: This document does not provide instructions on using the ODBC Administrator. See the ODBC Administrator tool online help for this information.

In the ODBC Administrator, in addition to the DSN, you must specify the parameters listed in Table B–5:

Table B–5 ODBC Administrator DSN Parameters

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

Table B–5 (Cont.) ODBC Administrator DSN Parameters

DSN Parameter	Description
`Default Cursor Type`	▪ *Forward Only*: Default. A non-scrollable cursor which only moves forward but not backward through the result set. As a result, the cursor cannot go back to previously fetched rows. ▪ *Dynamic*: Capable of detecting changes to the membership, order, or values of a result set after the cursor is opened. If a dynamic cursor fetches rows that are subsequently deleted or updated by another application, it detects those changes when it fetches those rows again. ▪ *Keyset Driven*: Does not detect change to the membership or order of a result set, but detects changes to the values of rows in the result set. ▪ *Static*: Does not detect changes to the membership, order or values of a result set after the cursor is opened. If a static cursor fetches a row that is subsequently updated by another application, it does not detect the changes even if it fetches the row again.

For example, the DSN entry for `POLITE` in the `ODBC.INI` file may contain:

```
[POLITE]
Description=Oracle Lite Data Source
DataDirectory=C:\ORANT\OLDB40
Database=POLITE
IsolationLevel=Repeatable Read
CursorType=Dynamic
```

B.7.1 Adding a DSN Using the ODBC Administrator

To add a DSN using the ODBC Administrator:

1. Start the ODBC Administrator, either by selecting its icon in the Oracle Database Lite program group, or by typing the following at a DOS prompt:

   ```
   C:\>ODBCAD32
   ```

2. Click **Add**.
3. Double-click the **Oracle Database Lite *nn* ODBC Driver**, where *nn* is the release number, from the list of Installed ODBC Drivers.
4. Next, add the DSN name and define the parameters in the ODBC driver setup dialog. Refer the preceding table for help in defining the parameters.

B.7.2 Adding a DSN which points to Read-Only Media (CD-ROM)

1. Create the DSN as explained in Section B.7.1, "Adding a DSN Using the ODBC Administrator".
2. Add the following line to the new DSN in the `ODBC.INI` file:

   ```
   ReadOnly = True
   ```

Note: You can define a DSN which points to a file on a CD-ROM. Simply point the DSN to the CD-ROM drive and directory and provide the file name of the database file. Then modify the `ODBC.INI` file to add the line `ReadOnly=True` to the data source definition. ODBC programmers can call the following before opening the database to enable this feature (instead of adding the line to the `ODBC.INI` file):

```
SQLSetConnectOption( hdbc, SQL_ACCESS_MODE, SQL_
MODE_READ_ONLY )
```

Setting a database file to read-only suppresses the creation of log files. Updates, insertions, deletions, or commits appear to work on the in-memory image of tables. However, when you commit, these changes are not written to the database file. If you exit your application, reconnect, and issue your query, you see your original data.

B.8 ODBINFO

Description

You can use `ODBINFO` to find out the version number and volume ID of an Oracle Database Lite database. `ODBINFO` can also display and set several parameters.

Syntax

To display current information without making any changes use the syntax:

```
odbinfo [-p passwd]DSN DBName
```

You can also use:

```
odbinfo [-p passwd] NONE dbpath\dbanme.odb
```

For example:

```
odbinfo -p tiger polite polite
```

```
odbinfo NONE c:\orant\oldb40\polite.odb
```

If your database is encrypted you need to include the password.

Parameters

To set or clear parameters, use one or more "+" or "-" parameter arguments before the DSN or NONE. For example:

```
odbinfo +reuseoid -pagelog -fsync polite polite
```

You can use the parameters listed in Table B–6 with the `ODBINFO` utility:

Table B–6 ODBINFO Parameters

Parameter	Description
`pagelog`	By default, a commit backs up modified database pages to `filename.plg` before actually writing the changes to `filename.odb`. If an application or the operating system experiences a failure during a commit, the transaction is cleanly rolled back during the next connect. If `-pagelog` is specified, no backup is created and the database can become corrupted if a failure occurs.
`fsync`	Oracle Database Lite generally forces the operating system to write all the modified buffers associated with the database back to disk during a commit. If this option is disabled (`-fsync`), the operating system can keep the changes in memory until a later time. If the system (but not the application) crashes before the buffers are flushed, the database can become corrupted. Using `odbinfo -fsync -pagelog` improves the performance of applications that use many small transactions (with autocommit on) or ones with massive updates. However, if the database is corrupted, there is no straightforward way to repair it or recover the data. Therefore these two options should only be cleared during initial loading of the database, if (1) the `.ODB` file is backed up on regular basis, or (2) the data in the database can be recovered from some other source. Using this option has no effect on applications that seldom update the database. Setting the transaction isolation level to `SINGLE USER` has more impact in this case.
`reuseoid`	By default, Oracle Database Lite does not reuse the `ROWID` of any row that exists in a table until the table is dropped. The "Slot Deleted" error is returned when accessing a deleted object. This uses two bytes of storage for each deleted object, causing performance and disk space usage to degrade over time if rows are constantly inserted and deleted. If you use `odbinfo +reuseoid`, new rows can reuse `ROWIDs` of previously deleted rows. However, this may not free all the space in a table that already has many deleted objects. For best results, you should set this option immediately after you create your database. This option is safe for pure relational applications. However, SQL applications that use `ROWID` and `OKAPI` applications that use direct pointers between objects need to verify that all references to an object are set to `NULL` before the object is deleted. Otherwise, dangling references may eventually point to some other, unrelated object.
`compress`	This option (which is "on" by default) enables run-length compression of objects. Run-length compression takes very little CPU time, so you should only deselect (`-compress`) this option if: ■ Operating system-level file compression is used, such as `DriveSpace` or a `NTFS` compressed attribute. In this case not compressing the same data twice provides a better compression ratio. ■ Most objects in the database are frequently updated to a highly compressible state (for example, all columns set to `NULL`), and the data cannot be compressed well (such as binary columns with random data). In these cases, using this option (`+compress`) can result in highly fragmented tables. Changing this option does not compress or decompress any existing objects in the database.

B.9 VALIDATEDB

Description

This command-line tool validates the structures within the database file and if the database structure is found to be corrupted, lists the errors found in a file designated by the user. The tool checks the following:

- Objects - Header information for database objects. Flags are checked for consistency in case the object was moved or compressed. Object length is checked against a valid range. If the object is a BLOB, the object's frames are checked against the volume page bitmap.
- Index page entries - Checks that the creation of an index page entry results in the correct number of nodes or list of object identifiers.
- Index pages - Checks that all key values on the page are sorted. All objects contained on the page are validated. Page descriptor information such as the number of objects, the number of free bytes, and the number of entries are checked against the actual objects on the page.
- Groups - As each page is validated, the group descriptor information is checked against the actual number of pages and objects.
- Indexes - All the pages are validated against the btree. The tool also validates all page pointers. All levels of the btree are checked to validate that key values are in the sorted order as a whole. For leaf elements of the btree, all OIDs from the leaf page entries are checked for consistency with the actual group objects.

Syntax

validatedb *DSName DBName* [-p *password*] [-t *schemaname.tablename*] -file *outputfilename*

Keywords and Parameters

DSName

The data source name. This can also be `NONE` if no DSN is present.

DBName

If there is a DSN present, this is the database file name (without the .odb extension) if it is different from the default filename for the DSN. If there is no DSN, then `VALIDATEDB` uses the current directory unless the full path is specified. If there is a log file in the same directory as the database file, it is also validated.

password

Password for an encrypted database.

schemaname

Optional schema name. The default schema name is used unless this is specified.

tablename

Optional table name. The specified table is validated along with all of its indexes. If no table name is specified, the entire database is validated.

outputfilename

Optional filename for the text file where all errors and other related information revealed by `VALIDATEDB` are saved. The default is stdout.

Examples

```
validatedb polite polite -t emp -file out.txt
```

B.9.1 Using the VALIDATEDB Utility for the Palm Platform

The following sections describe how to use the `validatedb` utility:

- Section B.9.1.1, "Overview"
- Section B.9.1.2, "Installing VALIDATEDB"
- Section B.9.1.3, "Running VALIDATEDB"
- Section B.9.1.4, "Sending Corrupted Databases"
- Section B.9.1.5, "Sending Companion Databases"

B.9.1.1 Overview

In some cases, the Oracle Database Lite database on Palm may become corrupted. It can be caused by hardware problems or bugs in the database code. Using the `validatedb` utility, you can inspect and diagnose database corruptions. As database users and application developers, you can execute the `validatedb` utility to check the database for consistency. The Oracle Database Lite development group uses the `validatedb` utility to diagnose the extent of corruption and fixes the problem.

B.9.1.2 Installing VALIDATEDB

To install the `validatedb` utility, install a single file named `validatedb.prc` that is located in the `Lite\Runtime` directory. Install the PRC file on your Palm device or emulator using the HotSync application.

B.9.1.3 Running VALIDATEDB

To run the `validatedb` utility, perform the following steps.

1. Click the `validatedb` icon. The main `validatedb` form appears. As Table B–7 describes, the `validatedb` form contains the following items.

Table B–7 `validatedb` ***Form Description***

Item	Description
Oracle Database Lite database list	List of Oracle Database Lite databases installed on the device (or emulator).
Log to Desktop	If selected, logs database information to your desktop.

2. To validate a database, choose the Oracle Database Lite database from the list displayed. To validate all databases, click **Validate All**.

> **Note:** You will be prompted for the password of each encrypted database that requires validation.

After validation, the `validatedb` utility displays one of the following alerts:

- No Errors Found—No corruption has been detected.
- CorruptedDB—The utility has detected some corruption and the databases need to be sent to Oracle Support for further investigation.
- System Fatal—The utility has detected system fatal alerts and the databases need to be sent to Oracle Support for further investigation. You should reset the device if you receive a system fatal alert.
- Assertion - The utility has detected some errors and the databases need to be sent to Oracle Support for further investigation.

Note: Do not select the "Log to Desktop" box as it is used by Oracle Support to log further debugging information to a desktop computer. If you select this box, the `validatedb` utility stops functioning and does not respond.

B.9.1.4 Sending Corrupted Databases

If the `validatedb` utility detects corruption in the database, it is mandatory that you send all such databases to Oracle Support. To send corrupt databases, click **BackupAll**. This command sets up the backup flag for all Oracle Database Lite databases on the device. During the next HotSync instance, the Oracle Database Lite databases are backed up on the desktop computer. After running the HotSync application, you will find these databases in the directory of the HotSync manager named `Palm\<HotSync user name>\Backup`.

B.9.1.5 Sending Companion Databases

Send along the following companion databases along with the corrupted database to Oracle Support:

- All databases which appear in the list on the `validatedb` form.
- Databases that start with the same name as the ones on the list, but contain the extension $1, $2, ...,. For example, `OrdersODB$1.PDB`, `OrdersODB$2.PDB`, and so on. These are Oracle Database Lite extensions for large databases.
- `okSysDB.PDB`
- `okTransLog.PDB`

To correct corrupted databases, Oracle Support retrieves further debugging information using the `validatedb` utility for diagnosis.

B.10 Transferring Data Between a Database and an External File

You can transfer data between an external file and the Oracle Lite database through either a command-line tool or programmatic APIs, as described in the following sections:

- Section B.10.1, "OLLOAD"
- Section B.10.2, "Oracle Database Lite Load Application Programming Interfaces (APIs)"

B.10.1 OLLOAD

The Oracle Database Lite Load Utility (`OLLOAD`) is a command-line tool, which enables you to load data from an external file into a table in Oracle Database Lite or to unload (dump) data from a table in Oracle Database Lite to an external file. Unlike SQL*Loader, `OLLOAD` does not use a control file in which you supply all data parameters and format information on the command-line.

When loading data, `OLLOAD` takes an input file that contains one record per line with a separator character between fields. The default field separator is a comma (,). These records can also include fields with values that are quoted strings. The default value is single quote (’). For more information on data parsing, see "Data Parsing".

B.10.1.1 Syntax

Loading a Datafile

To load a datafile, use the following syntax.

olload [options] -load *dbpath tbl* [*col1 col2 ...*] [<*datafile*]

Unloading (dump) to an Outfile

olload [options] -dump *dbpath tbl* [col1 col2 ...] [>*outfile*]

B.10.1.2 Keywords and Parameters

This section describes keywords and parameters that are available for the `OLLOAD` utility.

[options]

For a list of options, see Section B.10.1.2.1, "Options".

-load

To use the load utility.

-dump

To use the unload (dump) utility.

dbpath

The path to the Oracle Database Lite (`.odb`) file.

tbl

The table name. `OLLOAD` first attempts to find a table name in the user-specified case. If this fails, it searches for the upper-case of the user-specified name.

Note: The default user is `SYSTEM`. To specify an `OLLOAD` operation for another user name's tables, prefix the `tbl` parameter with the user name and a dot (.).

col1 col2

The column names. `OLLOAD` first attempts to find a column name in the user-specified case. If this fails, it searches for the upper-case of the user-specified name.

[datafile] [outfile]

The source or destination file for the load or unload operations. If you do not specify a datafile or outfile, `OLLOAD` displays the output on the screen.

B.10.1.2.1 Options This section describes keyword and parameter options that are available for the `OLLOAD` utility.

-sep *character*
The field separator. If you do not specify this option, `OLLOAD` assumes that the separator character is a comma (,).

-quote *character*
The quote character. If you do not specify this option, `OLLOAD` assumes that the quote character is a single quote (').

-file *filename*
Use this option when loading and unloading data to specify the source or destination file name. When loading data, filename specifies the source file to load into Oracle Database Lite. When unloading (dumping) data, it is the destination file for the unloaded data.

> **Note:** To unload data from Oracle Database Lite and load (or pipe) it to another Oracle Database Lite, do not specify a file name for this option. For a description of sample syntax, see "Examples".

-log *logfile*
Specify this option if you want to produce a log file listing rows that OLLOAD could not insert during load. If you do not specify a log file, loading stops at the first error.

-passwd *passwd*
The connection password for an encrypted database. You need to supply this password so that loading and unloading can occur.

-nosingle
Specify this option when you do not want to use single user mode. This degrades performance but allows other connections to the database.

-readonly
Specify this option when unloading data from a read-only Oracle Database Lite, for example, one located on a CD-ROM.

-commit *count*
Use this option if you want OLLOAD to commit after processing a specified number of rows. The default is 10000. OLLOAD prints an asterisk (*) to the screen each time it commits the specified number of rows. To disable the commit operation specify 0.

-mark *count*
Use this option if you want OLLOAD to print a dot on the screen after processing the specified number of records. The default is 1000. To disable this feature specify 0.

Data Parsing
Table B–8 lists examples for `OLLOAD` data parsing.

Table B–8 Data Parsing Examples

Input	Data	Explanation
'Redwood Shores, CA'	Redwood Shores CA	Enclosing the input string in quotes preserves spaces and punctuations within a string.
'O''Brien'	O'Brien	Represent a single quote with its escape sequence, two single quotes.
fire fly	firefly	Spaces in data that is not quoted is ignored.
,	NULL,NULL	Empty fields are NULL.
1,,3	1,NULL,3,NULL	Empty fields are NULL.
	[no row inserted]	Completely empty lines are ignored.

If there are more values than database columns, extra values are ignored. Any missing values at the end of the line are set to NULL.

OLLOAD Utility Restrictions

OLLOAD does not support tab-delimited input files and LONG datatypes.

Examples

```
olload -quote \" -file p_kakaku.csv -load c:\orant\oldb40\polite.odb skkm01
olload -dump c:\orant\oldb40\polite.odb emp empno ename | olload -load myfile.odb
myemp
```

B.10.2 Oracle Database Lite Load Application Programming Interfaces (APIs)

This document describes the Oracle Database Lite Load APIs. Each section of this document presents a different topic. These topics include:

- Section B.10.2.1, "Overview"
- Section B.10.2.2, "Oracle Database Lite Load APIs"
- Section B.10.2.3, "File Format"
- Section B.10.2.4, "Limitations"

B.10.2.1 Overview

The Oracle Database Lite Load APIs allow you to load data from an external file into a table in Oracle Database Lite, or to unload (dump) data from a table in Oracle Database Lite to an external file. For information on using the command line tool OLLOAD, see Section B.10.1, "OLLOAD". You can use the API calls presented in this document to make your own customizations.

B.10.2.2 Oracle Database Lite Load APIs

The Oracle Database Lite Load APIs include:

- Section B.10.2.2.1, "Connecting to the Database: olConnect"
- Section B.10.2.2.2, "Disconnecting from the Database: olDisconnect"
- Section B.10.2.2.3, "Deleting All Rows from a Table: olTruncate"
- Section B.10.2.2.4, "Setting Parameters for Load and Dump Operations: olSet"
- Section B.10.2.2.5, "Loading Data: olLoad"

- Section B.10.2.2.6, "Dumping Data: olDump"

The normal mechanism for unloading and loading a table is as follows:

1. Declare local variable, `DBHandle`.
2. Connect to the database using `olConnect`.
3. Optionally, set parameters for load or unload.
4. Dump or load the data using `olDump` or `olLoad`. You may optionally delete all rows from a table by calling `olTruncate`.
5. Disconnect from the database using `olDisconnect`.

B.10.2.2.1 Connecting to the Database: olConnect Use this API to connect to the database. This is the first API that you have to call. It creates a load and unload context that is used in subsequent APIs to influence the load and unload behavior. This returns an initialized database handle DBHandle.

Syntax

```
olError olConnect (char *database_path, char *password, DBHandle &dbh);
```

The arguments for `olConnect` are listed in Table B–9:

Table B–9 olConnect Arguments

Argument	Description
`database_path`	The full path to the database file (directory path and filename).
`password`	The password used for the encrypted database, for any other database the password = NULL.
`dbh`	The application handle for the current database connection. This allows multiple database connections for one application thread (each connection has a different handle).

Return Values

(short) integer error code

Values from -1 to -8999 are used for the error codes returned by the database, values from -9000 and below are used for olLoad-specific error codes.

B.10.2.2.2 Disconnecting from the Database: olDisconnect Disconnects from the database.

Syntax

```
olError olDisconnect (DBHandle dbh);
```

The arguments for `olDisconnect` are listed in Table B–10:

Table B–10 olDisconnect Arguments

Argument	Description
`dbh`	The current application handle.

Return Value

(short) integer error code

B.10.2.2.3 Deleting All Rows from a Table: olTruncate

This API can be used to delete all rows from an existing table.

> **Note:** Records removed from the server through a truncate command will not be removed from the client unless a complete refresh is triggered. The truncate command is considered a DDL operation. Consequently, the necessary DML triggers do not fire and therefore the operations are not logged for fast refresh.

Syntax

```
olError olTruncate (DBHandle  dbh, char* table );
```

The arguments for `olTruncate` are listed in Table B–11:

Table B–11 olTruncate Arguments

Argument	Description
`dbh`	The current application handle.
`tablename`	The name of the table in the form: owner_name.table_name. where owner_name is the name of the owner of the table.

Return Value

(short) integer error code

B.10.2.2.4 Setting Parameters for Load and Dump Operations: olSet

This is an optional API. This sets optional parameters for load and unload.

Syntax

```
olError olSet (DBHandle  dbh, char * parameter_name, char *parameter_value);
```

The arguments for `olSet` are listed in Table B–12:

Table B–12 olSet Arguments

Argument	Description
`dbh`	The current application handle.
`parameter_name`	The name of the given parameter. This is not case sensitive. See Section B.10.2.3.2, "Parameters" for a list of parameter names and their default values.
`parameter_value`	The value to be set. This is not case sensitive for most parameters.

Return Value

(short) integer error code

B.10.2.2.5 Loading Data: olLoad

`OlLoad` loads data from a file into a table using current parameter settings.

Syntax

```
olError olLoad (DBHandle dbh, char *table, char *file);
```

The arguments for `olLoad` are listed in Table B–13:

Table B–13 olLoad Arguments

Argument	Description
dbh	The current application handle.
table	The table information in the form: `owner_name.table_name(col1,col2,...)` where `col1,col2,...` is the list of column names to load. This allows you to load and dump certain columns instead of the entire table. If the entire table is to be dumped, the column list need not be specified.
file	The path to the file from which loading takes place.

Note: If table = NULL, `olLoad` tries to find the table description in the file header.

Return Value

(short) integer error code

B.10.2.2.6 Dumping Data: olDump `OlDump` dumps data from a table into a file using current parameter settings.

Syntax

```
olError olDump (DBHandle dbh, char *table, char *file);
```

The arguments for `olDump` are listed in Table B–14:

Table B–14 olDump Arguments

Argument	Description
`dbh`	The current application handle.
`table`	The table information in the same form as `olLoad`.
`file`	The file to which dump data is written.

Return Value

(short) integer error code

B.10.2.2.7 Compiling The declarations for the DBHandle, parameter constants and flags, and error message codes are given in the file **olloader.h** in the *ORACLE_HOME*\Mobile\SDK\include directory. For compilation of your product include olloader.h in your main source file.

B.10.2.2.8 Linking Linking use the file **olloader40.dll** and the library file **olloader40.lib**. Include these files in your project settings.

B.10.2.3 File Format

The Oracle Database Lite Load APIs support three file formats FIXEDASCII, BINARY and CSV. Each file contains an optional header followed by zero or more rows of data.

B.10.2.3.1 Header Format The header has the following format (comments are in bold):

```
$$OL_BH$$ [begins header]
```

```
VERSION=xx.xx.xx.xx   [version number]
TABLE=T1(C1, C2, ...)... [table name with list of column names dumped]
FILEFORMAT=FIXEDASCII
SEPARATOR=,
[any other parameters in the parameter list can be listed here]
$$OL_EH$$ [ends header]
```

The following is a header example:

```
$$OL_BH$$
VERSION=01.01.01.01
TABLE=T1(EMPNO,SALARY)
FILEFORMAT=BINARY
BITARRAY=TRUE
HEADER=TRUE
RDONLY=FALSE
LOGFILE=
COMMITCOUNT=-1
NOSINGLE=TRUE
$$OL_EH$$
```

The header lines can be in any order and all lines except $$OL_BH$$ and $$OL_EH$$ can be considered optional. Although, during the dump, if the header flag is on, table information and all parameter settings are dumped into the header.

When executing load, parameter information in the header overwrites current parameter settings. If the table argument in `olLoad` is NULL, the table name and list of columns in the header prevails, otherwise the table argument of `olLoad` prevails over the header.

B.10.2.3.2 Parameters Header file parameters listed in Table B–15 are not case sensitive.

Table B–15 Parameters

Parameter	Description
`FILEFORMAT`	Input and output file format. The following formats are supported: ■ FixedASCII - text file with fixed field width for each datatype. ■ CSV – comma separated values format. ■ Binary - binary file format. These key word values are not case sensitive.
`SEPARATOR`	The separator between the values (one character), comma by default.
`QUOTECHAR`	The quote character for the string datatype values in the file, single quote (') by default.
`LOGFILE`	The log file name. NULL by default (no log file produced and loading stops at the first error).
`NOSINGLE`	FALSE for single user mode (the default), or TRUE for no single user mode.
`READONLY`	FALSE (the default). TRUE to dump the data from read-only database (such as CD-ROM).
`COMMITCOUNT`	The number of rows processed after which `olLoad`, `olDump`, and `olTruncate` commit. The default value is -1, not to commit at all. Value 0 commits at the end of the operation, and values above 0 commit after the specified number of rows.

Table B–15 (Cont.) Parameters

Parameter	Description
`HEADER`	FALSE (the default). TRUE to create a header in the beginning of the file during `olDump`.
`BITARRAY`	TRUE (the default) to support writing and reading nulls in binary format. During the dump, a bit array with the null information is dumped before each row. For FALSE `olDump` provides an error trying to write nulls in binary.
`NONULL`	TRUE (the default) when trying to read or write nulls `olLoad` and `olDump` return an error. When the flag is set to FALSE nulls are supported, including binary format since the default BITARRAY value is TRUE.
`DATEFORMAT`	The string for which date and timestamp columns should be written into the file and read from the file in FIXED ASCII and CSV formats. Such formats as "YYYYMMDD", "YYYY-MM-DD", and "YYYY/MM/DD" are supported. The default value is empty string (which can also be set using NULL), and the default date format is "YYYY-MM-DD". (In Oracle mode, date is treated the same as timestamp so that the date format is the default timestamp format which is "YYYY-MM-DD HH:MM:SS.SSSSSS".)

B.10.2.3.3 Data Format The data format can be comma separated value (CSV), fixed ASCII, or binary. The following cases apply:

- CSV Format: Each row of the table is represented as a separate line in the file. Each line is separated by a carriage return and a line feed character on the Windows platform. Each value in the row is separated by a separator character which by default is a comma.

  Each value is also quoted by a quote character. Nulls are represented by an empty quoted string " ". The number of quoted strings in the file should be the same as the number of columns in the table, `olLoad` gives an error otherwise.

- FixedAscii Format: Each row of the table is represented as a separate line in the file. Each line is separated by a carriage return and a line feed character on the Windows platform. Each line is of the same size. The datatype of a column governs its format or representation in the file. Nulls are represented by a string of *n* '\0' (null) characters, where *n* is the fixed size of the field. Table B–16 describes data representation for each data type. The total record length for each line in the file should be the same as the sum of field lengths (precision) of each column, otherwise `olLoad` returns an error.

Table B–16 Datatypes

Datatype	Description
`CHAR(n)`	Length of the field in *n* characters. Data is left aligned and padded with blanks on the right.
`VARCHAR(n)`	Length of the field in n characters. Data is left aligned. It is padded with a null byte ('\0').

Table B–16 (Cont.) Datatypes

Datatype	Description
`NUMERIC(p,s)`	The default mode: length of the field is p+1 characters if scale s is zero or is not present. Otherwise, the length of the field is (p+2) characters. The value is right aligned in the output field. Format is optional negative sign, followed by zeros if required, followed by significant digits. If there is no negative sign, then '0' instead, for example, Number(5,2) 12.3 -> ' 012.30' -12.3 -> '-012.30' 1.23 -> ' 001.23' -1.23 -> '-001.23' The custom mode: the field length is one less: p if scale is not present, or zero and p+1 otherwise. The actual number stored in the file is of type NUMERIC(p-1, s). Correspondingly, `olDump` gives an error trying to insert a number within the range of NUMERIC(p, s), but out of the range of NUMERIC(p-1, s). Therefore, the first character in the NUMERIC field must be '0' or '-'; `olLoad` gives an error otherwise.
`DECIMAL(p,s)`	The same as NUMERIC(p,s).
`INTEGER`	Length of the field is 11 characters. A negative sign or space followed by 10 digits. Leading digits are filled with zeros.
`SMALLINT`	Field length is 6 characters. Minus sign or space followed by 5 digits.
`FLOAT`	Field length is 23 characters. In Oracle mode, it is minus sign or space, followed by leading zeroes, followed by some number of digits, followed by dot, followed by some number of digits. For example: 0 -> ' 0000000000000000000000' -12.34 -> '-0000000000000000012.34' In SQL92 mode the E (exponent) is always present and there is only 1 digit before the decimal point. For example: 0 -> ' 00000000000000000000E0' -12.34 -> '-000000000000001.234E10'
`REAL`	The same format as for double precision except that the total field length is only 16 characters instead of 23.
`DOUBLE PRECISION`	Field length is 23 characters. Minus sign or space followed by 22 characters which are digits, dot, or E, floating point number followed by E, followed by the exponent digits. In Oracle mode, if the number is small enough to fit in the field without using the exponent, E is not used. In SQL92 mode, E is always used. There is always one meaningful digit before the floating point, except 0. For example, in SQL92 mode: 0 -> ' 00000000000000000000E0' -1.79E10 -> '-0000000000000001.79E10' 12 -> ' 00000000000000001.2E10' For example, in Oracle mode: 1.2E75 -> ' 00000000000000001.2E75' -1.33333 -> '-0000000000000001.33333' -1.79E10 -> '-000000000017900000000'

Table B–16 (Cont.) Datatypes

Datatype	Description
`DATE`	In SQL92 mode: YYYY-MM-DD, 10 characters long, for example: October 1, 1999 -> 1999-10-01 In Oracle mode the date is dumped as timestamp. If it is not the default date format parameter, the date format corresponds to the specified date format string, for example: DATEFORMAT = "YYYYMMDD" October 1, 1999 -> 19991001
`TIME`	HH:MM:SS, 8 characters long, for example: 5:01:58 p.m. is 17:01:58
`TIMESTAMP`	Date format, space, time format, dot, 6 digits after dot (precision of microseconds), total length of 26 characters: YYYY-MM-DD HH:MM:SS.SSSSSS If it is not the default date format parameter, the timestamp format corresponds to the specified date format string. If no time is specified in the date format string, the time information in the timestamp is omitted when dumping into a file.

B.10.2.4 Limitations

Currently `olLoad` does not support the following features:

- Columns of the datatype Interval, Time with time zone, Timestamp with time zone, BLOB, and CLOB.
- Binary data is not supported.
- The only "var" type supported is varchar.

C ODBC Support on Palm

This document describes the Open Database Connectivity (ODBC) support provided in Oracle Database Lite 10*g* for the Palm OS Platform. Topics include:

- Section C.1, "ODBC Support"

C.1 ODBC Support

For the Palm OS platform, Oracle Database Lite 10*g* supports a subset of the ODBC 3.0 application programming interface standard. Using the ODBC API, applications can access data stored in Oracle Database Lite from your Palm handheld device.

The Oracle Database Lite ODBC library supports the Dynamic SQL model, in which applications can construct SQL statements at runtime and execute them directly on the handheld device.

The supported ODBC API functions are listed in Table C–1.

Table C–1 ODBC API Functions

Function	Description
SQLAllocConnect	Allocates memory for a connection handle using the specified environment.
SQLAllocEnv	Allocates memory for an environment handle.
SQLAllocHandle	A generic function for allocating environment, connection, and statement handles.
SQLAllocStmt	Allocates memory for a statement handle using the specified connection.
SQLFreeConnect	Disconnects from the connected database using the specified handle, and frees the handle.
SQLFreeEnv	Frees the specified handle. Uncommitted transactions associated with the handle are rolled back.
SQLFreeHandle	A generic handle to free environment, connection, and statement handles.
SQLFreeStmt	Frees the specified statement handle and its associated temporary memory.
SQLConnect	Connects to a database and saves information about the connection in the provided connection handle.
SQLDisconnect	Disconnects and closes a previously connected database.
SQLBindParameter	Binds a data buffer to a parameter marker in a SQL statement.

Table C–1 (Cont.) ODBC API Functions

Function	Description
SQLPrepare	Compiles a SQL statement and stores the information in the provided statement handle.
SQLExecDirect	Compiles and executes the specified SQL statement.
SQLExecute	Executes the prepared SQL using SQLPrepare.
SQLFetch	Reads in a row of data from the result set. After calling the function, the cursor is positioned to the next row to be read.
SQLBindCol	Binds a buffer to a column in the result set.
SQLDescribeCol	Retrieves information about a column of the result set.
SQLError	Extracts details about the last error associated to the provided handles.
SQLGetData	Reads in a single column from the current row into the specified buffer.
SQLNumResultCols	Returns the number of columns in the result set.
SQLRowCount	Returns the number of rows affected by a SQL SELECT, UPDATE, or DELETE statement.
SQLTransact	Requests a commit or rollback for all active operations on all statements associated with an environment.

C.1.1 SQLAllocConnect

Allocates memory for a connection handle using the specified environment, `hEnv`.

Syntax

```
RETCODE SQLAllocConnect( hEnv, hDbc )
```

Arguments

The arguments for `SQLAllocConnect` are listed in Table C–2:

Table C–2 SQLAllocConnect Arguments

Type	Name	Description
SQLHENV	`hEnv`	Environment handle. If set to NULL, creates a new environment.
SQLHDBC*	`hDbc`	Pointer to a connection handle where the routine stores the address of the newly allocated memory.

Usage Note

This function is supported for backward compatibility with ODBC 2.0. New applications should be coded using the function `SQLAllocHandle` and the handle type SQL_HANDLE_DBC. Internally, `SQLAllocConnect` calls `SQLAllocHandle`.

Returns

`SQLAllocConnect` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified environment handle.

C.1.2 SQLAllocEnv

`SQLAllocEnv` allocates memory for an environment handle.

To share a single transaction for different connections and statement handles, pass in the same environment handle to `SQLAllocConnection`, `SQLAllocStmt`, or `SQLAllocHandle`. This way, the new handles inherit, and share, the same environment handle. When these handles are freed, the actual connections and transaction are not freed. The resources are not released until the original environment handle is freed.

Syntax

```
RETCODE SQLAllocEnv( hEnv )
```

Arguments

The arguments for `SQLAllocEnv` are listed in Table C–3:

Table C–3 SQLAllocEnv Arguments

Type	Name	Description
SQLHENV*	`hEnv`	Pointer to an environment handle.

Usage Note

This function is supported for backward compatibility with ODBC 2.0. New applications should be coded using the function `SQLAllocHandle` and the handle type SQL_HANDLE_ENV. Internally, `SQLAllocEnv` calls `SQLAllocHandle`.

Returns

`SQLAllocEnv` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` and pass in NULL as the handle parameter.

C.1.3 SQLAllocHandle

`SQLAllocHandle` is a generic function for allocating environment, connection, and statement handles.

This function replaces the old allocation functions for each individual handle types (`SQLAllocEnv`, `SQLAllocConnection`, and `SQLAllocStmt`).

A transaction table (new OKAPI environment) is created for each new environment handle. To share a single transaction for different connections and statement handles, pass in the same environment handle to `SQLAllocHandle` as the `inputHandle` argument. This way, the new handles inherit and share the same environment handle. When these handles are freed, the actual connections and transaction are not freed. The resources are not released until the original environment handle is freed. You can also share a connection using the same method.

Syntax

```
RETCODE SQLAllocHandle( handleType, inputHandle, outputHandle )
```

Arguments

The arguments for SQLAllocHandle are listed in Table C–4:

Table C–4 SQLAllocHandle Arguments

Type	Name	Description
SQLSMALLINT	`handleType`	The type of handle to allocate. See the following "Usage Note" for more information.
SQLHANDLE	`inputHandle`	The handle to base on the new handle. This is either an environment or connection handle. To create a new handle from scratch, pass in NULL.
SQLHANDLE*	`outputHandle`	Pointer to the storage for the newly create handle.

Usage Note

An application allocates different handles to use with different API functions. The handle provides a context for each function. The supported handle types are listed in Table C–5:

Table C–5 Handle Parameters

Handle	Type	Description
Environment	SQL_TYPE_ENV	Environment handles are used to create an environment. Each environment contains generic information that allows you to access the database. A new transaction is associated with a newly-created environment handle.
Connection	SQL_TYPE_DBC	A connection handle is used to open a connection to a specific Oracle Database Lite. Connections can be based on the same environment handle, hence sharing the same transaction across multiple database connections. However, a maximum of eight connections can share a single environment.
Statement	SQL_TYPE_STMT	The statement handle contains information about the compiled SQL statement and its result sets.

Returns

`SQLAllocHandle` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the `inputHandle` argument.

C.1.4 SQLAllocStmt

`SQLAllocStmt` allocates memory for a statement handle using the specified connection, `hDbc`.

Syntax

```
RETCODE SQLAllocStmt( hDbc, hStmt )
```

Arguments

The arguments for SQLAllocStmt are listed in Table C–6:

Table C–6 SQLAllocStmt Arguments

Type	Name	Description
SQLHDBC	`hDbc`	The connection handle to creating the new handle.
SQLHSTMT*	`hStmt`	Pointer to a statement handle.

Usage Note

This function is supported for backward compatibility with ODBC 2.0. New applications should be coded using the function `SQLAllocHandle`, and the handle type SQL_HANDLE_STMT. Internally, `SQLAllocStmt` calls `SQLAllocHandle`.

Returns

`SQLAllocStmt` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified connection handle.

C.1.5 SQLFreeConnect

`SQLFreeConnect` disconnects from the connected database using the specified handle, and frees the handle.

Syntax

```
RETCODE SQLFreeConnect(hDbc )
```

Arguments

The arguments for SQLFreeConnect are listed in Table C–7:

Table C–7 SQLFreeConnect Arguments

Type	Name	Description
SQLHDBC	`hDbc`	The connection handle to free.

Usage Note

This function is deprecated and is replaced by the new generic function `SQLFreeHandle`.

Returns

`SQLFreeConnect` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified environment handle.

C.1.6 SQLFreeEnv

`SQLFreeEnv` frees the specified handle. Uncommitted transactions associated with the handle are rolled back.

Syntax

```
RETCODE SQLFreeEnv( hEnv )
```

Arguments

The arguments for SQLFreeEnv are listed in Table C–8:

Table C–8 SQLFreeEnv Arguments

Type	Name	Description
SQLHENV	`hEnv`	Environment handle to free.

Note

This function is deprecated and is replaced by the new generic function `SQLFreeHandle`.

Returns

`SQLFreeEnv` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified environment handle.

C.1.7 SQLFreeHandle

`SQLFreeHandle` is a generic function to free environment, connection, and statement handles.

The argument `handleType` is not used, because the handle internally contains information about how it is last used and therefore how it should be freed.

Syntax

```
RETCODE SQLFreeHandle( handleType, handle )
```

Arguments

The arguments for SQLFreeHandle are listed in Table C–9:

Table C–9 SQLFreeHandle Arguments

Type	Name	Description
SQLSMALLINT	`handleType`	The type of handle to free.
SQLHANDLE	`handle`	The handle to free.

Returns

`SQLFreeHandle` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified handle.

C.1.8 SQLFreeStmt

`SQLFreeStmt` frees the specified statement handle and its associated temporary memory.

Syntax

```
RETCODE SQLFreeStmt( hStmt, Option  )
```

Arguments

The arguments for SQLFreeStmt are listed in Table C–10:

Table C–10 SQLFreeStmt Arguments

Type	Name	Comments
SQLHSTMT	`hStmt`	Statement handle to free.
SQLUSMALLINT	`Option`	Use the value SQL-DROP to free handle. SQL-CLOSE is ignored.

Usage Note

This function is deprecated and is replaced by the new generic function `SQLFreeHandle`.

Returns

`SQLFreeStmt` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified environment handle.

C.1.9 SQLConnect

`SQLConnect` connects to a database and saves information about the connection in the provided connection handle. The handle must be previously allocated using the `SQLAllocateHandle` function.

Syntax

```
RETCODE SQLConnect( hConn, dbName, dbNameLen, userName, userNameLen, auth, authLen
)
```

Arguments

The arguments for SQLConnect are listed in Table C–11:

Table C–11 SQLConnect Arguments

Type	Name	Description
SQLHDBC	`hConn`	Newly allocated connection handle. If passed a connection handle that is in use, the function closes the existing connection.
SQLCHAR*	`dbName`	Name of the database to connect to.
SQLSMALLINT	`dbNameLen`	Length of the database name.
SQLCHAR*	`userName`	This argument is not currently supported and is ignored.
SQLSMALLINT	`userNameLen`	This argument is not currently supported and is ignored.
SQLCHAR*	`auth`	Database encryption password.
SQLSMALLINT	`authLen`	Database encryption password length.

Returns

`SQLConnect` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified connection handle.

C.1.10 SQLDisconnect

`SQLDisconnect` disconnects and closes a previously connected database.

If the environment used to make the connection is not committed before the connection is closed, committing afterwards fails.

Syntax

```
RETCODE SQLDisconnect( hDbc )
```

Arguments

The arguments for SQLDisconnect are listed in Table C–12:

Table C–12 SQLDisconnect Arguments

Type	Name	Description
SQLHDBC	`hDbc`	Handle of connection to be disconnected.

Returns

`SQLDisconnect` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified connection handle.

C.1.11 SQLBindParameter

`SQLBindParameter` binds a data buffer to a parameter marker in a SQL statement. Parameter markers are denoted by "?" in the SQL statement.

Syntax

```
RETCODE SQLBindParameter( hStmt, paramNo, paramType, cType, sqlType, colDef,
scale, value, valueMaxSize, valueSize )
```

Arguments

The arguments for SQLBindParameter are listed in Table C–13:

Table C–13 SQLBlindParameter Arguments

Type	Name	Description
SQLHSTMT	`hStmt`	Statement handle.
SQLUSMALLINT	`paramNo`	The number of the parameter marker to bind to. Starts from 1, counted from left to right.
SQLSMALLINT	`paramType`	The parameter type. Currently, only SQL_PARAM_INPUT is supported.
SQLSMALLINT	`cType`	The C datatype of the parameter.

Table C–13 (Cont.) SQLBlindParameter Arguments

Type	Name	Description
SQLSMALLINT	`sqlType`	The SQL datatype of the parameter.
SQLUINTEGER	`colDef`	The precision of the parameter.
SQLSMALLINT	`scale`	The scale of the parameter.
SQLPOINTER	`value`	Pointer to the buffer where the parameter value is stored.
SQLINTEGER	`valueMaxSize`	The size of the parameter buffer.
SQLINTEGER*	`valueSize`	Actual size of the parameter value.

Returns

`SQLBindParameter` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR.

C.1.12 SQLPrepare

`SQLPrepare` compiles a SQL statement and stores the information in the provided statement handle.

Syntax

```
RETCODE SQLPrepare( hStmt, statement, statementLen )
```

Arguments

The arguments for `SQLPrepare` are listed in Table C–14:

Table C–14 SQLPrepare Arguments

Type	Name	Description
SQLHSTMT	`hStmt`	Statement handle.
SQLCHAR*	`statement`	SQL statement string.
SQLINTEGER	`statementLen`	Length of the SQL statement string.

Returns

`SQLPrepare` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified statement handle.

C.1.13 SQLExecDirect

`SQLExecDirect` compiles and executes the specified SQL statement.

Syntax

```
RETCODE SQLExecDirect( hStmt, statement, statementLen )
```

Arguments

The arguments for `SQLExecDirect` are listed in Table C–15:

Table C–15 SQLExecDirect Arguments

Type	Name	Description
SQLHSTMT	`hStmt`	Statement handle.
SQLCHAR*	`statement`	SQL statement string.
SQLINTEGER	`statementLen`	Length of the SQL statement string.

Returns

`SQLExecDirect` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified statement handle.

C.1.14 SQLExecute

`SQLExecute` executes the prepared SQL using `SQLPrepare`.

Syntax

```
RETCODE SQLExecute( hStmt )
```

Arguments

The arguments for `SQLExecute` are listed in Table C–16:

Table C–16 SQLExecute Arguments

Type	Name	Description
SQLHSTMT	`hStmt`	Statement handle

Returns

`SQLExecute` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified statement handle.

C.1.15 SQLFetch

`SQLFetch` reads in a row of data from the result set. After calling the function, the cursor is positioned to the next row to be read.

Application can call `SQLGetData` to read in the columns of the read-in row.

If the application called `SQLBindCol` to bind columns, `SQLFetch` stores data from the row in the specified buffers.

Syntax

```
RETCODE SQLFetch( hStmt )
```

Arguments

The arguments for `SQLFetch` are listed in Table C–17:

Table C–17 SQLFetch Arguments

Type	Name	Description
SQLHSTMT	`hStmt`	Statement handle

Returns

`SQLFetch` returns SQL_SUCCESS if a new row of data is read successfully.

If there are no more rows to be read, `SQLFetch` returns SQL_NO_DATA_FOUND.

If an error occurs, the function returns SQL_ERROR. To find out specifics about an error, the application can call `SQLError` with the specified statement handle.

C.1.16 SQLBindCol

`SQLBindCol` binds a buffer to a column in the result set. The buffer is updated when `SQLFetch` is called. New columns from the result set are then read in.

`SQLBindCol` can be called after or before the statement is prepared and executed, as long as it is called before `SQLFetch` is called.

Syntax

```
RETCODE SQLBindCol( hStmt, columnNo, targetType, targetValue, targetSize,
actualSize )
```

Arguments

The arguments for `SQLBindCol` are listed in Table C–18:

Table C–18 SQLBindCol Arguments

Type	Name	Description
SQLHSTMT	`hStmt`	Statement handle.
SQLUSMALLINT	`columnNo`	The number of the column of the result set to bind to.
SQLSMALLINT	`targetType`	The C datatype of the buffer.
SQLPOINTER	`targetValue`	Pointer to buffer to hold the column data.
SQLINTEGER	`targetSize`	Size of the buffer in bytes.
SQLINTEGER*	`actualSize`	Pointer buffer to hold the size of the data read. Can pass in NULL if you do not want the information.

Returns

`SQLBindCol` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified statement handle.

C.1.17 SQLDescribeCol

`SQLDescribeCol` retrieves information about a column of the result set.

Syntax

```
RETCODE SQLDescribeCol( hStmt, columnNo, columnName, columnNameMaxLen,
datatype, columnNameLen, columnSize, decimalDigits, nullable  )
```

Arguments

The arguments for `SQLDescribeCol` are listed in Table C–19:

Table C–19 SQLDescribeCol Arguments

Type	Name	Description
SQLHSTMT	`hStmt`	Statement handle.
SQLUSMALLINT	`columnNo`	The number of the column in the result.
SQLCHAR*	`columnName`	Pointer to string buffer to store the returned name of the column.
SQLSMALLINT	`columnNameMaxLen`	Size of the string buffer.
SQLSMALLINT	`*columnNameLen`	Returned size of the column name in bytes.
SQLSMALLINT*	`dataType`	Returned SQL datatype.
SQLUINTEGER*	`columnSize`	Returned size of the column.
SQLSMALLINT*	`decimalDigits`	Returned precision of the column.
SQLSMALLINT*	`nullable`	Set to 1 if column is nullable, or 0 if it is not.

Returns

`SQLDescribeCol` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified statement handle.

C.1.18 SQLError

`SQLError` extracts details about the last error associated with the provided handles.

Syntax

```
RETCODE SQLError( hEnv, hConn, hStmt, sqlState, nativeError, messageText,
messageMaxSize, messageLength)
```

Arguments

The arguments for `SQLError` are listed in Table C–20:

Table C–20 SQLError Arguments

Type	Name	Description
SQLHENV	`hEnv`	Environment handle.
SQLHDBC	`hConn`	Database handle.
SQLHSTMT	`hStmt`	Statement handle.
SQLCHAR*	`sqlState`	Pointer to string buffer to store the returned SQLSTATE.
SQLINTEGER*	`nativeError`	Native error code.

Table C–20 (Cont.) SQLError Arguments

Type	Name	Description
SQLCHAR*	`messageText`	Error message text.
SQLSMALLINT	`messageMaxSize`	Size of buffer passed in.
SQLSMALLINT*	`messageLen`	Length of returned message text.

Returns

`SQLError` returns SQL_SUCCESS if it can retrieve information related to the last error. If there were no errors associated with the specified handle, the function returns SQL_NO_DATA_FOUND.

C.1.19 SQLGetData

`SQLGetData` reads in a single column from the current row into the specified buffer. The routine attempts to convert the data to the target buffer's type.

Syntax

```
RETCODE SQLGetData( hStmt, columnNo, targetType, targetValue, targetSize,
actualSize )
```

Arguments

The arguments for `SQLGetData` are listed in Table C–21:

Table C–21 SQLGetData Arguments

Type	Name	Description
SQLHSTMT	`hStmt`	Statement handle.
SQLUSMALLINT	`columnNo`	The number of the column.
SQLSMALLINT	`targetType`	The type of the buffer target Value.
SQLPOINTER	`targetValue`	Pointer to the buffer to store the result column data.
SQLINTEGER	`targetSize`	Size of the buffer.
SQLINTEGER*	`actualSize`	Actual number of bytes read into the specified buffer.

Returns

`SQLGetData` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified statement handle.

C.1.20 SQLNumResultCols

`SQLNumResultCols` returns the number of columns in the result set.

Syntax

```
RETCODE SQLNumResultCols( hStmt, columnCount )
```

Arguments

The arguments for `SQLNumResultCols` are listed in Table C–22:

Table C–22 SQLNumResultCols Arguments

Type	Name	Description
SQLHSTMT	`hStmt`	Statement handle.
SQLSMALLINT*	`columnCount`	Pointer to buffer to store the returned number of columns in the result set.

Returns

`SQLNumResultCols` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified statement handle.

C.1.21 SQLRowCount

`SQLRowCount` returns the number of rows affected by a SQL SELECT, UPDATE, or DELETE statement.

Syntax

```
RETCODE SQLRowCount( hStmt, rowCount )
```

Arguments

The arguments for `SQLRowCount` are listed in Table C–23:

Table C–23 SQLRowCount Arguments

Type	Name	Description
SQLHSTMT	`hStmt`	Statement handle.
SQLINTEGER*	`rowCount`	Number of rows in the result set.

Returns

`SQLRowCount` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified statement handle.

C.1.22 SQLTransact

`SQLTransact` requests a commit or rollback for all active operations on all statements associated with an environment.

Syntax

```
RETCODE SQLTransact( hEnv, hDbc, completionType )
```

Arguments

The arguments for `SQLTransact` are listed in Table C–24:

Table C–24 SQLTransact Arguments

Type	Name	Description
SQLHENV	`hEnv`	Environment handle.
SQLHDBC	`hDbc`	Connection handle. Not used.
SQLUSMALLINT	`completionType`	The transaction action, which could be either SQL_COMMIT or SQL_ROLLBACK.

Returns

`SQLTransact` returns SQL_SUCCESS if it is successful. Otherwise, it returns SQL_ERROR. To find out the specifics about an error, the application can call `SQLError` with the specified environment handle.

Glossary

Apache Server

The Apache Server is a public domain HTTP server derived from the National Center for Supercomputing Applications (NCSA).

Base Table

A source of data, either a table or a view, that underlies a view. When you access data in a view, you are really accessing data from its base tables.

Connected

Connected is a generic term that refers to users, applications, or devices that are connected to a server. The Mobile client for Web-to-go is "connected" when it is in online mode.

Database Object

A database object is a named database structure: a table, view, sequence, index, snapshot, or synonym.

Database Server

The Oracle database server is the third tier of the Mobile Server/Mobile Client Web model. It stores the application data.

Disconnected

Disconnected is a generic term that refers to users, applications, or devices that are not connected to a server. The Mobile client for Web-to-go is "disconnected" when it is in offline mode.

Foreign Key

A foreign key is a column or group of columns in one table or view whose values provide a reference to the rows in another table or view. A foreign key generally contains a value that matches a primary key value in another table. See also "Primary Key".

Index

An index is a database object that provides fast access to individual rows in a table. You create an index to accelerate the queries and sorting operations performed against the table's data. You also use indexes to enforce certain constraints on tables, such as unique and primary key constraints.

Indexes, once created, are automatically maintained and used for data access by the database engine whenever possible.

Integrity Constraint

An integrity constraint is a rule that restricts the values that can be entered into one or more columns of a table.

Java Applets

Java applets are small applications that are executed in the browser that extend the functionality of HTML pages by adding dynamic content.

JDBC

JDBC (Java Database Connectivity) is a standard set of java classes providing vendor-independent access to relational data. Modeled on ODBC, the JDBC classes provide standard features such as simultaneous connections to several databases, transaction management, simple queries, manipulation of pre-compiled statements with bind variables, and calls to stored procedures. JDBC supports both static and dynamic SQL.

JavaServer Pages

JavaServer Pages (JSP) is a technology that enables developers to change a page's layout without altering the page's underlying content. JSP, which uses HTML and pieces of Java code to combine the presentation of dynamic content with business logic.

Java Servlets

Java servlets are protocol and platform-independent server-side components that are written in Java. Java servlets dynamically extend Java-enabled servers and provide a general framework for services built using the request-response paradigm.

Join

A relationship established between keys (both primary and foreign) in two different tables or views. Joins are used to link tables that have been normalized to eliminate redundant data in a relational database. A common type of join links the primary key in one table to the foreign key in another table to establish a master-detail relationship. A join corresponds to a WHERE clause condition in a SQL statement.

Master-Detail Relationship

A master-detail relationship exists between tables or views in a database when multiple rows in one table or view (the detail table or view) are associated with a single master row in another table or view (the master table or view).

Master and detail rows are normally joined by a primary key column in the master table or view that matches a foreign key column in the detail table or view.

When you change values for the primary key, the application should query a new set of detail records, so that values in the foreign key match values in the primary key. For example, if detail records in the EMP table are to be kept synchronized with master records in the DEPT table, the primary key in DEPT should be DEPTNO, and the foreign key in EMP should be DEPTNO. See also "Primary Key" and "Foreign Key".

MIME

MIME (Multipurpose Internet Mail Extensions) is a message format used on the Internet to describe the contents of a message. MIME is used by HTTP servers to describe the type of file being delivered.

MIME Type

MIME Type is a file format defined by Multipurpose Internet Mail Extension (MIME).

Mobile client for Web-to-go

The Mobile client for Web-to-go is the client tier of the Web-to-Go model. It contains the Mobile Server and Oracle Database Lite. Web-to-Go replicates the user applications and data to the Mobile device. When the user synchronizes, Web-to-Go replicates any data changes to the Oracle database.

Mobile Development Kit for Web-to-go

The Mobile Development Kit for Web-to-go enables application developers to develop and debug Web-to-go applications that consist of Java servlets, JavaServer Pages (JSP), or Java applets.

Mobile Server

The Mobile Server resides on the application server tier of the three-tier Web-to-go model and processes requests from the Mobile client for Web-to-go to modify data in the Oracle database server. The Mobile Server can be configured to run with the Oracle application server or as a standalone Mobile Server.

Mobile Server Repository

The Mobile Server repository is a virtual file system. It is a persistent resource repository that contains all application files and definitions of the applications.

ODBC

ODBC (Open Database Connectivity) is a Microsoft standard that enables database access on different platforms. You can enable ODBC support on the Mobile client for Web-to-go for troubleshooting purposes. ODBC support enables you to view the client's data, which is stored on the local Oracle Database Lite. To view this information, you can use Mobile SQL.

Oracle Database

The Oracle database is the database component of the Mobile Server.

Oracle Database Lite

Oracle Database Lite is a small footprint relational database.

Packaging Wizard

The Packaging Wizard enables administrators to package and publish Mobile applications to the Mobile Server repository. Developers can use the Packaging Wizard to create a new application or to edit an existing application definition.

Positioned DELETE

A positioned DELETE statement deletes the current row of the cursor. Its format is:

```
DELETE FROM table
   WHERE CURRENT OF cursor_name
```

Positioned UPDATE

A positioned UPDATE statement updates the current row of the cursor. Its format is:

```
UPDATE table SET set_list
   WHERE CURRENT OF cursor_name
```

Primary Key

A table's primary key is a column or group of columns used to uniquely identify each row in the table. The primary key provides fast access to the table's records, and is frequently used as the basis of a join between two tables or views. Only one primary key may be defined per table.

To satisfy a PRIMARY KEY constraint, no primary key value can appear in more than one row of the table, and no column that is part of the primary key can contain a NULL value.

Publication Item

A publication item is a SQL select statement that specifies which data subset a client can access. A publication item usually corresponds to a replica table on the client device. You can create publication items using the Mobile Server Admin API. This API contains Java functions that implement the publish/subscribe model. You can call the functions in this API from within Java programs as standard function calls.

Referential Integrity

Referential integrity is defined as the accuracy of links between tables in a master-detail relationship that is maintained when records are added, modified, or deleted.

Carefully defined master-detail relationships promote referential integrity. Constraints in your database enforce referential integrity at the database (the server in a client/server environment).

The goal of referential integrity is to prevent the creation of an orphan record, which is a detail record that has no valid link to a master record. Rules that enforce referential integrity prevent the deletion or update of a master record, or the insertion or update of a detail record, that creates an orphan record.

Replication

Replication is the process of copying and maintaining database objects in multiple databases that make up a distributed database system. Changes applied at one site are captured and stored locally before being forwarded and applied at each of the remote locations. Replication provides users with fast, local access to shared data, and protects the availability of applications because alternate data access options exist. Even if one site becomes unavailable, users can continue to query or even update the remaining locations.

Replication Conflict

Replication conflicts occur when contradictory changes to the same data are made.

Schema

A schema is a named collection of database objects, including tables, views, indexes, and sequences.

Sequence

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

Sequence Window

The sequence window contains a unique range of values. The range of values never overlaps with those of other clients. When a client uses all the values in the range of its sequence window, the Mobile client recreates the sequence with a new, unique range of values.

Snapshots

A snapshot is a subset of application data for a specific user. For each user, the publication (within the application) contains a SQL query that defines the information relevant to this user. This information is known as the snapshot.

The Mobile client retrieves the appropriate data from the Oracle database and downloads to the client before it goes offline. A snapshot can be a copy of an entire database table, or a subset of rows from the table. The first time a user synchronizes, the Mobile client automatically creates the snapshots on the client machine. Each subsequent time that a user synchronizes, the Mobile client either refreshes the snapshots with the most recent data or recreates them depending on the complexity of the snapshot.

SQL

SQL, or Structured Query Language, is a non-procedural database access language used by most relational database engines. Statements in SQL describe operations to be performed on sets of data. When a SQL statement is sent to a database, the database engine automatically generates a procedure to perform the specified tasks.

Synchronization

Synchronization is the process the Mobile client uses to replicate data between the Mobile client and the Oracle database. The Mobile client replicates the user applications and data to Oracle Database Lite when the user synchronizes. The Mobile client replicates any data changes made on the client to the Oracle database.

Synonym

A synonym is an alternative name, or alias, for a table, view, sequence, snapshot, or another synonym.

Table

A table is a database object that stores data that is organized into rows and columns. In a well designed database, each table stores information about a single topic (such as company employees or customer addresses).

Three-Tier Web Model

The three-tier Web model is an Internet database configuration that contains a client, a middle tier, and an Oracle database server. Web-to-go architecture follows the three-tier Web model.

Transaction

A set of changes made to selected data in a relational database. Transactions are usually executed with a SQL statement such as INSERT, UPDATE, or DELETE. A transaction is complete when it is either committed (the changes are made permanent) or rolled back (the changes are discarded).

A transaction is frequently preceded by a query, which selects specific records from the database that you want to change. See also "SQL".

Unique key

A table's unique key is a column or group of columns that are unique in each row of a table. To satisfy a UNIQUE KEY constraint, no unique key value can appear in more than one row of the table. However, unlike the PRIMARY KEY constraint, a unique key made up of a single column can contain NULL values.

View

A view is a customized presentation of data selected from one or more tables (or other views). A view is like a "virtual table" that allows you to relate and combine data from multiple tables (called base tables) and views. A view is a kind of "stored query" because you can specify selection criteria for the data that the view displays.

Views, like tables, are organized into rows and columns. However, views contain no data themselves. Views allow you to treat multiple tables or views as one database object.

Web-to-go

Oracle Web-to-go is a framework for the creation and deployment of Mobile, Web-based, database applications. Web-to-go contains a three-tier database architecture consisting of the Mobile client for Web-to-go, the Mobile Server and Oracle database. It is centrally managed from the server and Web-to-go applications can be run when Web-to-go connected to the server (online) or disconnected from the server (offline). When Web-to-go is offline it caches data locally and synchronizes the data with the server when it goes back online.

Workspace

The Mobile Server Workspace is a Web page that provides users with access to Web-to-go applications. Web-to-go generates the Workspace in the user's browser after the user logs in to Web-to-go. The Workspace displays icons, links, and descriptions of all applications that are available to the user. An application is available to the user after the administrator publishes it to the Web-to-go system and grants access privileges to the user.

Index