

Oracle® Database Lite

Developer's Guide

10g (10.0.0)

Part No. B13788-01

June 2004

Oracle Database Lite Developer's Guide 10g (10.0.0)

Part No. B13788-01

Copyright © 2003, 2004, Oracle. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Send Us Your Comments	xvii
Preface	xix
Intended Audience.....	xix
Documentation Accessibility	xix
Structure	xix
1 Overview	
1.1 Introduction	1-1
1.2 Oracle Database Lite 10g Application Model and Architecture	1-3
1.2.1 Oracle Database Lite RDBMS.....	1-4
1.2.2 Mobile Sync	1-4
1.2.3 Mobile Server	1-5
1.2.4 Message Generator and Processor (MGP)	1-6
1.2.5 Mobile Server Repository	1-6
1.3 Mobile Development Kit (MDK)	1-7
1.3.1 Mobile SQL (MSQL)	1-8
1.3.2 Using the Packaging Wizard.....	1-8
1.4 Supported Platforms.....	1-9
1.5 Java Support	1-9
1.6 Data Source Name	1-9
2 The Oracle Database Lite RDBMS	
2.1 Introduction	2-1
2.2 Development Interfaces	2-1
2.2.1 Development Interface Overview	2-2
2.2.1.1 JDBC	2-2
2.2.1.2 Starting a Multi User Oracle Database Lite Database Service	2-2
2.2.1.3 Accessing the Multi User Oracle Database Lite 10g Database Service	2-2
2.2.1.4 ODBC.....	2-5
2.2.1.5 SODA.....	2-5
2.2.2 Mobile Sync Client Module Application Programming Interfaces (APIs).....	2-6
2.2.3 Oracle Database Lite Load APIs.....	2-6
2.2.4 Oracle Database Lite Load Utility (OLLOAD)	2-6
2.2.5 ADO.NET.....	2-6

2.3	Using the Starter Database	2-6
2.4	Working With Your Database	2-7
2.4.1	Creating a New Database	2-7
2.4.2	Creating a Data Source Name with ODBC Administrator	2-7
2.4.3	Creating a New Database Using the Command-Line Utility	2-7
2.4.4	Connecting to a New Database	2-8
2.5	Creating Multiple Users	2-8
2.5.1	Pre-defined Roles	2-8
2.5.2	Creating Users	2-9
2.5.3	Dropping Users	2-9
2.5.4	Changing Passwords	2-9
2.5.5	Granting Roles	2-10
2.5.6	Granting Privileges	2-10
2.5.7	Revoking Roles	2-10
2.5.8	Revoking Privileges	2-10
2.5.9	Building Demo Tables	2-10
2.5.10	Populate Your Database Using Mobile SQL	2-10
2.5.11	Backing Up a Database	2-11
2.5.12	Encrypting and Decrypting a Database	2-11
2.6	Oracle Database Lite Transaction Support	2-11
2.6.1	Atomicity	2-11
2.6.2	Consistency	2-11
2.6.3	Isolation	2-12
2.6.3.1	Durability	2-13
2.6.3.2	Locking	2-13
2.6.3.3	Changing the Default Isolation Level	2-13
2.6.3.4	Supported Combinations of Isolation Levels and Cursor Types	2-13
2.6.4	Tuning the Application	2-14
2.7	Support for Linguistic Sort	2-14
2.7.1	Creating Linguistic Sort Enabled Databases	2-14
2.7.2	How Collation Works	2-14
2.7.3	Collation Element Examples	2-15
2.7.3.1	Sorting Normal Characters	2-15
2.7.3.2	Reverse Sorting of French Accents	2-15
2.7.3.3	Sorting Contracting Characters	2-15
2.7.3.4	Sorting Expanding Characters	2-16
2.7.3.5	Sorting Numeric Characters	2-16
2.8	Creating Snapshot Definitions	2-16
2.8.1	Creating a Snapshot Definition Declaratively	2-16
2.8.2	Creating the Snapshot Definition Programmatically	2-17
2.9	Using Oracle Database Lite Samples	2-18
2.9.1	Overview	2-18
2.9.2	BLOB Manager Example Notes	2-18
2.9.3	Running the Visual Basic Sample Application	2-19
2.9.3.1	Open Visual Basic	2-19
2.9.3.2	View the Sample Application Tables and Data	2-19
2.9.3.3	Open the Sample Application	2-20

2.9.3.4	View and Manipulate the Data in the EMP Table	2-20
2.9.4	ODBC Examples.....	2-20
2.9.4.1	What the Examples Do	2-20
2.9.4.1.1	odbctbl.....	2-20
2.9.4.1.2	odbcview.....	2-20
2.9.4.1.3	odbcfunc	2-21
2.9.4.1.4	odbctype	2-21
2.9.4.1.5	long.....	2-21
2.10	Limitations	2-21
2.11	Tracing	2-21
2.11.1	Enabling Trace Output.....	2-22
2.11.2	Basic Functions.....	2-22
2.11.3	SQL Tracing	2-23
2.11.3.1	The Tid Output	2-23
2.11.3.2	SQL Statement Output.....	2-23
2.11.3.3	Compilation Time Output.....	2-23
2.11.3.4	Bind Values Output.....	2-23
2.11.3.5	Explain Plan Output.....	2-24
2.11.3.6	Temporary Table Created Output	2-24
2.11.3.7	Table Name Output.....	2-24
2.11.3.8	Temporary Table Sorted Output	2-24
2.11.3.9	First Fetch Time Output	2-24

3 Synchronization

3.1	Overview	3-1
3.1.1	Synchronization Concepts.....	3-2
3.1.1.1	Publication Item	3-2
3.1.1.2	Publication	3-2
3.1.1.3	Application	3-2
3.1.1.4	Subscription.....	3-3
3.1.1.5	Data Subsetting	3-3
3.1.1.6	Shared Maps.....	3-3
3.1.2	Synchronization Example.....	3-3
3.2	Synchronization Process	3-5
3.2.1	Fast Refresh Synchronization.....	3-5
3.2.1.1	Client Upload and Download Operations.....	3-6
3.2.1.2	Mobile Server Apply Operation.....	3-6
3.2.1.3	Mobile Server Compose Operation.....	3-7
3.2.2	Complete Refresh Synchronization.....	3-7
3.2.3	Synchronizing an Encrypted Database.....	3-7
3.3	Mobile Sync Application Programming Interfaces (APIs)	3-7
3.4	The Publish and Subscribe Model and Oracle Database Lite Synchronization.....	3-8
3.4.1	The Publish and Subscribe Model Step by Step	3-9
3.5	Using Consolidator to Define the Sample11.java Example	3-9
3.5.1	Sample11.java	3-10
3.5.2	Create Required Tables Using Standard JDBC	3-12
3.5.3	Connecting to the Mobile Server	3-12

3.5.4	Creating Publications	3-13
3.5.4.1	CreatePublication	3-13
3.5.5	Creating Publication Items	3-14
3.5.5.1	CreatePublicationItem	3-14
3.5.5.2	Defining Publication Items for Updatable Multi-table Views	3-15
3.5.5.3	Data Subsetting	3-15
3.5.6	Sequence Support	3-15
3.5.7	Defining Client Subscription Parameters for Publications	3-16
3.5.7.1	SetSubscriptionParameter	3-16
3.5.8	Creating Publication Item Indexes	3-17
3.5.8.1	CreatePublicationItemIndex	3-17
3.5.8.2	Define Client Indexes	3-18
3.5.9	Adding Publication Items to a Publication	3-18
3.5.9.1	AddPublicationItem	3-18
3.5.9.2	Defining Conflict Rules.....	3-19
3.5.9.3	Using Table Weight.....	3-20
3.5.10	Creating Users	3-20
3.5.10.1	createUser	3-20
3.5.11	Drop User.....	3-21
3.5.11.1	dropUser	3-21
3.5.12	Subscribing Users to a Publication.....	3-21
3.5.12.1	CreateSubscription	3-21
3.5.13	Instantiating a Subscription	3-22
3.5.13.1	InstantiateSubscription	3-22
3.6	Other Standard Consolidator Functionality	3-22
3.6.1	Client Device Database DDL Operations.....	3-23
3.6.2	Change Password	3-23
3.6.2.1	setPassword.....	3-23
3.6.3	Remote Database Link Support.....	3-23
3.6.3.1	Publishing Synonyms for the Remote Object Using CreatePublicationItem....	3-24
3.6.3.2	Creating a Dependency Hint	3-25
3.6.3.3	Remove a Dependency Hint	3-26
3.7	Advanced Features for Customizing Consolidator	3-26
3.7.1	Compose Phase Customization Using MyCompose	3-27
3.7.1.1	Extending MyCompose as a User Defined Sub-Class	3-27
3.7.1.2	Primary MyCompose Methods	3-27
3.7.1.2.1	needCompose Method	3-27
3.7.1.2.2	doCompose Method.....	3-28
3.7.1.2.3	init Method	3-29
3.7.1.2.4	destroy Method.....	3-29
3.7.1.3	Subsidiary MyCompose Methods.....	3-30
3.7.1.3.1	getPublication	3-30
3.7.1.3.2	getPublicationItem	3-30
3.7.1.3.3	getPubItemDMLTableName	3-30
3.7.1.3.4	getPubItemPK.....	3-30
3.7.1.3.5	getBaseTables.....	3-31
3.7.1.3.6	getBaseTablePK	3-31

3.7.1.3.7	baseTableDirty	3-31
3.7.1.3.8	getBaseTableDMLLogName	3-31
3.7.1.3.9	getMapView()	3-32
3.7.1.4	Consolidator API Methods for Registering MyCompose Sub-Classes	3-32
3.7.1.4.1	RegisterMyCompose Method	3-32
3.7.1.4.2	DeRegisterMyCompose	3-33
3.7.2	Sync Discovery API	3-33
3.7.2.1	getDownloadInfo Method	3-34
3.7.2.2	DownloadInfo Class Access Methods	3-34
3.7.2.3	PublicationSize Class	3-35
3.7.3	Map Table Partition APIs	3-37
3.7.3.1	Create a Map Table Partition	3-38
3.7.3.2	Add Map Table Partitions	3-38
3.7.3.3	Drop a Map Table Partition	3-39
3.7.3.4	Drop All Map Table Partitions	3-39
3.7.3.5	Merge Map Table Partitions	3-39
3.7.4	Modifying a Publication Item Using AlterPublicationItem	3-40
3.7.4.1	Alter Publication Item	3-40
3.7.5	Fast Refresh and Update Operation for Multi-Table Publications (Views)	3-41
3.7.5.1	Updatable Parent Tables	3-41
3.7.5.2	Using Parent Table Hints and INSTEAD OF Triggers	3-41
3.7.5.2.1	Creating a Parent Hint	3-41
3.7.5.2.2	INSTEAD OF Triggers	3-41
3.7.5.3	Fast Refresh for Views	3-42
3.7.5.3.1	PrimaryKeyHint	3-42
3.7.5.4	Complete Refresh for Views	3-42
3.7.5.4.1	CompleteRefresh	3-43
3.7.6	Virtual Primary Key	3-43
3.7.6.1	Create Virtual Primary Key Column	3-43
3.7.6.2	Drop Virtual Primary Key Column	3-43
3.7.7	Caching Publication Item Queries	3-44
3.7.7.1	Enabling Publication Item Query Caching	3-44
3.7.7.2	Disabling Publication Item Query Caching	3-45
3.7.8	Binding User-Defined PL/SQL Procedures	3-45
3.7.9	Queue Interface for Customizing Replication	3-45
3.7.9.1	Queue Interface Operation	3-45
3.7.9.2	Queue Creation	3-46
3.7.9.3	Queue Interface PL/SQL Procedure	3-47
3.7.9.4	CreateQueuePublicationItem API	3-48
3.7.9.5	Defining a PL/SQL Package Outside the Repository	3-49
3.7.9.5.1	RegisterQueuePkg	3-49
3.7.9.5.2	GetQueuePkg	3-49
3.7.9.5.3	UnRegisterQueuePkg	3-49
3.7.10	Null Sync Callout	3-49
3.7.11	Foreign Key Constraints in Updatable Publication Items	3-49
3.7.11.1	Foreign Key Constraint Violation Example	3-50
3.7.11.2	Avoiding Constraint Violations with BeforeApply and After Apply	3-50

3.7.11.3	Avoiding Constraint Violations with Table Weights	3-51
3.7.12	Callback Customization for Before and After Compose/Apply	3-51
3.7.13	Callback Customization for DML Operations.....	3-52
3.7.13.1	DML Procedure Example	3-52
3.7.14	Restricting Predicate.....	3-54
3.7.15	Priority-Based Replication.....	3-54
3.7.16	Shared Maps	3-54
3.7.16.1	Concepts.....	3-54
3.7.16.2	Performance Attributes.....	3-55
3.7.16.3	Usage	3-55
3.7.16.4	Compatibility and Migration.....	3-56
3.8	Synchronization Errors and Conflicts.....	3-56
3.8.1	Versioning.....	3-57
3.8.2	Winning Rules	3-57
3.8.3	Resolving Conflicts Using the Error Queue.....	3-57
3.8.3.1	Execute Transaction	3-57
3.8.3.2	Purge Transaction.....	3-58
3.8.4	Space Constraints.....	3-58
3.9	Mapping Datatypes Between the Oracle Server and Clients	3-58
3.9.1	Oracle Database Lite Datatypes.....	3-58

4 Developing Mobile Web Applications

4.1	Setting up the Mobile Client.....	4-1
4.2	Developing and Testing the Application	4-1
4.2.1	Building Web-to-Go Applications.....	4-2
4.2.1.1	Static Components.....	4-2
4.2.1.2	Dynamic Components	4-2
4.2.1.3	Database Components	4-2
4.2.1.4	Database Connections.....	4-3
4.2.2	Application Roles.....	4-3
4.2.3	Developing JavaServer Pages	4-3
4.2.3.1	Mobile Server or Mobile Development Kit Web Server	4-3
4.2.3.2	Mobile Client for Web-to-Go	4-3
4.2.4	Developing Java Servlets for Web-to-Go	4-4
4.2.4.1	Limitations.....	4-4
4.2.4.2	Accessing Applications on the Mobile Development Kit for Web-to-Go	4-4
4.2.4.3	Creating a Servlet	4-4
4.2.4.3.1	Packages.....	4-5
4.2.4.3.2	Web-to-Go User Context	4-6
4.2.4.3.3	Database Connectivity in Java Code	4-6
4.2.4.3.4	Accessing the Mobile Server Repository.....	4-6
4.2.4.4	Running a Servlet	4-7
4.2.4.4.1	Registering Servlets Using wtgpack.exe	4-7
4.2.4.4.2	The webtogo.ora File.....	4-8
4.2.4.4.3	Using wtgdebug.exe	4-9
4.2.4.4.4	Using WebtoGoServer.class.....	4-9
4.2.4.4.5	Controlling Web Server Properties.....	4-10

4.2.4.4.6	Registering MIME Types.....	4-10
4.2.4.5	Debugging a Servlet.....	4-11
4.2.4.6	Accessing the Schema Directly in Oracle Database Lite.....	4-11
4.2.5	Using Web-to-Go Applets.....	4-11
4.2.5.1	Creating the Web-to-Go Applet.....	4-11
4.2.5.2	Creating the HTML Page for the Applet.....	4-12
4.2.5.2.1	Static HTML Page.....	4-12
4.2.5.2.2	HTML Page Generated from a Servlet.....	4-12
4.2.6	Developing Applet JDBC Communication.....	4-13
4.2.6.1	getConnection().....	4-13
4.2.6.2	Design Issue.....	4-14
4.2.7	Developing Applet Servlet Communication.....	4-14
4.2.7.1	Creating the Web-to-Go Servlet.....	4-14
4.2.7.1.1	getResultObject().....	4-15
4.2.7.1.2	setSessionID().....	4-15
4.2.7.1.3	showDocument().....	4-15
4.2.8	Debugging Web-to-Go Applications.....	4-16
4.2.8.1	Running Sample 1 Using Oracle9i JDeveloper.....	4-16
4.2.8.1.1	Creating a Debug Project.....	4-16
4.2.8.1.2	Creating a Library.....	4-18
4.2.8.1.3	Adding Files to the Project.....	4-19
4.2.8.1.4	Running and Debugging.....	4-19
4.2.8.1.5	Troubleshooting.....	4-20
4.2.9	Customizing the Workspace Application.....	4-20
4.2.9.1	Web-to-Go Parameters.....	4-21
4.2.9.2	Sample Workspace.....	4-21
4.2.10	Using the Mobile Server Admin API.....	4-21

5 Native Application Development

5.1	Supported Platforms.....	5-1
5.2	Java Support.....	5-1
5.3	Data Source Name.....	5-2
5.4	Mobile Sync Application Programming Interfaces (APIs).....	5-2
5.4.1	COM Interface.....	5-3
5.4.1.1	Features and Components.....	5-3
5.4.1.2	ISync Interface.....	5-3
5.4.1.3	ISyncOption Interface.....	5-4
5.4.1.4	Selective Synchronization.....	5-5
5.4.1.5	COM Interface SyncParam Settings.....	5-6
5.4.1.6	COM Interface TransportParam Parameters.....	5-7
5.4.1.7	ISyncProgressListener Interface.....	5-7
5.4.2	C/C++ Interface.....	5-8
5.4.2.1	ocSessionInit.....	5-9
5.4.2.2	ocSessionTerm.....	5-9
5.4.2.3	ocSaveUserInfo.....	5-9
5.4.2.4	ocDoSynchronize.....	5-10
5.4.2.5	ocSetTableSyncFlag.....	5-11

5.4.2.6	ocGetPublication.....	5-12
5.4.2.7	C/C++ Data Structures.....	5-13
5.4.2.7.1	ocEnv.....	5-13
5.4.2.7.2	ocTransportEnv	5-16
5.5	Using the Packaging Wizard.....	5-17

6 Oracle Database Lite 10g ADO.NET Provider

6.1	Classes	6-1
6.1.1	OracleConnection	6-1
6.1.2	Transaction Management	6-1
6.1.3	OracleCommand.....	6-2
6.1.4	OracleParameter and Prepared Statements.....	6-2
6.1.4.1	Parameters	6-2
6.1.5	OracleBlob and Large Object Support	6-2
6.1.6	OracleSync and Data Synchronization	6-3
6.2	Running the Demo.....	6-5
6.3	Limitations	6-6
6.3.1	Thread Safety.....	6-6

7 Developing Mobile Applications for Palm OS Devices

7.1	Installing Oracle Database Lite Runtime on the Device.....	7-1
7.2	Uninstalling or Replacing Oracle Database Lite Runtime	7-2
7.3	Running Oracle Database Lite on Palm OS Emulator.....	7-2
7.4	Running Oracle Database Lite on Palm OS Simulator	7-2
7.5	Using Oracle Database Lite Base Libraries.....	7-2
7.6	Building a SODA Application.....	7-3
7.7	Building a SODA Forms Application	7-3
7.8	Building an ODBC Application	7-3
7.9	Packaging your Application with Oracle Database Lite Runtime.....	7-3
7.10	Customizing Oracle Database Lite Runtime.....	7-4
7.11	Palm Shared Library Manager (PSLM).....	7-4

8 Palm Shared Library Manager (PSLM)

8.1	Overview	8-1
8.2	Trying out PSLM.....	8-1
8.3	Writing a PSLM Library	8-2
8.4	Building a Shared Library Project	8-3
8.5	Calling a PSLM Library from Your Application	8-5
8.6	Building an Application Using PSLM.....	8-6
8.7	Exceptions Across Modules.....	8-6
8.8	Cloaked Shared Libraries.....	8-6
8.9	Patching the CodeWarrior Runtime.....	8-7

9 Using Mobile Sync for Palm

9.1	Configuring mSync.....	9-1
9.2	Using HotSync to Synchronize Data with the Mobile Server	9-2

9.2.1	Configuring HotSync for a PalmOS Device.....	9-3
9.2.2	HotSync Timeout Errors	9-3
9.2.3	Configuring PalmOS Emulator for HotSync	9-3
9.3	Using Network Sync.....	9-3
9.3.1	Synchronizing Using a Cradle and Windows Desktop	9-4
9.3.2	Network Sync With PalmOS Emulator	9-4

10 Building Offline Mobile Applications for Win32: A Tutorial

10.1	Overview	10-1
10.2	Developing Offline Mobile Applications for Win32.....	10-1
10.2.1	Command Sequence.....	10-2
10.2.1.1	Step 1. Create TASK Table on the Server Database.....	10-2
10.2.1.2	Step 2. Define a Publication Item and Publish the Application.....	10-3
10.2.1.3	Step 3. Create Users and Subscriptions.....	10-5
10.2.1.4	Step 4. Install the Oracle Database Lite 10g Client and the Mobile Field Service Application and Data 10-6	
10.2.1.5	Step 5. Browse the TASK Snapshot and Update a Row.....	10-7
10.2.1.6	Step 6. Synchronize the Change with the Server	10-7
10.2.1.7	Step 7. Check your changes on the server and modify a server record	10-7
10.2.1.8	Step 8. Synchronize again to get the server changes.....	10-8
10.2.1.9	Step 9. Develop your Mobile Field Service Application Using Oracle Database Lite 10-8	
10.2.1.10	Step 10. Republish the Application with the Application Program.....	10-8

11 Building Offline Mobile Applications for Windows CE: A Tutorial

11.1	Overview	11-1
11.1.1	Before You Start	11-1
11.1.1.1	Application Development Computer Requirements	11-2
11.1.1.2	Client Device Requirements.....	11-2
11.2	Developing the Application	11-2
11.2.1	Creating Database Objects in the Oracle Server.....	11-2
11.2.1.1	The Pocket PC Transport Application Database Objects.....	11-2
11.2.2	Writing the Application Code.....	11-4
11.2.2.1	Transport Module (Transport.vb).....	11-4
11.2.2.2	Main Form (frmMain.vb)	11-4
11.2.2.3	View Packages (frmView.vb).....	11-4
11.2.2.4	Create Package (frmNew.vb).....	11-5
11.2.3	Compiling the Application.....	11-6
11.2.3.1	Creating CAB Files	11-6
11.2.3.2	Installing the Application from the CAB File.....	11-7
11.3	Packaging and Publishing the Application.....	11-7
11.3.1	Defining the Application Using the Packaging Wizard.....	11-7
11.3.1.1	Creating a New Application	11-7
11.3.2	Defining the Application Connection to the Oracle Database Server.....	11-9
11.3.3	Defining Snapshots.....	11-10
11.3.4	Publishing the Application.....	11-13

11.4	Administering the Application	11-14
11.4.1	Starting the Mobile Server	11-14
11.4.2	Launching the Mobile Manager	11-14
11.4.3	Creating a New User	11-15
11.4.4	Setting the Application Properties	11-16
11.4.5	Granting User Access to the Application	11-17
11.4.6	Starting the Message Generator and Processor (MGP)	11-17
11.5	Running the Application on the Pocket PC	11-18
11.5.1	Installing the Oracle Database Lite Mobile Client for Pocket PC	11-18
11.5.2	Installing and Synchronizing the Transport Application and Data	11-19

12 Building Mobile Web Applications: A Tutorial

12.1	Overview	12-1
12.1.1	Before You Start	12-1
12.1.1.1	Development Computer Requirements	12-1
12.1.1.2	Client Computer Requirements	12-2
12.2	Developing the Application	12-2
12.2.1	Step 1: Creating Database Objects in Oracle Database Lite	12-3
12.2.1.1	The To Do List Application Database Objects	12-3
12.2.1.2	Required Action	12-3
12.2.2	Step 2: Compiling the Application	12-4
12.2.2.1	Required Action	12-4
12.2.3	Step 3: Defining the Application and Registering the Servlet	12-5
12.2.3.1	The Packaging Wizard	12-5
12.2.3.2	Required Action	12-5
12.2.4	Step 4: Conducting a Trial Run	12-10
12.2.4.1	The Mobile Development Kit for Web-to-Go Web Server	12-10
12.2.4.2	Required Action	12-10
12.3	Packaging the Application	12-12
12.3.1	Step 1: Defining the Application	12-12
12.3.1.1	The Packaging Wizard	12-12
12.3.1.2	Required Action	12-12
12.3.2	Step 2: Specifying Database Details	12-14
12.3.2.1	Required Action	12-15
12.3.3	Step 3: Defining the Snapshot	12-15
12.3.3.1	The Snapshots Tab	12-15
12.3.3.2	Required Action	12-16
12.3.4	Step 4: Defining Sequences	12-19
12.3.5	Step 5: Creating SQL Files for the Application	12-21
12.3.5.1	Required Action	12-21
12.3.6	Step 6: Package the Application	12-21
12.3.6.1	Required Action	12-21
12.4	Publishing the Application	12-22
12.4.1	Step1: Create the Table Owner Account	12-22
12.4.2	Step 2: Create the Database Objects in the Oracle Database	12-22
12.4.2.1	Required Action	12-22
12.4.3	Step 3: Start the Mobile Server	12-23

12.4.3.1	Required Action.....	12-23
12.4.4	Step 4: Log on to the Mobile Server and Start the Mobile Manager	12-23
12.4.4.1	Required Action.....	12-23
12.4.5	Step 5: Upload the Application.....	12-25
12.4.5.1	Required Action.....	12-25
12.5	Administering the Application	12-27
12.5.1	Step 1: Starting the Mobile Manager	12-27
12.5.1.1	Required Action.....	12-27
12.5.2	Step 2: Using the Mobile Manager to Create a New User	12-28
12.5.2.1	Required Action.....	12-28
12.5.3	Step 3: Setting Application Properties	12-29
12.5.3.1	Required Action.....	12-29
12.5.4	Step 4: Granting User Access to the Application	12-30
12.5.4.1	Required Action.....	12-30
12.5.5	Step 5: Defining Snapshot Template Values for the User	12-31
12.5.5.1	Required Action.....	12-31
12.6	Running the Application on the Mobile Client for Web-to-Go.....	12-33
12.6.1	Step 1: Installing the Mobile Client for Web-to-Go.....	12-33
12.6.1.1	Required Action.....	12-33
12.6.2	Step 2: Logging into the Mobile Client for Web-to-Go	12-35
12.6.2.1	Required Action.....	12-35
12.6.3	Step 3: Synchronizing the Mobile Client for Web-to-Go.....	12-36
12.6.3.1	Required Action.....	12-37

13 Building Offline Mobile Web Applications Using BC4J: A Tutorial

13.1	Overview	13-1
13.1.1	Before You Start	13-2
13.1.1.1	Development Computer Requirements	13-2
13.2	Developing the Application	13-2
13.2.1	Creating the Database Connection.....	13-3
13.2.2	Creating the BC4J Component.....	13-11
13.2.3	Configuring the BC4J Component to Use the WTGJdbc Connection.....	13-13
13.2.4	Building and Deploying the BC4J Component as a Simple Archive	13-13
13.2.5	Writing the JSP Application to Access the BC4J Component	13-14
13.2.6	Deploying the JSP Application as a Simple Archive	13-16
13.3	Packaging the JSP Application.....	13-16
13.4	Publishing and Configuring the JSP Application from the Mobile Manager	13-18
13.5	Testing the BC4J Application	13-18
13.6	Running the BC4J Application on the Mobile Client for Web-to-Go	13-19
13.7	Deploying the Sample Application	13-19

A Optimizing SQL Queries

A.1	Optimizing Single-Table Queries	A-1
A.2	Optimizing Join Queries	A-2
A.2.1	Create an Index on the Join Column(s) of the Inner Table	A-2
A.2.2	Bypassing the Query Optimizer	A-2

A.3	Optimizing with Order By and Group By Clauses.....	A-3
A.3.1	IN Subquery Conversion	A-3
A.3.2	ORDER BY Optimization with No GROUP BY	A-3
A.3.3	GROUP BY Optimization with No ORDER BY	A-3
A.3.4	ORDER BY Optimization with GROUP BY	A-4
A.3.5	Cache Subquery Results	A-4

B Oracle Database Lite Load Application Programming Interfaces (APIs)

B.1	Overview	B-1
B.2	Oracle Database Lite Load APIs	B-1
B.2.1	Connecting to the Database: olConnect.....	B-2
B.2.2	Disconnecting from the Database: olDisconnect.....	B-2
B.2.3	Deleting All Rows from a Table: olTruncate	B-2
B.2.4	Setting Parameters for Load and Dump Operations: olSet	B-3
B.2.5	Loading Data: olLoad.....	B-3
B.2.6	Dumping Data: olDump	B-4
B.2.7	Compiling	B-4
B.2.8	Linking	B-4
B.3	File Format	B-4
B.3.1	Header Format	B-4
B.3.2	Parameters	B-5
B.3.3	Data Format	B-6
B.3.3.1	CSV Format	B-6
B.3.3.2	FixedAscii Format.....	B-6
B.4	Limitations	B-8

C Web-to-Go Sample Applications

C.1	Introduction	C-1
C.1.1	The Mobile Server.....	C-1
C.1.2	The Mobile Development Kit for Web-to-Go	C-1
C.1.3	Accessing Sample Programs from the Mobile Development Kit for Web-to-Go.....	C-1
C.1.4	Accessing Sample Programs from the Mobile Server	C-2
C.2	Sample 1 - Hello World.....	C-2
C.2.1	Source Code Location.....	C-2
C.2.2	Application Files	C-2
C.3	Sample 3 - Recording Tracker	C-2
C.3.1	Using Sample 3.....	C-2
C.3.2	Sample 3 Database Tables	C-3
C.3.3	Sample 3 Servlets	C-3
C.3.4	Sample 3 Resource Bundle	C-3
C.3.5	Source Code Location.....	C-3
C.3.6	Application Files	C-4
C.4	Sample 4 - Hello Applet.....	C-4
C.4.1	Sample 4 Servlets	C-4
C.4.2	Source Code Location.....	C-5
C.4.3	Application Files	C-5
C.5	Sample 6 - Image Gallery.....	C-5

C.5.1	Source Code Location.....	C-5
C.5.2	Application Files	C-5
C.6	Sample 7 - Employee Data Applet.....	C-6
C.6.1	Source Code Location.....	C-6
C.6.2	Application Files	C-7

D ODBC Support on Palm

D.1	ODBC Support.....	D-1
D.1.1	SQLAllocConnect.....	D-2
D.1.2	SQLAllocEnv	D-3
D.1.3	SQLAllocHandle	D-3
D.1.4	SQLAllocStmt	D-4
D.1.5	SQLFreeConnect	D-5
D.1.6	SQLFreeEnv	D-5
D.1.7	SQLFreeHandle.....	D-6
D.1.8	SQLFreeStmt.....	D-6
D.1.9	SQLConnect.....	D-7
D.1.10	SQLDisconnect.....	D-8
D.1.11	SQLBindParameter	D-8
D.1.12	SQLPrepare	D-9
D.1.13	SQLExecDirect.....	D-9
D.1.14	SQLExecute.....	D-10
D.1.15	SQLFetch	D-10
D.1.16	SQLBindCol	D-11
D.1.17	SQLDescribeCol	D-11
D.1.18	SQLError	D-12
D.1.19	SQLGetData	D-13
D.1.20	SQLNumResultCols	D-13
D.1.21	SQLRowCount	D-14
D.1.22	SQLTransact	D-14

Glossary

Index

Send Us Your Comments

Oracle Database Lite Developer's Guide 10g (10.0.0)

Part No. B13788-01

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: helplite_ca@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. 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 additional information, visit the Oracle Accessibility Program Web site at

<http://www.oracle.com/accessibility/>

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, 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, JAWS 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.

Structure

This guide includes the following topics:

- [Chapter 1, "Overview"](#)

Provides an introduction to Oracle Database Lite 10g 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, "Developing Mobile Web Applications"](#)
Discusses web based mobile application development.
- [Chapter 5, "Native Application Development"](#)
Discusses mobile application development for native platforms.
- [Chapter 6, "Oracle Database Lite 10g ADO.NET Provider"](#)
Discusses ADO.NET support for Windows CE.
- [Chapter 7, "Developing Mobile Applications for Palm OS Devices"](#)
Discusses building Oracle Database Lite 10g applications for Palm devices. Oracle Database Lite 10g 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 10g applications using Metrowerks CodeWarrior 9.
- [Chapter 8, "Palm Shared Library Manager \(PSLM\)"](#)
Discusses the Palm Shared Library Manager (PSLM).
- [Chapter 9, "Using Mobile Sync for Palm"](#)
Discusses using Mobile Sync (mSync) for Palm. mSync for PalmOS allows a user or a developer to synchronize data with the Mobile Server.
- [Chapter 10, "Building Offline Mobile Applications for Win32: A Tutorial"](#)
Guides you through the mobile application development process for the Win32 platform through a tutorial.
- [Chapter 11, "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 10g ADO.NET interface for Pocket PC through a tutorial.
- [Chapter 12, "Building Mobile Web Applications: A Tutorial"](#)
Guides you through the relevant phases of implementing a web application for mobile devices through a tutorial.
- [Chapter 13, "Building Offline Mobile Web Applications Using BC4J: A Tutorial"](#)
Enables you to create, deploy, and use a BC4J application through a tutorial.
- [Appendix A, "Optimizing SQL Queries"](#)
Provides tips on improving the performance of your SQL queries.
- [Appendix B, "Oracle Database Lite Load Application Programming Interfaces \(APIs\)"](#)
Describes the Oracle Database Lite Load APIs.
- [Appendix C, "Web-to-Go Sample Applications"](#)

Contains sample Web-to-Go applications.

- [Appendix D, "ODBC Support on Palm"](#)

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

This chapter provides an introduction to Oracle Database Lite 10g and presents an overview of the application development process, using the Mobile Development Kit and its components. This chapter discusses the following topics.

- [Section 1.1, "Introduction"](#)
- [Section 1.2, "Oracle Database Lite 10g Application Model and Architecture"](#)
- [Section 1.3, "Mobile Development Kit \(MDK\)"](#)
- [Section 1.4, "Supported Platforms"](#)
- [Section 1.5, "Java Support"](#)
- [Section 1.6, "Data Source Name"](#)

1.1 Introduction

Oracle Database Lite 10g 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 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 10g 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 Server 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 10g 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 10g 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 Oracle9i Application Server (Oracle9iAS) and the Oracle database server, Oracle9i.
2. Publishing the application to the server. This refers to installing all the components for an application on the Mobile Server. Oracle Database Lite 10g 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 10g 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 10g 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 10g 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 10g 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 data. The Mobile Server automatically updates the mobile clients that have older version of the application or the data.

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

In the Oracle Database Lite 10g 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*.

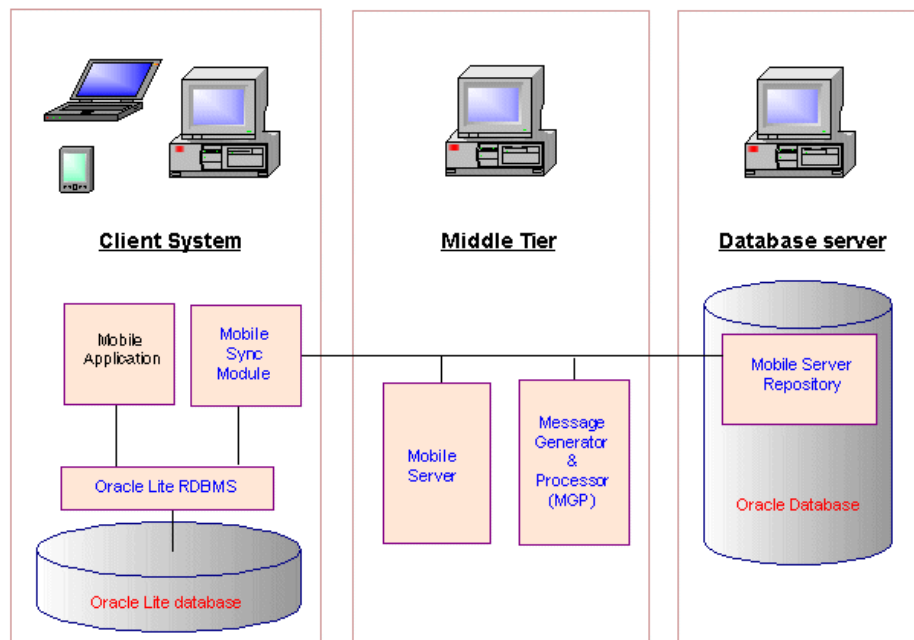
A subscription defines the relationship between a user and a publication. This is analogous to a newspaper or magazine subscription. Accordingly, once you subscribe to a particular publication, you begin to receive information associated with that publication. With a newspaper you receive the daily paper 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 logs in to the Mobile Server for the first time, the Mobile Server creates an Oracle Database Lite database on the client machine for each subscription that is provisioned to the user. The Mobile Server 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 10g 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 10g applications.

Figure 1–1 Oracle Database Lite 10g Architecture

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

1.2.1 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, 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 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 10g applications, only the RDBMS, the Mobile Sync, and Mobile SQL applications are installed on the client machine.

1.2.2 Mobile Sync

Mobile Sync 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, 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.

1.2.3 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. The Resource Manager is responsible for application publishing, application provisioning, and application installation. The Consolidator 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 10g 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 10g 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 module of the Mobile Server and downloads the applications and the data provisioned to you.

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

1.2.4 Message Generator and Processor (MGP)

The Consolidator 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.

1.2.5 Mobile Server Repository

The Mobile Server Repository (the Repository for short) 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.

The Repository should only be made using the Mobile Server **Mobile Manager** or the Resource Manager API of the Mobile Server.

1.3 Mobile Development Kit (MDK)

Before you develop an offline application using Oracle Database Lite 10g, you should install the Oracle Database Lite 10g 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 10g Installation and Configuration Guide for Windows NT/2000/XP*.

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

- Oracle Database Lite RDBMS - A lightweight, object-relational database management system
- Packaging Wizard - A tool to publish applications to the Mobile Server
- Mobile Sync - A transactional synchronization engine
- msqll - An interactive tool that enables you to create, access, and manipulate Oracle Database Lite on laptops and handheld devices.

Using any C or C++ 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 10g Installation and Configuration Guide for Windows NT/2000/XP* 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 (msqll.exe), are described in the *Oracle Database Lite Tools and Utilities Guide*. Mobile SQL is written in Java. It requires the Java runtime environment JRE 1.3 or higher to be installed on your system before you can use it. If you have installed JDK 1.3 or higher, the JRE is already installed in your machine.
- Mobile Sync, the executable (msync.exe), the COM interface, and the Java wrapper for it.
- Packaging Wizard (wtgpack.exe)
- ODBC data source administrator (odbcad32.exe) that you can use to create ODBC data sources (DSN).

The Examples directory <Oracle_home>\Mobile\Sdk\Examples directory contains some sample applications. [Chapter 2, "The Oracle Database Lite RDBMS"](#), [Chapter 2.9, "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 10g 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
```

1.3.1 Mobile SQL (MSQL)

MSQL is an interactive tool that allows you to create, access, and manipulate Oracle Database Lite on laptops and handheld devices. 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

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

1.3.2 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 the *Oracle Database Lite Tools and Utilities Guide*.

1.4 Supported Platforms

Your development environment must include Oracle Database Lite 10g as the encompassing platform. For developing native applications on the Oracle Database Lite 10g 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

1.5 Java Support

For more information, refer [Chapter 5, "Native Application Development", Section 1.5, "Java Support"](#).

1.6 Data Source Name

For more information, refer [Chapter 5, "Native Application Development", Section 1.6, "Data Source Name"](#).

The Oracle Database Lite RDBMS

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

- [Section 2.1, "Introduction"](#)
- [Section 2.2, "Development Interfaces"](#)
- [Section 2.3, "Using the Starter Database"](#)
- [Section 2.4, "Working With Your Database"](#)
- [Section 2.5, "Creating Multiple Users"](#)
- [Section 2.6, "Oracle Database Lite Transaction Support"](#)
- [Section 2.8, "Creating Snapshot Definitions"](#)
- [Section 2.9, "Using Oracle Database Lite Samples"](#)
- [Section 2.10, "Limitations"](#)
- [Section 2.11, "Tracing"](#)

2.1 Introduction

The Oracle Database Lite RDBMS is a small footprint, administration free, object-relational database management system that supports ODBC, JDBC, and SODA interfaces. SODA provides access to SQL as well as object-oriented functionality. The Oracle Database Lite RDBMS supports SQL92 language with some extensions. It is designed to be used as a local RDBMS for mobile clients. Data stored in the Oracle Database Lite database can be synchronized with the data stored in Oracle server databases, such as Oracle9i.

2.2 Development Interfaces

This section provides an overview of the development interface. Topics include:

- [Section 2.2.1, "Development Interface Overview"](#)
- [Section 2.2.2, "Mobile Sync Client Module Application Programming Interfaces \(APIs\)"](#)
- [Section 2.2.3, "Oracle Database Lite Load APIs"](#)
- [Section 2.2.4, "Oracle Database Lite Load Utility \(OLLOAD\)"](#)
- [Section 2.2.5, "ADO.NET"](#)

2.2.1 Development Interface Overview

Oracle Database Lite provides the following interfaces for developing database applications:

- For relational database development:
 - JDBC
 - ODBC
- For object and relational database development:
 - Simple Object Data Access (SODA)

Any interface that supports ODBC or JDBC data sources, such as ADO can also be used to access Oracle Database Lite. The interfaces can be used either independently or in combination.

2.2.1.1 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 (Java Developer's Kit) core, provides an object interface to relational databases. Oracle Database Lite supports JDBC through an Oracle Database Lite Type 2 and Type 4 (for multi user version only) JDBC drivers that interpret the JDBC calls and pass them to Oracle Database Lite.

The following section describes how to start a multi user Oracle Database Lite service.

2.2.1.2 Starting a Multi User Oracle Database Lite Database Service

Oracle Database Lite 10g provides a means to install, start, and stop an Oracle Database Lite multi user database service. Once started, you can manipulate the local databases from any machine on the local network. The Branch Office infrastructure demonstrates the use of a multi user database service. For more information on the client/server computing architecture, refer to Section 15.3 "Architecture" in the *Oracle Database Lite Administration and Deployment Guide*.

The following section describes how to install, configure, start, debug, create DSNs, access the database, and verify the database connection using `msql` for a multi user database service.

2.2.1.3 Accessing the Multi User Oracle Database Lite 10g Database Service

This section describes how to install and configure the multi user Oracle Database Lite database service. Topics include:

- [Installation and Configuration](#)
- [Starting the Service](#)
- [Debugging the Service](#)
- [Creating DSNs](#)
- [Accessing the Database](#)
- [Verifying the Connection Using msql](#)

Installation and Configuration

To install and configure the multiuser Oracle Database Lite 10g database service, perform the following steps.

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

<Oracle_Lite_home>\Mobile\Sdk\bin

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

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

olsv2040.exe/install

3. If you have *JDK* 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
- <Oracle_Lite_home>\Mobile\Sdk\bin

For example, the <Jdk_home> directory could be C:\jdk1.3.1_05. Ensure that you use *JDK 1.3.1* variants only.

4. If you have *JRE* installed on your PC, ensure that the system PATH variable includes the following:

- <jre_home>\bin
- <jre_home>\bin\hotspot
- <Oracle_Lite_home>\Mobile\Sdk\bin

For example, the <Jre_home> directory could be C:\Program Files\JavaSoft\JRE\1.3.1_05. Please use *JDK 1.3* variants only.

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.

5. Ensure that your system CLASSPATH variable includes the following:

<Oracle_Lite_home>\Mobile\Classes\Olite40.jar and '.'

6. You may change the startup type from the Windows NT service console. Highlight the Oracle Database Lite 10g 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.
7. Ensure that the `SuggestedSharedAddress` and `SharedAddress` parameters are not present in the `polite.ini` file.
8. After changing the above mentioned variables, reboot your PC.

Starting the Service

The Multi User Oracle Database Lite Database Service can be started in many ways. If the service property "Startup Type" is automatic in the `polite.ini` file, the multiuser service is started every time you reboot the machine.

Using the Command Prompt, you can start the multiuser service by entering any one of the following startup commands.

- `net start "Oracle Lite Multiuser Service"`
- `net start "Oracle Lite Multiuser Service" /wdir=<a_working_directory> /port=nnn`

If you use ' . ' in SQL scripts that load Java classes, you must specify a working directory. The port parameter defaults to 100.

To stop the service, use the following command.

```
net stop "Oracle Lite Multiuser Service"
```

Debugging the Service

If the service does not start, you can debug the service using the following method.

1. Edit the `polite.ini` file which is available under `%WINDIR%\polite.ini` to add the entry `SvTrace=on` under the `[ALL_DATABASES]` section. The information in this file is not case-sensitive.
2. Start the Command Prompt and enter the following.

```
olsv2040/debug/port/=nnn
```

The port parameter in the above command is optional.
3. Should the service fail, the multiuser service generates a log file named `olsv.log` in the current working directory. Ensure that the `PATH` and `CLASSPATH` variables are accurate.
4. Correct the cause and retry.

Creating DSNs

To access the database using an ODBC or VB application, you must create the DSN differently enabled from the embedded connection. When you 'Add' a DSN using the ODBC administration tool, choose the **Oracle Lite 40 ODBC Driver(Client)**. In this way, you will create a client DSN. If you are running the service on the same PC where the client application is running, you can 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 100, you must specify the Database Port Number.

Accessing the Database

To access the database, you need not make any changes to the ODBC or VB application. The DSN automatically routes the request to the client via the ODBC driver `olc12040.dll`. For a JDBC application, you must change the URL for the connect string. The URL syntax is documented in the *Oracle Database Lite Developer's Guide for Java*. It is similar to the one used while connecting to the database using `msql`. For more information, refer to Section 3.3, "Connect to Oracle Database Lite," in the *Oracle Database Lite Developer's Guide for Java*.

Verifying the Connection Using msql

Using the Command Prompt, you can verify the connection to the multiuser service in the following ways.

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

The above command connects to a-dsn on the local host or port 100.

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

The above command connects to a-dsn on a local host on port 1000.

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

The above command connects to a-dsn on a local host on port 100 using the Type4 JDBC driver.

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. The Type2 JDBC driver talks to the remote ODBC driver (oci12040.dll) using a native ocijdbc40.dll (JNI Implementation).

For more information on JDBC and Oracle Database Lite, see the *Oracle Database Lite Developer's Guide for Java*.

2.2.1.4 ODBC

Microsoft's Open Database Connectivity (ODBC) interface is a procedural, call-level interface for accessing SQL databases, and is supported by most database vendors. It specifies a set of functions that allow applications to connect to databases, 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 that interpret the ODBC calls and pass them to Oracle Database Lite.

For more information on ODBC, see the following:

- Microsoft's ODBC documentation.
- The Oracle Database Lite ODBC sample application. For its location in this document, see [Section 2.9, "Using Oracle Database Lite Samples"](#).
- The *Oracle Database Lite Tools and Utilities Guide*.
- Section 4.4.2.1, "Returning Multiple Rows in ODBC", in the *Oracle Database Lite Developer's Guide for Java*.

2.2.1.5 SODA

SODA is a comprehensive and easy 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 roughly 3 times faster than ODBC for simple operations. It allows rich datatypes such as arrays and object pointers in addition to standard SQL columns. A programmer now has an option to just store any data structure in the database and not worry about relational design or doing joins.

On the other hand, a C++ developer can also use an interface that is similar to JDBC for executing SQL statements when necessary. The resulting code is much shorter and clearer than its ODBC equivalent. SQL queries can optionally return objects that can be examined and modified directly through the object-oriented layer, without calling any additional SQL statements.

Finally, object-relational mapping allows 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 database server.

2.2.2 Mobile Sync Client Module Application Programming Interfaces (APIs)

These APIs allow the application to programmatically control the data synchronization process. The application invokes the functions in the Mobile Sync APIs to initiate the data synchronization process and capture error messages generated by the Mobile Sync APIs. For more information on the Mobile Sync APIs please see [Chapter 5, "Native Application Development"](#), [Section 5.4, "Mobile Sync Application Programming Interfaces \(APIs\)"](#).

2.2.3 Oracle Database Lite Load 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 B, "Oracle Database Lite Load Application Programming Interfaces \(APIs\)"](#).

2.2.4 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 OLOAD see the *Oracle Database Lite Tools and Utilities Guide*.

2.2.5 ADO.NET

The Oracle Database Lite ADO.NET Provider implements Microsoft's ADO.NET specification. Developers can use this programming interface to access Oracle Database Lite and trigger Data Synchronization in their .NET based applications. The Oracle Database Lite ADO.NET data provider supports both .NET and Compact .NET frameworks.

2.3 Using the Starter Database

When you install the Mobile Development Kit, an ODBC data source name (DSN) POLITE, and a starter database called **POLITE.ODB** are created. The location of new database for the DSN POLITE is set to <Oracle_home>\Mobile\Sdk\oldb40.

A default user named SYSTEM is set up for you during installation of the samples. SYSTEM contains all database privileges and has a no password. You can create a password for SYSTEM by using the ALTER USER command. (The following section describes sample syntax.) You can either use the default user name or establish user names of your own.

Note: Review the *Oracle Database Lite SQL Reference* before using the starter database. This reference describes the Structured Query Language (SQL) used to manage information in Oracle Database Lite.

You can connect to the Oracle Database Lite starter database using an application such as Mobile SQL. Mobile SQL is a command line interface. To connect to the POLITE database, use the following command from the Command Prompt.

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

You can assign SYSTEM a password by entering the following command.

```
SQL> ALTER USER SYSTEM IDENTIFIED BY <password>
```

Note: For more information, see [Section 2.5.4, "Changing Passwords"](#).

When connecting to the starter database from an ODBC application, use the default ODBC DSN POLITE.

2.4 Working With Your Database

This section provides an overview of working with your Oracle Database Lite, including creating a database, connecting to a database, creating users, and administering the database.

2.4.1 Creating a New Database

When you create a new database using the POLITE data source name, the new database file is located in the <Oracle_home>\Monile\Sdk\olddb40 directory. For ease of maintenance, it is recommended that you use one database directory for all databases.

Note: All newly created databases contain the user SYSTEM, which has a NULL password.

You can create a new data source name using the ODBC Administrator. For more information, refer the following section.

2.4.2 Creating a Data Source Name with ODBC Administrator

The ODBC Administrator is a tool provided by Microsoft to manage the **ODBC.INI** file and associated registry entries in Windows 98/NT/2000/XP. It allows you to add a data source name and specify the database file you want to dedicate as the default for the data source name. For more information on the ODBC Administrator, and for instructions on creating a data source name using the tool, refer to Section 3.7, "ODBC Administrator and the Oracle Database Lite ODBC Driver," in the *Oracle Database Lite Tools and Utilities Guide*.

2.4.3 Creating a New Database Using the Command-Line Utility

To create a new database from the command line, use the CREATEDB utility. The syntax is:

```
CREATEDB mydsn mydbname
```

For example:

```
CREATEDB polite newdb
```

where mydsn is the DSN name and mydbname is the new database name.

See the *Oracle Database Lite Tools and Utilities Guide* for more information on CREATEDB.

2.4.4 Connecting to a New Database

To connect to a new database using Mobile SQL (MSQL), connect as the user named `SYSTEM`, with the password `MANAGER` and the data source name. For example:

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

You can replace `mydsn` with a previously defined ODBC data source name.

2.5 Creating Multiple Users

You can create multiple users in Oracle Database Lite by using the `CREATE USER` command. 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 and is not assigned a password.

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

2.5.1 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 2–1 Pre-Defined Roles

Role Name	Privileges Granted To Role
ADMIN	Enables the user to create other users and grant privileges other than <code>DBA</code> and <code>ADMIN</code> 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 <code>SYSTEM</code> : All <code>ADMIN</code> 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 2–1 (Cont.) Pre-Defined Roles

Role Name	Privileges Granted To Role
RESOURCE	<p>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:</p> <p>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.</p>

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.

2.5.2 Creating Users

You can create users if you are connected to the database as "system", or if you are granted the ADMIN or DBA role. To create a user, issue the following statement:

```
CREATE USER <user> IDENTIFIED BY <password>
```

Here, <user> is a unique user name with up to 128 characters, beginning with a letter, and <password> is a string of up to 128 characters. This statement creates a schema with the user name and assigns the schema as the default schema for the user.

For encrypted databases, all user names and passwords are written to a file named **mydbname.opw**. Each user can then use their own 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 and the .plg file.

2.5.3 Dropping Users

You can drop users if you are connected to the database as "system", or if you are granted the ADMIN or DBA role.

To drop a user when the user's schema does not contain any objects, use the syntax:

```
DROP USER <user>
```

To drop all objects in the user's schema before dropping the user, use the syntax:

```
DROP USER <user> CASCADE
```

For more information on the DROP USER command, see the *Oracle Database Lite SQL Reference*.

2.5.4 Changing Passwords

You can change a user's password if you meet one of the following conditions:

- You are connected to the database as that user
- You are connected to the database as SYSTEM
- You are granted the ADMIN or DBA role

To change a user's password, issue the following statement:

```
ALTER USER <user> IDENTIFIED BY <password>
```

2.5.5 Granting Roles

You can grant the ADMIN or DBA roles to users by issuing the following statement:

```
GRANT <role> TO <user_list>
```

Here, <user_list> is either one user or a comma separated list of multiple users.

2.5.6 Granting Privileges

You can grant privileges on a database object to users by issuing the following statement:

```
GRANT <privilege_list> ON <object_name> TO <user_list>
```

Here, <privilege_list> is either a comma separated list of the following privileges or a combination called ALL:

- ALL
- INSERT
- DELETE
- UPDATE (column_list)
- SELECT

Object_name is a table name prefixed with a schema name.

If <privilege_list> is ALL, then the user can INSERT, DELETE, UPDATE or SELECT from the table or view. If <privilege_list> is either INSERT, DELETE, UPDATE, or SELECT, then the user has that privilege on a table.

2.5.7 Revoking Roles

You can revoke user roles by issuing the following statement:

```
REVOKE <role> FROM <user_list>
```

2.5.8 Revoking Privileges

You can revoke privileges on database objects from users by issuing the following statement:

```
REVOKE <privilege_list> ON <table_name> FROM <user_list>
```

2.5.9 Building 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 Database Lite default starter database (POLITE.ODB).

2.5.10 Populate Your Database Using Mobile SQL

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>\DBS\Poldemo.sql
```

You can also enter:

```
SQL> START <filename>
```

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

2.5.11 Backing Up a Database

The Oracle Database Lite occupies one file, and has dependent log files which can be backed up by copying to another location. Before any files can be copied, however, your database administrator must shut down the database which ensures that log file changes are applied to the database. Once that has been accomplished, you can copy the `*.odb`, `*.opw`, and `*.plg` files to another directory to make a backup of the database.

2.5.12 Encrypting and Decrypting a Database

Two utilities, `ENCRYPDB` and `DECRYPDB`, enable you to encrypt and decrypt Oracle Database Lite databases. These utilities enable you to encrypt an Oracle Database Lite database with a password. The password can be used to prevent unauthorized access to the database and also to encrypt the database so that the data stored in the database files cannot be interpreted by examining the files. The password is used to derive a 40-bit encryption key. Oracle Database Lite uses a version of the Data Encryption Standard (DES) algorithm known as CAST5. A new database created in 10g uses the Advanced Encryption Standard (AES) encryption. Oracle Database Lite will continue to support CAST5 for previous databases.

See the *Oracle Database Lite Tools and Utilities Guide* for more information about these utilities.

2.6 Oracle Database Lite Transaction Support

When an application connects to 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.

2.6.1 Atomicity

A transaction is a sequence of database operations, such as `SELECT`, `UPDATE`, `DELETE`, and `INSERT`. All operations either succeed and are committed or are rolled back. This is called the atomicity property of a transaction.

Oracle Database Lite implements atomicity by not updating the actual database file until a database commit. During commit, a temporary undo log is created and then the database file is updated. If an event, such as a power outage, interrupts commit, the database is restored from the log during the next connection.

2.6.2 Consistency

Transactions preserve database consistency. A transaction transforms a consistent state of the database into another consistent state, without necessarily preserving consistency at all intermediate points. Oracle Database Lite does not permit a transaction to commit if it violates a constraint and would therefore violate consistency.

2.6.3 Isolation

Transactions are isolated from one another. Even though many transactions run concurrently, a given transaction's updates are concealed from other transactions until the transaction commits. Oracle Database Lite supports the isolation levels for transactions listed in [Table 2–2](#):

Table 2–2 Isolation Levels

Isolation Level	Description
Read Committed	<p>In Oracle Database Lite, a <code>READ COMMITTED</code> transaction first acquires a temporary database level read lock, materializes 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 <code>commit</code>), during this time. Since a commit operation locks the database in "intent" exclusive mode, a read committed transaction, while materializing the query result, will block another transaction that is trying to commit or vice versa. A <code>READ COMMITTED</code> transaction gives 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 (<code>SELECT</code> statement) may return more or less number of rows based on other transactions made to the data in the result set of the query.</p> <p>Note: A <code>SELECT</code> statement containing the <code>FOR UPDATE</code> clause is always executed as if it is running in a <code>REPEATABLE READ</code> isolation level.</p> <p>In Oracle Database Lite, a <code>SELECT</code> statement can execute Java stored procedures. If the transaction executing the Java stored procedure is in the <code>READ COMMITTED</code> isolation level and the Java stored procedure updates the database, then the <code>SELECT</code> statement to execute the Java stored procedure must have a <code>FOR UPDATE</code> clause. Otherwise, Oracle Database Lite issues an error.</p>
Repeatable Read	<p>In this isolation level, a query acquires read locks on all its returned rows. More rows may be read locked because of the complexity of the query itself, the indexes defined on its tables, or because of the execution plan chosen by the query optimizer. The <code>REPEATABLE READ</code> isolation level provides less concurrency than a <code>READ COMMITTED</code> isolation level transaction because the locks are held until the end of the transaction.</p> <p>A "Phantom" read is possible in this isolation level. this happens when another transaction inserts rows that meet the search criteria of the current query and the transaction re-executes the query.</p> <p>If a <code>FOR UPDATE</code> 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 <code>READ COMMITTED</code> 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.</p>

Table 2–2 (Cont.) Isolation Levels

Isolation Level	Description
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, specifically, for the terms "Dirty Read", "Nonrepeatable Read", and "Phantom", which define transaction isolation levels.

2.6.3.1 Durability

Transactions are guaranteed to be durable. That is, once a transaction commits, all its changes are persistent in the database file even if the system subsequently fails at any point. If a transaction fails during a commit or rollback due to some system failure, the undo log file is required to restore the database to a consistent state.

2.6.3.2 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. Different transactions can read the same row, which is read locked. However, a write locked row cannot be accessed by another transaction.

2.6.3.3 Changing the Default Isolation Level

In Oracle Database Lite, the `READ COMMITTED` isolation level is the default.

You can change the default isolation level for a data source name (DSN) by using the ODBC Administrator, or by manually editing the `ODBC.INI` file to include:

```
IsolationLevel = XX
```

where the value for `XX` is `RC` for Read Committed, `RR` for Repeatable Read, `SR` for Serializable, or `SU` for Single User.

Also, you can establish the isolation level of a transaction by using the SQL statement:

```
SET TRANSACTION ISOLATION LEVEL <ISOLATION_LEVEL>;
```

where `ISOLATION_LEVEL` is `READ COMMITTED`, `REPEATABLE READ`, `SERIALIZABLE`, or `SINGLE USER`.

See [Section 2.6.3.4, "Supported Combinations of Isolation Levels and Cursor Types"](#), for more information.

2.6.3.4 Supported Combinations of Isolation Levels and Cursor Types

[Table 2–3](#) shows the supported combinations of isolation levels and cursor types. Isolation levels appear in the left column and cursor types appear in the top row. "S" indicates *supported*, "U" indicates *unsupported*.

Table 2–3 Supported Combinations

Isolation Level	Forward Only	Static	Keyset Driven	Dynamic
Read Committed	S	S	U	U
Repeatable Read	S	U	S	S
Serializable	S	U	S	S
Single User	S	S	S	S

Unsupported combinations generate error messages.

2.6.4 Tuning the Application

Tuning your application design ideally occurs before you begin to implement your application. Before beginning your design, you should carefully read about each of the Oracle Database Lite features available and consider which features best suit your requirements. Also, you should work with your Oracle database administrator to determine how the Oracle master site can be tuned to accommodate your application. Some specific design tips to consider are outlined in [Appendix A, "Optimizing SQL Queries"](#).

2.7 Support for Linguistic Sort

Linguistic sort is a new 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.

2.7.1 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.

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.

Unicode and NLSRT are not supported in the current version of Oracle Database Lite. Therefore, NCHAR data type and customization of collation sequence are not yet available. For more information on how collation sequences are enabled using the file `polite.ini`, refer the *Oracle Database Lite Developers Guide*.

2.7.2 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.

2.7.3 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.

2.7.3.1 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.

2.7.3.2 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'.

2.7.3.3 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.

2.7.3.4 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.

2.7.3.5 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.

2.8 Creating Snapshot Definitions

The data that your offline applications operate on is stored in Oracle Database Lite as either base tables or snapshots. Base tables can be created using the `CREATE TABLE` SQL statement. Base table store data that is independent of the server data; changes made to them are never synchronized with the server database.

Snapshots store a subset of server data. Changes made to a snapshot can be synchronized with the server data. However, snapshots cannot be created in Oracle Database Lite by using SQL statements. Snapshots are created by the Mobile Server as part of the application installation. They are created based on the publication items defined on the Mobile server. A publication items contains a parameterized SQL query that defines the subset of server data that needs to be stored in the snapshot.

In most situations, a table or view already exist on the server from which you will create snapshots for your application to use. The following techniques 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. The options for creating publication items/snapshot definitions are:

1. [Creating a Snapshot Definition Declaratively](#) - Create publication items using the Packaging Wizard. This is the recommended method.
2. [Creating the Snapshot Definition Programmatically](#) - Create a publication item programmatically using the Consolidator API.

2.8.1 Creating a Snapshot Definition Declaratively

This method uses the Packaging Wizard, a GUI based tool of Oracle Database Lite. 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 one.

- Use the Packaging Wizard to define an application and the publication items (snapshot definitions) for it.
- Use the Packaging Wizard to publish the application to the Mobile Server. This will create the publication items associated with the application.
- 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.

Using the Packaging Wizard, as described in the *Oracle Database Lite Tools and Utilities Guide* provides additional details for this approach.

2.8.2 Creating the Snapshot Definition Programmatically

The second way to create a snapshot definition is to use the Consolidator API to programmatically create the publication items on the Mobile Server. While this method is more involved, requiring the knowledge of the Oracle Database Lite 10g application model, it does provide all the features of the product, including creation of publication items from views, customize code to construct snapshots, which is described in [Chapter 3, "Synchronization"](#). The database base tables must exist before the Consolidator 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

Creating Publications

Publications are Mobile Server objects that are used to organize other objects such as publication items, indexes on them, platform specific information, etc., required by an application. You can create publications using the Consolidator API. You can call the functions in these APIs from within Java programs as standard function calls.

Creating Publication Items

A publication item is a Mobile Server object that contains the SQL select statement that specifies which data subset of the parent table or view or synonym is replicated on the client. A publication item usually corresponds to a snapshot on the client device. You can create publication items using the Consolidator API. You can call the functions in this API from within Java programs as standard function calls.

Creating Users

Each client is identified by a user ID. For development purposes, a user must be created using the Consolidator API in order to assign data subscriptions to a particular user.

Creating Subscriptions

A subscription is a Mobile Server object that relates a user to a publication. You can create subscriptions using the Consolidator API. Before a subscription can be used to create a client database, every parameter of the publication must be given a value. You can assign a value to each parameter using the `SetSubscriptionParameter` method of

the Consolidator API. You can call the functions in this API from within Java programs as standard function calls. To create publications and subscriptions using Java, see [Section 3.4, "The Publish and Subscribe Model and Oracle Database Lite Synchronization"](#), in [Chapter 3, "Synchronization"](#).

2.9 Using Oracle Database Lite Samples

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

2.9.1 Overview

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	Demonstrates the use of the Oracle BLOB datatype and Visual Basic's ODBC programming methods and object manipulation. See Section 2.9.2, "BLOB Manager Example Notes" for more information.
Java	<Oracle_home>\Mobile\Sdk	Demonstrates programming with JDBC. See the <i>Oracle Database Lite Developer's Guide for Java</i> for more information.
ODBC	<Oracle_home>\Mobile\Sdk	Provides ODBC programs written in C.
Visual Basic	<Oracle_home>\Mobile\Sdk	Demonstrates the ease of querying tables in Oracle Database Lite with Visual Basic tools. See Section 2.9.3, "Running the Visual Basic Sample Application" for more information.

Note: Most examples use the data source name (DSN) POLITE. If you need to drop and recreate, use the REMOVEDB and CREATEDB utilities.

2.9.2 BLOB Manager Example Notes

To install the BLOB Manager example, open the \SETUP folder in <Oracle_home>\Lite\Sdk and run **setup.exe**. After you complete the installation, click the 'Start' button and select 'BLOB Manager' from the 'Programs' menu.

To uninstall the example, click the 'Start' button, select 'Settings', and then 'Control Panel'. Select 'Add/Remove'. Select 'BLOB Manager' and click the 'Add/Remove' button.

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 as stated above before you open the Visual Basic project file and run it with Visual Basic. Running the program from the Programs menu will prepare the table in the database for you automatically.

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

2.9.3 Running the Visual Basic Sample Application

This example (which uses Visual Basic 5.0 or higher) 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.

These instructions for installing and running the Visual Basic sample application assume that you have already installed Oracle Database Lite and Visual Basic (version 5.0 or higher).

Note: If you have not installed Visual Basic and the ODBC drivers, you need to install them before you begin.

2.9.3.1 Open Visual Basic

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

2.9.3.2 View the Sample Application Tables and Data

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

1. From the Add-Ins menu, select Visual Data Manager. In the VisData window, select Open Database from the File menu and select ODBC.
2. 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

3. 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.

2.9.3.3 Open the Sample Application

1. To open the sample application, select Open Project from the File menu. In the dialog box, navigate to your <Oracle_home>\Mobile\Sdk\Examples\VB directory. Select **update.mak**, and click Open.

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

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

2.9.3.4 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.

2.9.4 ODBC Examples

These examples are located in <Oracle_home>\Mobile\Sdk.

These examples must be compiled using a C++ compiler. To build them, open a console, switch to the <Oracle_home>\Mobile\Sdk directory and type "nmake".

There are 5 odbc examples namely, odbctbl, odbcvview, odbcfunc, odbctype, and long. You only need the POLITE data source name (DSN) to run these examples. The POLITE DSN is automatically created during the Mobile Development Kit installation.

To run the examples, execute **run.bat** in the <Oracle_home>\Mobile\Sdk directory. The first four examples have their own output windows showing the log of what is done. Closing the current example window causes the next example to be run. The output displayed in the example windows is also printed in the log files, **odbctbl.log**, **odbcvview.log**, **odbcfunc.log**, **odbctype.log**. The long example output is collected in the output file **long.out**.

2.9.4.1 What the Examples Do

The following sections describe the functionality of the samples found in <Oracle_home>\Mobile\Sdk.

2.9.4.1.1 odbctbl This is an ODBC SQL Table example. It shows you how to manipulate tables using ODBC API. It creates table EMP with columns ID, NAME, START_DATE, SALARY, populates this table with the data, does 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.

2.9.4.1.2 odbcvview This is an ODBC SQL View example. It shows you how to manipulate views using the ODBC API. It creates table EMP (as above) and view HIGH_PAID_EMP selecting the full name (using the CONCAT scalar function), HIRE_DATE and SALARY from the EMP table. Then EMP is populated. After that a select is performed from the HIGH_PAID_EMP view is issued to see the populated data. Then the salary column of EMP is updated, some rows are deleted from EMP, and again the

select from HIGH_PAID_EMP is issued to see how those changes are reflected in the view. Finally, the view and the table are dropped.

2.9.4.1.3 odbcfunc This is an ODBC SQL Scalar Functions example. It shows you how to use scalar functions in the ODBC API. It creates table EMP, populates it with the data, then does select ID, FULL_NAME from EMP, where to calculate full name it uses odbcf scalar function CONCAT with last and first names as arguments. Then it updates the table converting last name to uppercase and first name to lowercase for IDs < 3 using odbcf scalar functions UCASE and LCASE. The new data is selected and displayed again. At the end the table EMP is dropped.

2.9.4.1.4 odbctype This is ODBC SQL Types Example. It shows you how to manipulate different data types using ODBC API. This test just creates table EMP, populates it with data, selects all the rows and displays the result, but 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 get all the information about that column, such as column name, column name length, column type, column length, column scale, etc. This information is then used to bind the column. This shows how you can get the type information from the database using the ODBC API.

2.9.4.1.5 long This example exercises the basic read/write functions of SQL LONG VARCHAR. It first drops, then creates the table LONG_DATA 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 the select from the table is issued 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 file "long.out".

2.10 Limitations

Currently, the Oracle Database Lite engine has a limitation of not being able to sort any row that exceeds 4040 bytes in length. The selected columns exceed 4040 bytes and the database engine issues this error. Therefore, queries that use the UNION operation that are implemented by sorting the intermediate results from the two select clauses in the query cannot be fixed.

2.11 Tracing

The Oracle Database Lite 10g 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 code written by users, other Oracle tools such as forms, SQLJ, OC4J or in the Oracle Database Lite 10g database component. Errors also occur in simple environments where a user application talks directly to the Oracle Database Lite 10g database through JDBC or ODBC drivers. At first glance, 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.

This section describes how to set the Tracing feature. Topics include:

- [Section 2.11.1, "Enabling Trace Output"](#)
- [Section 2.11.2, "Basic Functions"](#)
- [Section 2.11.3, "SQL Tracing"](#)

2.11.1 Enabling Trace Output

To enable Trace output, perform the following.

Include the following line in the **polite.ini** configuration file.

```
OLITE_SQL_TRACE= yes
```

The parameter name and the value string "yes" are not case sensitive. For example, the following line also enables trace output.

```
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.

2.11.2 Basic Functions

The Tracing facility can be enabled through a configuration parameter in the **polite.ini** file. On enabling the trace feature, the information generated is dumped to a trace file named **olddb_trc.txt** in the current working directory of the database process. If the file exists, the trace output is added to it. If it does not exist, a new file is created automatically. If the database fails to create or update the file, the tracing feature does not take place. The following information is dumped in a trace file.

1. Each time a SQL statement is prepared, its text is dumped into the trace file. The text begins with a header titled **Statement Text**.
2. After the SQL statement is compiled, the compilation time is printed in one line titled **Compilation Time**.
3. 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.
4. If a SQL statement contains markers, then the bind value is printed for every line.
5. Each time a temporary table is created, its name is dumped into the trace file. The text begins with a header titled **Temporary Table Created**.
6. Each time a table is accessed, the following information is dumped into the trace file:
 1. Table Name: The name of the table been accessed is dumped into the trace file. The text begins with a header titled **Table Name**.
 2. Access Method: The access method used by the database is dumped into the trace file. The text begins with a header titled **Access Method**.
7. Each time a temporary table is sorted, its name and sorting time are dumped into the trace file. The text begins with a header titled **Temporary Table Sorted**

followed by the sorted temporary table name and the time it takes (in milliseconds) to sort the table.

8. If the SQL statement is a SELECT statement, the time spent on fetching the first row is dumped into the trace file. The text begins with a header titled **First Fetch Time**.
9. The thread ID is dumped into the trace file in front of some of the dumped information. The **Tid** is the title used to represent the Thread ID.

2.11.3 SQL Tracing

SQL trace output is dumped to a trace file named **olddb_trc.txt** in the current working directory of the database process. For a database service on Windows, Windows NT or the Oracle Database Lite daemon for a Linux platform, the current working directory is specified by the **wdir** parameter during startup of the database service or daemon. To implement the Tracing feature, the database process must contain permissions to create the trace file in the current working directory. The Trace output is always included in the trace file. If the trace file does not exist, it is created automatically.

The SQL trace facility dumps the following information to the trace file.

1. The thread ID.
2. SQL statements after compilation.
3. Compilation time including optimization.
4. Value of marker as it exists just before execution of the SQL statement.
5. Execution plan as described in the `EXPLAIN PLAN` statement in the *Oracle Database Lite SQL Reference*.
6. The name of the temporary table created.
7. The name of the table being accessed and the access method used.
8. The name of the temporary table been sorted and the sorting time.
9. The time spent on fetching the first row if the SQL statement is a SELECT statement.

2.11.3.1 The Tid Output

The thread ID of the running operation is printed in front of some of the dumped information. The thread is displayed in the following format:

Tid: <thread id>

2.11.3.2 SQL Statement Output

Each SQL statement is preceded by the prefix **Statement Text**. 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.

2.11.3.3 Compilation Time Output

After the SQL statement is compiled, the compilation time is printed in one line. This line begins with the title **Compilation Time**.

2.11.3.4 Bind Values Output

The value of markers or bind variables is one per line. This line is displayed in the following format.

Marker [<number>]: <Value>

Where, <number> is the number of the marker and <value> denotes the value of the marker just before execution.

2.11.3.5 Explain Plan Output

This output is printed in the same format as printed by the `EXPLAIN PLAN SQL` statement.

2.11.3.6 Temporary Table Created Output

The name of the temporary table created is printed if a temporary table is created by the database system.

2.11.3.7 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.

2.11.3.8 Temporary Table Sorted Output

The name of the temporary table sorted and the time it takes to sort the table.

2.11.3.9 First Fetch Time Output

The time the database takes to retrieve the first row when performing a `SELECT` operation. The "First Fetch Time" is the time to retrieve the first row in the result set.

Synchronization

This document describes how synchronization functions between Oracle Database Lite and an Oracle database using the Mobile Server and the Mobile Sync client application. It also discusses the Consolidator, including the publish and subscribe model, the use of the Consolidator and Resource Manager APIs to customize applications, and the advanced features for customizing the Consolidator, among other topics. The topics that are discussed in this document are the following:

- [Section 3.1, "Overview"](#)
- [Section 3.2, "Synchronization Process"](#)
- [Section 3.3, "Mobile Sync Application Programming Interfaces \(APIs\)"](#)
- [Section 3.4, "The Publish and Subscribe Model and Oracle Database Lite Synchronization"](#)
- [Section 3.5, "Using Consolidator to Define the Sample11.java Example"](#)
- [Section 3.6, "Other Standard Consolidator Functionality"](#)
- [Section 3.7, "Advanced Features for Customizing Consolidator"](#)
- [Section 3.8, "Synchronization Errors and Conflicts"](#)
- [Section 3.9, "Mapping Datatypes Between the Oracle Server and Clients"](#)

3.1 Overview

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

There are basically three types of publication items that can be used to define synchronization; fast refresh, complete refresh, and queue based.

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, a background process called the Message Generator and Processor (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.

Another method of synchronization is the complete refresh publication item. 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.

Lastly, there is the queue based publication item. 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.

3.1.1 Synchronization Concepts

Data is synchronized between Oracle Database Lite and an Oracle database server. This is accomplished by invoking the Mobile Sync client which interacts with the Mobile Server. The Mobile Server uses synchronization objects such as users, publications, publication items, and subscriptions to process client and server data changes. This is often referred to as the publish and subscribe model.

This section describes the following synchronization concepts. Topics include:

- [Section 3.1.1.1, "Publication Item"](#)
- [Section 3.1.1.2, "Publication"](#)
- [Section 3.1.1.3, "Application"](#)
- [Section 3.1.1.4, "Subscription"](#)
- [Section 3.1.1.5, "Data Subsetting"](#)
- [Section 3.1.1.6, "Shared Maps"](#)

3.1.1.1 Publication Item

A publication item is a Mobile Server object that has a unique name and contains a SQL query that is defined against an Oracle database table, view, or a synonym. The query in the publication item can have optional parameters, known as subscription parameters or template variables, which are used to determine what subset of the data of the table, view, or synonym is synchronized for each user.

3.1.1.2 Publication

A publication is a Mobile Server object that has a unique name and serves as a container of publication items. A publication may contain zero or more publication items, and a publication item may be contained in zero or more publications. A publication keeps track of all the subscription parameters used in the member publication items. A publication also contains indexes defined on publication items as well as platform specific information such as the type of database to be created on the client.

3.1.1.3 Application

Every Oracle Database Lite application has an associated publication that defines the data needed by the application.

3.1.1.4 Subscription

A subscription relates a publication to a user. A subscription cannot be used unless and until every parameter of the publication is initialized to a value. When a user synchronizes with the Mobile Server, an Oracle Database Lite is created for each subscription. Each publication item of the publication becomes a snapshot in this database.

3.1.1.5 Data Subsetting

Through established subscriptions, the Mobile Server prepares any new data for each client which is then downloaded when the client synchronizes. Only the required subset of data is downloaded to each client. If the publication has been flagged for complete refresh, all the qualifying data is downloaded.

3.1.1.6 Shared Maps

Shared maps save space on the server by improving the scalability of replication for multiple users sharing subscription data sets. This feature, which is turned on by default, reduces the size of the map tables for large lookup publication items. When multiple users share the same data, usually their query subsetting parameters are identical.

3.1.2 Synchronization Example

The following steps take you through the components and procedures necessary to perform a synchronization. These steps assume you are installing the client on a Windows system. By completing the steps listed, you will be able to synchronize every time. Steps 1 and 2 may require the assistance of your administrator.

You must download and configure the **msync.exe** client, and use it to create a local Oracle Database Lite for a sample user named "S11U1." This user exists as part of the samples installed.

1. Install and configure an instance of the Mobile Server as described in the *Oracle9i Lite Installation and Configuration Guide for Windows NT/2000/XP*.
2. From the command line on the Mobile Server system, run **instdemo.bat** (**instdemo** on Solaris) to create sample applications in the Mobile Server repository.

On Solaris

You can specify the following path to create and store sample applications in the Mobile Server repository.

```
<Oracle_home>\Mobile\Server\Samples
```

On Windows NT

You can specify the following path to create and store sample applications in the Mobile Server repository.

```
<Oracle_home>\Mobile\Server\Samples
```

Note: Replace <Oracle_home> with your actual Oracle Home directory name.

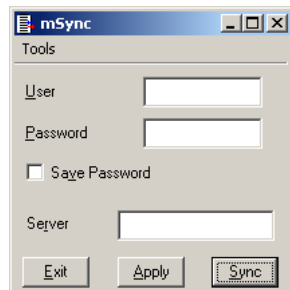
3. Using a browser, install the Mobile Sync application to connect to your Mobile Server instance using the following URL.

`http://<mobile_server>/webtogo/setup`

where <mobile_server> is the hostname of your Mobile Server instance. Click the link which installs the client for "Windows 32" and follow instructions to install the Mobile Sync application. At this stage, you will be prompted for an installation directory. This procedure assumes and recommends you install it in your <Oracle_home> directory.

4. Open your <Oracle_home>\Mobile\Sdk\bin directory and run **msync.exe**.
5. As [Figure 3–1](#) displays, the mSync dialog appears.

Figure 3–1 mSync Dialog



6. As [Table 3–1](#) describes, enter the appropriate parameters in the corresponding fields.

Table 3–1 Mobile Sync Parameters

Field	Value	Description
User	s11u1	Mobile Client user name. This field is not case sensitive.
Password	MANAGER	Mobile Client password. This field is case sensitive.
Save Password	Select	Select this check box to save the password.
Server	Your Mobile Server instance hostname	The Mobile Server IP address or URL.

7. To save this information, click **Apply**.
8. To start synchronizing, click **Sync**.

A progress bar appears to indicate the completion of each synchronization task such as composing, sending, receiving, and processing. The progress bar also displays the duration for completion of each task. If synchronization executes successfully, the message "Sync success" appears. When you see this message, a sample database **orders.odb** is created in the <Oracle_home>\Mobile\oldb40\S11U1 directory on the client system. You can view this database using a SQL viewer such as Mobile SQL. It contains two tables named ORD_MASTER and ORD_DETAIL.

If synchronization fails, the message "Sync Failed" appears. To determine the cause of a failed synchronization, the Mobile Server administrator can view tracing information in the Mobile Server log file.

The preferred way to create synchronization objects such as publications and publication items is a tool called the Packaging Wizard which is included in the Mobile

Development Kit. For more information on the Packaging Wizard, see the *Oracle Database Lite Tools and Utilities Guide*.

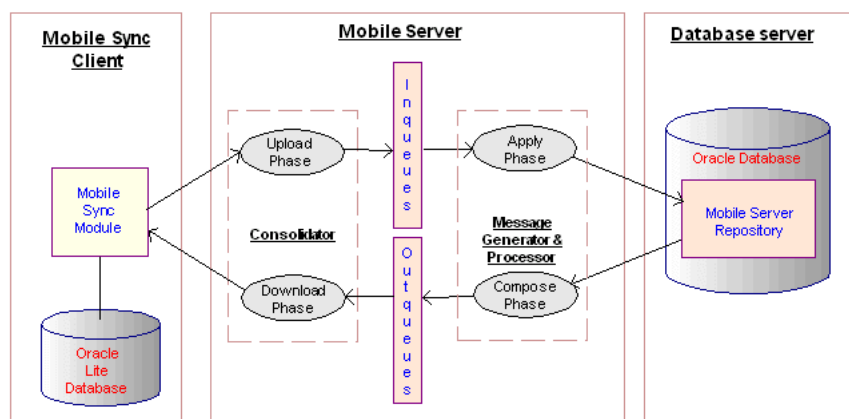
Synchronization objects can also be created programmatically using the Consolidator API and Resource Manager APIs. For more information, see [Section 3.4, "The Publish and Subscribe Model and Oracle Database Lite Synchronization"](#) in [Chapter 3, "Synchronization"](#).

3.2 Synchronization Process

Now that you have performed at least one synchronization, we can look at the synchronization process in more detail. Oracle Database Lite uses an asynchronous method for synchronization between Oracle Database Lite clients and the Oracle database server through the Mobile Server. This means that the Mobile Sync module operates independently of the MGP as neither component is dependent on the other to complete its operation.

[Figure 3–2](#) illustrates the fast refresh synchronization process.

Figure 3–2 Fast Refresh Synchronization



3.2.1 Fast Refresh Synchronization

The default synchronization method is the fast refresh mode as displayed in [Figure 3–2](#). Fast refresh is an incremental refresh where changes are uploaded and stored in queues during the upload phase. Next, the changes which have been stored in out queues are downloaded and applied to the client. Meanwhile, the MGP periodically views the In Queues and takes anything found in an In Queue and applies it to the database during the apply phase. Changes generated by this client, other clients, and server-side applications to the Oracle database are then composed and stored in an out queue until the next time a client is synchronized.

The upload and download phases are performed independently of any apply or compose phase. An apply phase is not dependent on an upload phase, nor is a download phase dependent on a compose phase. During any synchronization session, download occurs after upload, and compose occurs after apply.

A complete refresh is simply an execution of the snapshot query. When application synchronization performance is slow, application developers must tune the snapshot query. Complete refresh items such as publication items are not optimized for

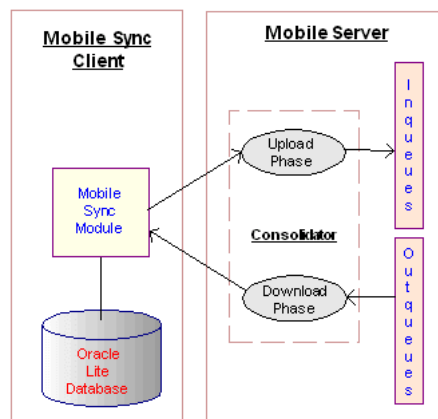
performance. Therefore, to improve performance, application developers can use the fast refresh option. The Consperf utility only analyzes fast refresh publication items.

3.2.1.1 Client Upload and Download Operations

When synchronization is initiated, the client opens a connection to the Mobile Server via the selected mode of transport, which causes the Mobile Server to open a connection to the Oracle database server. This process is illustrated in the following figure.

Figure 3–3 displays the Client Upload and Download phases.

Figure 3–3 Upload/ Download Phases



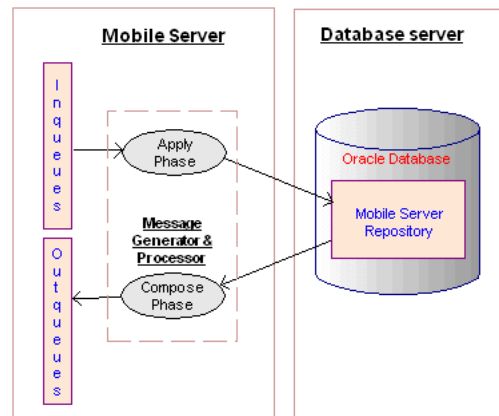
Changes to Oracle Database Lite records are accumulated and flagged with codes for the type of Data Manipulation Language (DML) operation performed such as insert, update, or delete. The data is encrypted, compressed, and sent to the Mobile Server to populate objects called In Queues. An in queue is a persistent database object created to store data temporarily during synchronization.

During the same session, snapshots on the client are updated by applying data from the out queue to the Oracle Database Lite. The difference between an out queue and an in queue is not a table, but a data structure containing a reference to data contained in the Oracle database base tables. For more information on customizing the synchronization process using the In Queue and Out Queues, see [Section 3.7.9, "Queue Interface for Customizing Replication"](#).

3.2.1.2 Mobile Server Apply Operation

For each user, the MGP takes any content of the in queue and applies it to the base tables on the Oracle database. Any conflicts are detected and resolved at this time. For more information, see in [Section 3.8.3, "Resolving Conflicts Using the Error Queue"](#). The apply phase is completed when the changes uploaded by all users are processed.

Figure 3–4 illustrates the Apply and Compose phases in MGP.

Figure 3–4 Apply/Compose Phases

3.2.1.3 Mobile Server Compose Operation

After the apply phase, the MGP reviews the base tables. It composes and stores any changes in the Out Queues to be downloaded to the client.

Note: The Message Generator and Processor (MGP) is a background process which periodically becomes active, looks at the in queues, and applies the changes to the Oracle database base tables. Based on how MGP is configured, there may be a delay in how quickly it composes and readies the out queues to be downloaded to the client regardless of how frequently you synchronize. The changes are stored safely in the in queues until MGP processes them, after which they are downloaded to the client on the next synchronization process.

3.2.2 Complete Refresh Synchronization

During a complete refresh, all contents of the snapshot tables present on the client are refreshed from the Oracle database tables. The MGP is not involved because all the contents are refreshed, but this form of synchronization is time consuming and engages system-resources intensively.

3.2.3 Synchronizing an Encrypted Database

It is possible to encrypt Oracle Database Lite using a utility called `ENCRYPDB`. Synchronizing with an encrypted database requires an understanding of how Oracle Database Lite manages encrypted databases. For more information, see the *Oracle Database Lite Tools and Utilities Guide*.

3.3 Mobile Sync Application Programming Interfaces (APIs)

For a detailed description of Mobile Sync Application Programming Interfaces, refer [Section 5, "Native Application Development"](#).

3.4 The Publish and Subscribe Model and Oracle Database Lite Synchronization

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 most easily using the Packaging Wizard. These functions can also be performed using the Consolidator API and Resource Manager API by writing Java programs to customize the functions as needed. More advanced functionality can only be enabled programmatically using the Consolidator API and Resource Manager API.

The publish and subscribe model uses database objects described in [Table 3–2](#):

Table 3–2 Publish/Subscribe Model Elements

Item	Description
publication item	A publication item is a SQL select statement that specifies which data subset a user can access. A publication item corresponds to a replica table on the client, making a publication item a snapshot definition of a table on the server.
publication	A publication is a group of publication items.
snapshot	A snapshot is a subset of the data in an Oracle database base table which has been defined by the snapshot definition in a publication item.
subscription	A subscription associates a user with a publication and may contain subscription parameters. Subscription parameters are set for all publication items within the publication to which a client is subscribed.
subscription parameter	Subscription parameters use names and string values to define an individual client's subscription to an individual publication. Subscription parameters enable clients to perform data subsetting, and they restrict the number of rows assigned to each client. Typical subscription parameters can include user names and application specific values like employee numbers or area codes.
user	<p>A user is defined by a user name and a password. The Mobile Server synchronizes data according to the client's subscriptions.</p> <ul style="list-style-type: none"> ■ A user can use a single user name to synchronize data from a single client. This is the recommended mode of use. ■ A user can use a single user name to synchronize data stored on multiple devices. When the user changes devices, the Mobile Server performs a complete refresh of all the user's subscriptions on the new device. This technique is not recommended for general use.

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

- Declaratively, using the Packaging Wizard to package and publish applications. This is the recommended method. This method is described fully in the *Oracle Database Lite Tools and Utilities Guide*.
- Programmatically, using the Resource Manager API and the Consolidator API can invoke certain advanced features or customize an implementation. This technique is recommended for advanced users requiring specialized functionality.

3.4.1 The Publish and Subscribe Model Step by Step

The publish and subscribe model can be customized programmatically using the Resource Manager API and the Consolidator API. The basic procedure to invoke Consolidator to implement the publish and subscribe model involves the following steps:

1. Create database tables.
2. Connect to Mobile Server.
3. Create publications.
4. Create publication items.
5. Create publication item indexes as required.
6. Create Users
7. Add publication items to publications.
8. Subscribe users to publications.
9. Define user subscription parameters to publications.
10. Instantiate the subscriptions.

Note: To call the Publish and Subscribe APIs, the following JAR files must be specified in your classpath.

- <Oracle_home>\mobile\classes\consolidator.jar
 - <Oracle_home>\mobile\server\bin\webtogo.jar
 - <Oracle_home>\jdbc\lib\classes12.zip
 - <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
-

3.5 Using Consolidator to Define the Sample11.java Example

To illustrate how these APIs are used to define Consolidator, the following sections use a sample Java program included with Oracle Database Lite 10g, called **sample11.java**. Entries referring to the Resource Manager package are children of the Mobile Admin class found in the Web-to-Go API. Entries referring to the Consolidator class are part of the Consolidator API.

This sample can be found:

On Solaris

<Oracle_home>/mobile/server/samples

On Windows NT

<Oracle_home>\Mobile\Server\Samples

3.5.1 Sample11.java

Here is the source code for the program:

```
import java.sql.SQLException;
import java.sql.*;

import oracle.lite.sync.Consolidator;

public class sample11
{
    static String CONS_SCHEMA;
    static String DEFAULT_PASSWORD;

    public static void main(String argv[]) throws Throwable
    {
        //////////////////////////////////////
        //SAMPLE11
        //////////////////////////////////////
        if(argv.length != 2)
        {
            System.out.println("Syntax: java sample11 <Schema> <Password>");
            return;
        }
        CONS_SCHEMA = argv[0] ;
        DEFAULT_PASSWORD = argv[1] ;

        //Create Required Tables Using Standard JDBC
        DriverManager.registerDriver ((Driver)Class.forName
("oracle.jdbc.driver.OracleDriver").newInstance ());
        Connection c = null;
        Statement s = null;
        try
        {
            c = DriverManager.getConnection ("jdbc:oracle:oci8:@WEBTOGO.WORLD",
"MASTER", "MASTER" );
            s = c.createStatement ();
            s.executeUpdate("create table MASTER.ORD_MASTER("
+ "ID number(9),"
+ "DDATE DATE default TO_DATE('1990-01-01 15:35:40', 'YYYY-MM-DD
HH24:MI:SS'), "
+ "STATUS number(9),"
+ "NAME varchar2(20),"
+ "DESCRIPTION varchar2(20) "
+ ")");

            s.executeUpdate("alter table MASTER.ORD_MASTER add constraint"
+ " orders_pk primary key(ID)");

            s.execute("GRANT ALL ON MASTER.ORD_MASTER to " + CONS_SCHEMA + " WITH GRANT
OPTION");

            s.executeUpdate("create table MASTER.ORD_DETAIL("
+ "ID number(9),"
+ "KEY number(9),"
+ "DDATE DATE default TO_DATE('1995-01-01 15:35:40', 'YYYY-MM-DD
HH24:MI:SS'), "
+ "DESCRIPTION varchar2(20),"
```

```

        + "QTYORDERED number(9), "
        + "QTYSHIPPED number(9), "
        + "QTYRECEIVED number(9), "
        + "COST number(9) "
        + ")");

s.executeUpdate("alter table MASTER.ORD_DETAIL add constraint"
    +" items_pk primary key(ID, KEY)");

        s.execute("GRANT ALL ON MASTER.ORD_DETAIL to " + CONS_SCHEMA + " WITH
GRANT OPTION");
        c.commit ();
    }
    catch (SQLException ee)
    {
        ee.printStackTrace ();
    }
    finally
    {
        if (s!= null) try {s.close ();}catch (SQLException e1){}
        if (c!= null) try {c.close ();}catch (SQLException e2){}
    }

    //Connecting to the Mobile Server
    oracle.mobile.admin.ResourceManager.openConnection(CONS_SCHEMA, DEFAULT_
PASSWORD);
    //Creating Publications
    try {
    Consolidator.DropPublication("T_SAMPLE11");
    } catch (Throwable e) {
    //e.printStackTrace(); ignore error
    }
    Consolidator.CreatePublication("T_SAMPLE11", Consolidator.OKPI_CREATOR_ID,
"OrdersODB.%s", null);

    //Creating Publication Items
    try {
    Consolidator.DropPublicationItem("P_SAMPLE11-M");
    } catch (Throwable e) {
    //e.printStackTrace(); ignore error
    }

    try
    {
        Consolidator.CreatePublicationItem("P_SAMPLE11-M", "MASTER", "ORD_MASTER", "F",
"SELECT * FROM MASTER.ORD_MASTER", null, null);
    } catch (Throwable e) {
    e.printStackTrace();
    }

    try {
    Consolidator.DropPublicationItem("P_SAMPLE11-D");
    } catch (Throwable e) {
    //e.printStackTrace();
    }

    try
    {
        Consolidator.CreatePublicationItem("P_SAMPLE11-D", "MASTER", "ORD_DETAIL", "F",
"SELECT * FROM MASTER.ORD_DETAIL", null, null);

```

```
//Creating Publication Item Indexes

Consolidator.CreatePublicationItemIndex("P_SAMPLE11M-I1", "P_SAMPLE11-M", "I",
"DDATE");
Consolidator.CreatePublicationItemIndex("P_SAMPLE11M-I2", "P_SAMPLE11-M", "I",
"STATUS");
Consolidator.CreatePublicationItemIndex("P_SAMPLE11M-I3", "P_SAMPLE11-M", "I",
"NAME");
Consolidator.CreatePublicationItemIndex("P_SAMPLE11D-I2", "P_SAMPLE11-D", "I",
"KEY");
Consolidator.CreatePublicationItemIndex("P_SAMPLE11D-I3", "P_SAMPLE11-D", "I",
"DESCRIPTION");

//Adding Publication Items to a Publication

Consolidator.AddPublicationItem("T_SAMPLE11", "P_SAMPLE11-M", null, null, "S",
null, null);
Consolidator.AddPublicationItem("T_SAMPLE11", "P_SAMPLE11-D", null, null, "S",
null, null);
    }
    catch (Throwable e)
    {
        e.printStackTrace ();
    }

// Creating Users
try {
oracle.mobile.admin.ResourceManager.Example("S11U1");
} catch (Throwable e) {
//e.printStackTrace(); ignore error
}

oracle.mobile.admin.ResourceManager.Example("S11U1", "manager", "S11U1", "C");

// Instantiating a Subscription
Consolidator.Example("T_SAMPLE11", "S11U1");
Consolidator.InstantiateSubscription("T_SAMPLE11", "S11U1");

oracle.mobile.admin.ResourceManager.commitTransaction();
oracle.mobile.admin.ResourceManager.closeConnection();

}
}
```

3.5.2 Create Required Tables Using Standard JDBC

The first section of the program gets a JDBC connection to database MASTER, and creates the base tables ORD_MASTER and ORD_DETAIL in the database. This part of the process can also be done using SQL. If you have gone through the steps described in [Section 3.1.2, "Synchronization Example" in Chapter 3, "Synchronization"](#), these tables have been created in the Mobile Server repository and on the client.

3.5.3 Connecting to the Mobile Server

The following expression connects to the Mobile Server.

openConnection

For example,

```

ResourceManager.openConnection(<USERNAME>, <PASSWORD>);
oracle.mobile.admin.ResourceManager.openConnection
(CONS_SCHEMA,
DEFAULT_PASSWORD);

```

For this example, the <USERNAME> is S11U1 and the <PASSWORD> is MANAGER.

3.5.4 Creating Publications

The next step is to create a publication using the Consolidator Class. Publications are essentially sets of publication items. Sample11.java creates two publications. The DropPublication command is used first to make certain that the publication being created doesn't already exist.

Special characters including spaces are supported in publication names.

3.5.4.1 CreatePublication

CreatePublication has the following syntax:

```

public static void CreatePublication
(String name,
int client_storage_type,
String client_name_template,
String enforce_ri) throws Throwable

```

Example

In Sample11.java, the publication being created is T_SAMPLE11:

```

Consolidator.CreatePublication("T_SAMPLE11", Consolidator.OKPI_CREATOR_ID,
"OrdersODB.%s", null);

```

The parameters of CreatePublication are listed in [Table 3-3](#):

Table 3-3 CreatePublication Parameters

Parameter	Definition
name	The name of the publication being created.
client_storage_type	A constant which defines the platform type.
client_name_template	<p>This is the template for publication item names on client devices. This can be one of several choices:</p> <ul style="list-style-type: none"> ■ %s - This is the default setting which causes the publication item to be stored in the default database, conscli.odb. ■ <DATABASE>.%s - This option stores the publication item in a database named <DATABASE>. This option does not support filename extensions. ■ Instead of using a template, you can use a specific value for publications containing a single publication item. For example, you can use "AddressBook" for the Palm OS Address Book application.
enforce_ri	This parameter is reserved for future enhancement and should always be NULL.

Note: If you use Oracle Database Lite as the client storage type, the database does not have an extension.

3.5.5 Creating Publication Items

After creating the publication, it is necessary to create the publication item. Publication items define the snapshot of the base tables which is downloaded to Oracle Database Lite. The refresh mode of the publication item is specified during creation so it is pre-configured for fast- or complete-refresh. You can also establish data-subsetting parameters when creating the publication item, to provide 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 following examples create a publication item named P_SAMPLE11-M. Before creating the publication item, the sample uses `DropPublicationItem` to clean up any prior publication items that might have the same name.

3.5.5.1 CreatePublicationItem

`CreatePublicationItem` has the following syntax:

```
public static void CreatePublicationItem
    (String name,
     String owner,
     String store,
     String refresh_mode,
     String select_stmt,
     String cbk_owner,
     String cbk_name) throws Throwable
```

The parameters of `CreatePublicationItem` are listed in [Table 3–4](#):

Table 3–4 *CreatePublicationItem Sample Parameters*

Parameter	Definition
name	Specifies the publication item name.
owner	Specifies the base object schema owner. For example, MASTER is the owner of the base object ORD_MASTER.
store	Specifies the base table or view name in the Oracle database. The snapshot which is defined is also assigned this name.
refresh_mode	Defines the refresh mode as fast or complete. See Section 3.7.5, "Fast Refresh and Update Operation for Multi-Table Publications (Views)" for more information.
select_stmt	A SQL select statement which identifies data from the specified columns in the database table.
cbk_owner	Specifies the callback package owner. For more information, see Section 3.7.12, "Callback Customization for Before and After Compose/Apply" .
cbk_name	Specifies the callback package name. For more information, see Section 3.7.12, "Callback Customization for Before and After Compose/Apply" .

Example

In the Sample11.java program, the following commands create snapshot definitions, or publication items, called P_SAMPLE-M and P_SAMPLE-D, of the ORD_MASTER and ORD_DETAIL database tables, which were created in the repository earlier.

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

```
Consolidator.CreatePublicationItem("P_SAMPLE11-D", "MASTER", "ORD_DETAIL", "F",
"SELECT * FROM MASTER.ORD_DETAIL", null, null);
```

3.5.5.2 Defining Publication Items for Updatable Multi-table Views

Publication items can be defined for both tables and views.

When publishing updatable multi-table views, there are certain restrictions that 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 3.7.5, "Fast Refresh and Update Operation for Multi-Table Publications \(Views\)"](#) for more information.
- All base tables of the view must be published.

3.5.5.3 Data Subsetting

Data subsetting is the ability to create specific subsets of data and assign them to a parameter name which can then be assigned to a subscribing user. When creating publication items, a parameterized select statement with a character limit of up to 8k 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

```
Consolidator.CreatePublicationItem("CORP_DIR1", "DIRECTORY1", "ADDR1RL4P", "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.addr1rl4p WHERE company > :COMPANY", null, null);
```

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

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

3.5.6 Sequence Support

Sequence support has been enhanced with the 10g release. The previous implementation currently exists, but will be deprecated and unsupported.

The Enhancements

The following enhancements to sequence support are available.

- **True sequence support on the client** - The Consolidator now 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 will propagate 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.
- **Online and Offline** - There are two types of sequences, online and offline. 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 will be incremented by two and started with staggered values; one will start with an even number and one will start with an odd number. By using an odd/even window model such as the one described above, the Consolidator will ensure uniqueness regardless of whether the application is running in online mode or in offline mode. An offline sequence is similar to an online sequence except that the server-side sequence is not created and the developer can specify the increment value. Whether the sequence uses consecutive numbers or not is up to the application developer.

- **Sequence management** - Once the sequences have been defined and associated with a publication, the Consolidator will manage all aspects of administering them for subscribing users, including allocation of new windows once predefined thresholds are met.
- **Complete Application Programming Interface (API) to manage the sequences** - The API enables you to manage the sequences; for example, create/drop a sequence, add/remove a sequence from a publication, modify a sequence, and advance a sequence window for a user.

See the *Consolidator Admin API Specification* (included on the CD) for a complete listing of the APIs to define and administer sequences.

3.5.7 Defining Client Subscription Parameters for Publications

When a publication uses Data Subsetting parameters, you must set the parameters for each subscription to the publication. An example of a parameter is "COMPANY" and is described in [Section 3.5.5.3, "Data Subsetting"](#).

3.5.7.1 SetSubscriptionParameter

```
public static void SetSubscriptionParameter
    (String publication,
     String clientid,
     String param_name,
     String param_value) throws Throwable
```

The parameters for SetSubscriptionParameter are listed in [Table 3–5](#):

Table 3–5 SetSubscriptionParameter Sample Parameters

Parameter	Definition
publication	Defines the publication from which the subset is to be taken.
clientid	Defines the client ID which the data subset data is for.
param_name	Defines the parameter name.
param_value	Defines the parameter value being passed which determines what data is returned from publication item queries using this parameter.

Example

This example sets the subscription parameter for the client DAVIDL, subscribing to the publication named CORP_DIR1:

```
Consolidator.SetSubscriptionParameter("CORP_DIR1", "DAVIDL", "COMPANY",
    "'DAVECO'");
```

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

3.5.8 Creating 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 API provides calls to explicitly deploy unique, regular, and primary key indexes to clients as well.

3.5.8.1 CreatePublicationItemIndex

CreatePublicationItemIndex uses the following syntax:

```
public static void CreatePublicationItemIndex
    (String name,
     String publication_item,
     String pmode,
     String columns) throws Throwable
```

The parameters of CreatePublicationItemIndex are listed in [Table 3–6](#):

Table 3–6 *CreatePublicationItemIndex Parameters*

Parameter	Definition
name	Defines the name of the index to be created.
publication_item	Defines the index's publication item.
pmode	Defines the index mode, I - regular, U - unique, P - primary key mode. See Section 3.5.8.2, "Define Client Indexes" for more information.
columns	Defines the names of the columns included in the index. There can be more than one column listed per statement, the list of columns should be separated by commas and not contain any spaces.

Example 1

In our Sample11.java sample code this takes the following form:

```
Consolidator.CreatePublicationItemIndex("P_SAMPLE11M-I1", "P_SAMPLE11-M", "I",
"DDATE");
Consolidator.CreatePublicationItemIndex("P_SAMPLE11M-I2", "P_SAMPLE11-M", "I",
"STATUS");
Consolidator.CreatePublicationItemIndex("P_SAMPLE11M-I3", "P_SAMPLE11-M", "I",
"NAME");
Consolidator.CreatePublicationItemIndex("P_SAMPLE11D-I2", "P_SAMPLE11-D", "I",
"KEY");
Consolidator.CreatePublicationItemIndex("P_SAMPLE11D-I3", "P_SAMPLE11-D", "I",
"DESCRIPTION");
```

Sample11.java creates 5 indexes which establish regular indexes on the "DDATE", "STATUS", and "NAME" columns of the P_SAMPLE-M publication item, and the "KEY" and "DESCRIPTION" columns of the P_SAMPLE-D publication item. An index can contain more than one column. You could also define an index with multiple columns as follows:

Example 2

```
Consolidator.CreatePublicationItemIndex("P_SAMPLE11D-I1", "P_SAMPLE11-D", "I",
"KEY,DESCRIPTION");
```

3.5.8.2 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, Mobile Sync may fail with a duplicate key violation. See the *Consolidator Admin API Specification* for more information.

3.5.9 Adding Publication Items to a Publication

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 *Consolidator Admin API Specification* for more information.

3.5.9.1 AddPublicationItem

The syntax for AddPublicationItem is:

```
public static void AddPublicationItem
(String publication,
String item,
String columns,
String disabled_dml,
String conflict_rule,
String restricting-predicate,
String weight) throws Throwable
```

The following examples add a publication item named P_SAMPLE1 to the publication T_SAMPLE1. The parameters of AddPublicationItem are listed in [Table 3-7](#):

Table 3-7 AddPublicationItem Parameter

Parameter	Definition
publication	Defines the publication to receive the new item.
item	Defines the publication item to be added.
columns	<p>Specifies a new name for publication item columns Using null specifies that no columns are renamed. All columns in the publication item query must be specified in the proper order which is either:</p> <ul style="list-style-type: none"> ■ The order specified in the publication item's select statement. ■ If you are using a statement with "SELECT * FROM..." then the column names must be ordered identically to the order of the base table or view.

Table 3–7 (Cont.) AddPublicationItem Parameter

Parameter	Definition
disabled_dml	Specifies options for disabling DML. The possible values are: <ul style="list-style-type: none"> ■ Y - Defines a fully updatable publication item. ■ N - Defines a read-only publication item. You can also define a read-only publication item by using the "IUD" option. ■ I - Disables the propagation of individual insert operations. ■ U - Disables the propagation of individual update operations. ■ D - Disables the propagation of individual delete operations. ■ null - Specifies that no options are selected for disabling DML.
conflict_rule	Defines the winner in conflict resolution: either 'C' for client wins or 'S' for server wins. See Section 3.5.9.2, "Defining Conflict Rules" for more information.
restricting_predicate	Specifies high-priority mode. A restricting predicate can be assigned to a publication item as it is added to a publication. When a client is synchronizing in high priority mode, the predicate is used to limit data pushed to the client. This parameter can be null. This parameter is for advanced use.
weight	Specified as null or an integer to determine priority in executing Client Operations to master tables. See Section 3.5.9.3, "Using Table Weight" for more information. This value must be an integer between 1 and 1023.

Example

```
Consolidator.AddPublicationItem("T_SAMPLE1", "P_SAMPLE1", null, null, "S", null, null);
```

3.5.9.2 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 'C' or the server 'S'. 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.

3.5.9.3 Using Table Weight

Table weight is an integer property of association between publications and publication items. Mobile Server uses table weight to determine which order to apply Client Operations to master tables within each publication, 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.
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 after those for any publication item of a lower weight within the same publication.

3.5.10 Creating Users

Sample11 has you drop users using `dropUser()`, before creating the new user. This serves to clear out any spurious user ID's before creating the new one. See [Section 3.5.11, "Drop User"](#) for details. The parameters for this function are not case sensitive.

3.5.10.1 createUser

The syntax for `createUser` is:

```
public static boolean createUser
    (String userName,
     String password,
     String fullName,
     String privilege) throws Throwable;
```

The parameters of `createUser` are listed in [Table 3–8](#):

Table 3–8 *createUser - Sample Parameters*

Parameter	Definition
<code>userName</code>	Defines the user name for mobile client.
<code>password</code>	Defines the password for this user name.
<code>fullName</code>	Optional. Specifies the full name for user, for example, John Smith.

Table 3–8 (Cont.) createUser - Sample Parameters

Parameter	Definition
privilege	<p>This parameter defines the Mobile Server user privilege. This value can be one of the following:</p> <ul style="list-style-type: none"> ▪ "O" for publishing an application ▪ "U" for connecting to the Mobile Server ▪ "A" for administrating the Mobile Server ▪ NULL represents no privilege

The following example creates a user "S11U1" with the parameters listed in the table:

Example

```
oracle.mobile.admin.ResourceManager.createUser("S11U1", "manager", "John Smith", "C")
```

3.5.11 Drop User

You can drop existing Mobile Server users with the `dropUser` function. The parameters for this function are not case sensitive.

3.5.11.1 dropUser

The syntax for `dropUser` is:

The following example drops the user "S11U1":

```
public static void dropUser(String userName);
```

The parameters of `dropUser` are listed in [Table 3–9](#):

Table 3–9 dropUser - Sample Parameters

Parameter	Definition
userName	Specifies user name for mobile client.

Example

```
oracle.mobile.admin.ResourceManager.dropUser("S11U1");
```

3.5.12 Subscribing Users to a Publication

You can subscribe users to a publication using the `CreateSubscription` function.

3.5.12.1 CreateSubscription

`CreateSubscription` has the following syntax:

```
public static void CreateSubscription
    (String publication,
     String clientid) throws Throwable
```

The following examples subscribe the client, S11U1, to the publication, T_SAMPLE11, with the parameters listed in [Table 3–10](#).

Table 3–10 Create Subscription - Sample Parameters

Parameter	Definition
publication	Specifies the publication being subscribed to.
clientid	Specifies the user subscribing to the publication.

Example

```
Consolidator.CreateSubscription("T_SAMPLE11", "S11U1");
```

3.5.13 Instantiating a Subscription

After you subscribe a user to a publication, you then complete the subscription process by instantiating the subscription. When the Mobile Server instantiates a subscription, it creates a complete internal representation of the subscription.

Note: If you need to set subscription parameters for data subsetting, this must be completed before instantiating the subscription. See [Section 3.5.5.3, "Data Subsetting"](#) for more information.

3.5.13.1 InstantiateSubscription

The syntax for `InstantiateSubscription` is:

```
public static void InstantiateSubscription
(String publication,
String clientid) throws Throwable
```

The parameters for `InstantiateSubscription` are listed in [Table 3–11](#).

Table 3–11 InstantiateSubscription - Sample Parameters

Parameter	Definition
publication	Specifies the publication being subscribed to.
clientid	Specifies the user subscribing to the publication.

The following example instantiates a client's subscription to a publication, with the values specified in the table:

Example

```
Consolidator.InstantiateSubscription("T_SAMPLE1", "DAVIDL");
```

3.6 Other Standard Consolidator Functionality

The API calls used in [Section 3.4, "The Publish and Subscribe Model and Oracle Database Lite Synchronization"](#) are those necessary when creating publications, publication items, and subscriptions programmatically. The topics in this section are used less frequently, but are still important.

- [Section 3.6.1, "Client Device Database DDL Operations"](#)
- [Section 3.6.2, "Change Password"](#)
- [Section 3.6.3, "Remote Database Link Support"](#)

3.6.1 Client Device Database DDL Operations

The first time a client synchronizes, the Mobile Server automatically enables Mobile Server 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 the through a publication item. If you do not want the primary index, you must explicitly drop it from the publication items. See the *Consolidator Admin API Specification*, for specific API information.

3.6.2 Change Password

You can change passwords for Mobile Server users with the `setPassword()` function which has the following syntax:

3.6.2.1 setPassword

The syntax for `setPassword` is:

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

The parameters for `setPassword` are listed in [Table 3–12](#):

Table 3–12 *setPassword - Sample Parameters*

Parameter	Definition
userName	Specifies user name for mobile client.
newpwd	Specifies the new password for the mobile client.

The following example changes the password for the user "MOBILE":

Example

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

3.6.3 Remote Database Link Support

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 logging objects.

The synonyms should then be published using the `CreatePublicationItem` API. 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 APIs 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.

3.6.3.1 Publishing Synonyms for the Remote Object Using CreatePublicationItem

The `CreatePublicationItem` API, used with the following parameters, creates a new, stand-alone publication item as a remote database object.

Syntax

```
public static void CreatePublicationItem
    ((String rmt_jdbc_url),
     String name,
     String owner,
     String store,
     String refresh_mode,
     String select_stmt,
     String cbk_owner,
     String cbk_name) throws Throwable
```

or,

```
public static void CreatePublicationItem
    ((Connection rmt_jdbc_conn),
     String name,
     String owner,
     String store,
     String refresh_mode,
     String select_stmt,
     String cbk_owner,
     String cbk_name) throws Throwable
```

The parameters for synonym creation using `CreatePublicationItem` are listed in [Table 3–13](#):

Table 3–13 *CreatePublicationItem Parameters for Remote Database Linking*

Parameter	Description
<code>rmt_jdbc_url</code>	The string specifying a jdbc URL for the remote database instance.
<code>rmt_jdbc_conn</code>	The connection to the Oracle database where the remote instance resides.
<code>name</code>	A string defining a new publication item name.
<code>owner</code>	A string specifying the synonym owner.
<code>store</code>	A string specifying the synonym name. Note: to publish a remote object, a private synonym for it must be created.
<code>refresh_mode</code>	A string specifying the refresh mode. F for fast refresh or C for complete refresh. The default is fast refresh.
<code>select_stmt</code>	A string specifying a select statement for the new publication. This statement may be parameterized. In the example that follows the parameter is :CAP, defined by placing a colon in front of the parameter name.

Table 3–13 (Cont.) CreatePublicationItem Parameters for Remote Database Linking

Parameter	Description
cbk_owner	Specifies the callback package owner as NULL. For more information, see Section 3.7.12, "Callback Customization for Before and After Compose/Apply" .
cbk_name	Specifies the callback package name as NULL. For more information, see Section 3.7.12, "Callback Customization for Before and After Compose/Apply" .

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.

Example

```
Consolidator.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.

3.6.3.2 Creating a Dependency Hint

This creates a hint for a non-apparent dependency.

Syntax

```
public static void DependencyHint
    (String owner,
     Sting store,
     String owner_d,
     String store_d) throws Throwable
```

The parameters for CreateDependencyHint are listed in [Table 3–14](#):

Table 3–14 CreateDependencyHint Parameters

Parameter	Description
owner	A string specifying the owner of the view.
store	A string specifying the name of the view.
owner_d	A string specifying the owner of the base table or view.
store_d	A string specifying the name of the base table or view.

Example

```
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`:

```
Consolidator.DependencyHint("SAMPLE1", "V_PAYROLL_SYN", "SAMPLE1", "T_EMP_SYN");
```

3.6.3.3 Remove a Dependency Hint

This removes a hint for a non-apparent dependency.

Syntax

```
public static void RemoveDependencyHint
(String owner,
String store,
String owner_d,
String store_d) throws Throwable
```

The parameters for `RemoveDependencyHint` are listed in [Table 3–15](#):

Table 3–15 *RemoveDependencyHint Parameters*

Parameter	Description
owner	A string specifying the view owner.
store	A string specifying the view name.
owner_d	A string specifying the base object owner.
store_d	A string specifying the base object name.

3.7 Advanced Features for Customizing Consolidator

The following features include special functions which are not required for most application designs. These features may require advanced understanding of both Java and the design of the database being manipulated, including how queries have been constructed, how tables have been arranged and any dependencies that apply. The topics discussed are:

- [Section 3.7.1, "Compose Phase Customization Using MyCompose"](#)
- [Section 3.7.2, "Sync Discovery API"](#)
- [Section 3.7.3, "Map Table Partition APIs"](#)
- [Section 3.7.4, "Modifying a Publication Item Using AlterPublicationItem"](#)
- [Section 3.7.5, "Fast Refresh and Update Operation for Multi-Table Publications \(Views\)"](#)
- [Section 3.7.6, "Virtual Primary Key"](#)
- [Section 3.7.7, "Caching Publication Item Queries"](#)
- [Section 3.7.8, "Binding User-Defined PL/SQL Procedures"](#)
- [Section 3.7.9, "Queue Interface for Customizing Replication"](#)
- [Section 3.7.10, "Null Sync Callout"](#)

- [Section 3.7.11, "Foreign Key Constraints in Updatable Publication Items"](#)
- [Section 3.7.12, "Callback Customization for Before and After Compose/Apply"](#)
- [Section 3.7.13, "Callback Customization for DML Operations"](#)
- [Section 3.7.14, "Restricting Predicate"](#)

3.7.1 Compose Phase Customization Using MyCompose

The compose phase takes a query on one or more server-side base tables and puts the generated DML operations for the publication item the query describes into the out queue to be downloaded into the client. Consolidator manages these DML operations in a "generic" way 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 APIs to register and load the customized class.

3.7.1.1 Extending MyCompose as a User Defined Sub-Class

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

ItemACompose

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

The user-written class produces publication item DML operations to be sent to a client device by interpreting the base table DML logs. The extended `MyCompose` sub-class is registered with a publication item, and takes over all compose phase operations for that publication item. An extended `MyCompose` class can be registered with more than one publication item if it is sufficiently generic, however, internally, there is a unique instance of the extended class for each publication item.

3.7.1.2 Primary MyCompose Methods

The `MyCompose` class uses the following four methods: `needCompose`, `doCompose`, `init`, and `destroy` to customize the compose phase. One or more of these methods can be overridden in the customized sub-class to customize compose phase operations. For most users attempting to customize the compose phase for one client at a time, `doCompose` and `needCompose` are sufficient. The `init` and `destroy` methods are used when some process must be performed for all clients, either before or after individual client processing. There are several more methods described in [Section 3.7.1.3, "Subsidiary MyCompose Methods"](#) that provide useful information for the use of these four methods.

3.7.1.2.1 needCompose Method

Use this method to identify a client that has changes to a specific publication item to be downloaded. This method is primarily useful as a way to trigger `doCompose`.

Syntax

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

The parameters for `needCompose` are listed in [Table 3–16](#):

Table 3–16 *needCompose Parameters*

Parameter	Definition
<code>conn</code>	Database connection to the Mobile Server repository.
<code>clientid</code>	Specifies the client which is connecting to the database.

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

Example

```
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 code overrides the `needCompose` method, and uses subsidiary methods discussed in [Section 3.7.1.3, "Subsidiary MyCompose Methods"](#), 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 `doCompose`.

3.7.1.2.2 doCompose Method

This method populates the DML log table for a specific publication item subscribed to by a client.

Syntax

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

The parameters for `doCompose` are listed in [Table 3–17](#):

Table 3–17 *doCompose Parameters*

Parameter	Definition
<code>conn</code>	Database connection to the Mobile Server repository.
<code>clientid</code>	Specifies the client which is connecting to the database.

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

Example

```
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 sample code overrides the `doCompose` method and uses subsidiary methods discussed in [Section 3.7.1.3, "Subsidiary MyCompose Methods"](#) to create a SQL statement. Using this sample you have `MyCompose` retrieve 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 then use the table names and other information returned by these methods to create a dynamic SQL statement ("`sql`") which performs an insert into the publication item DML table of the contents of the base table primary key and DML operation from the base table DML log.

3.7.1.2.3 init Method

This 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.

Syntax

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

The parameters for `init` are listed in [Table 3–18](#):

Table 3–18 *init Parameters*

Parameter	Definition
<code>conn</code>	Database connection to the Mobile Server repository.

3.7.1.2.4 destroy Method

This method provides the framework for user-created 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.

Syntax

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

The parameters for destroy are listed in [Table 3–18](#):

Table 3–19 *destroy Parameters*

Parameter	Definition
conn	Database connection to the Mobile Server repository.

3.7.1.3 Subsidiary MyCompose Methods

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

3.7.1.3.1 getPublication

This returns the name of the publication.

Syntax

```
public String getPublication()
```

3.7.1.3.2 getPublicationItem

This returns the publication item name.

Syntax

```
public String getPublicationItem()
```

3.7.1.3.3 getPubItemDMLTableName

Returns the name of the DML table or DML table view, including schema name, which doCompose or init are supposed to insert into.

Syntax

```
public String getPubItemDMLTableName()
```

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

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

The parameters for getPubItemDMLTableName are listed in [Table 3–20](#):

Table 3–20 *getPubItemDMLTableName View Structure Parameters*

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

3.7.1.3.4 getPubItemPK

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

Syntax

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

3.7.1.3.5 `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]`

Syntax

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

3.7.1.3.6 `getBaseTablePK`

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

Syntax

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

The parameters for `getBaseTablePK` are listed in [Table 3–21](#):

Table 3–21 *`getBaseTablePK` Parameters*

Parameter	Definition
<code>owner</code>	The schema name of the base table owner.
<code>baseTable</code>	The base table name.

3.7.1.3.7 `baseTableDirty`

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

Syntax

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

The parameters for `baseTableDirty` are listed in [Table 3–22](#):

Table 3–22 *`baseTableDirty` Parameters*

Parameter	Definition
<code>owner</code>	The schema name of the base table.
<code>store</code>	The base table name.

3.7.1.3.8 `getBaseTableDMLLogName`

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

Syntax

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

The parameters for `getBaseTableDMLLogName` are listed in [Table 3–23](#):

Table 3–23 *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's physical primary key corresponds to the primary key of the publication item. The structure of the log is:

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

The parameters for `getBaseTableDMLLogName` view structure are listed in [Table 3–24](#):

Table 3–24 *getBaseTableDMLLogName View Structure Parameters*

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

3.7.1.3.9 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.

Syntax

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

The structure of the map table view is:

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

The parameters of the map table view are listed in [Table 3–25](#):

Table 3–25 *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.

3.7.1.4 Consolidator API Methods for Registering MyCompose Sub-Classes

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

3.7.1.4.1 RegisterMyCompose Method

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.

Syntax

```
public static void RegisterMyCompose
( String publication,
  String pubItem,
  String className,
  boolean reloadBytecode) throws Throwable
```

The parameters of `RegisterMyCompose` are listed in [Table 3–26](#):

Table 3–26 *RegisterMyCompose Parameters*

Parameter	Definition
publication	The name of the publication the publication item is part of.
pubItem	The name of the publication item to which the sub-class is being registered.
className	The name of the customized MyCompose sub-class.
reloadBytecode	If this value is true, then the existing byte code for the class in the Mobile Server repository is overwritten.

3.7.1.4.2 DeRegisterMyCompose

The `DeRegisterMyCompose` method removes the sub-class from the Mobile Server repository.

Syntax

```
public static void DeRegisterMyCompose
( String publication,
  String pubItem,
  boolean removeBytecode) throws Throwable
```

The parameters of `DeRegisterMyCompose` are listed in [Table 3–27](#):

Table 3–27 *DeRegisterMyCompose Parameters*

Parameter	Definition
publication	The name of the publication the publication item belongs too.
pubItem	The name of the publication item the sub-class is being registered too.
removeBytecode	If this value is true, then the existing byte code for the class in the Mobile Server repository is removed. If the byte code is removed, all publication items registered with this class have their registration removed.

3.7.2 Sync Discovery API

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.

3.7.2.1 getDownloadInfo Method

The API consists of the `getDownloadInfo` method which 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.

Syntax

```
public DownloadInfo getDownloadInfo
    (String clientid,
     boolean uncompressed,
     boolean completeRefresh)
```

The parameters of `getDownloadInfo` are listed in [Table 3–28](#):

Table 3–28 *getDownloadInfo Parameters*

Parameter	Description
<code>clientid</code>	The name of the client.
<code>uncompressed</code>	If set to true, returns the true size of the data object, if false the size of the data object after being compressed.
<code>completeRefresh</code>	If set to true, returns the size of all rows that will be synchronized during a complete refresh regardless of the refresh mode.

Example

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

3.7.2.2 DownloadInfo Class Access Methods

The access methods provided by the `DownloadInfo` class are listed in [Table 3–29](#):

Table 3–29 *DownloadInfo Class Access Methods*

Method	Definition
<code>public Iterator iterator ()</code>	This returns an <code>Iterator</code> object so that the user can traverse through the all the <code>PublicationSize</code> objects that are contained inside the <code>DownloadInfo</code> object.
<code>public long getTotalSize ()</code>	This returns the size information of all <code>PublicationSize</code> 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'.
<code>public long getPubSize (String pubName)</code>	This returns the size of all publication items that belong to the publication referred to by the string <code>pubName</code> . If no historical information is available for those publication items, the value returned is '-1'.

Table 3–29 (Cont.) DownloadInfo Class Access Methods

Method	Definition
<code>public long getPubRecCount (String pubName)</code>	This will return the number of all records of all the publication items that belong to the publication referred by the string <code>pubName</code> , that will be synchronization during the next synchronization.
<code>public long getPubItemSize (String pubItemName)</code>	This will return the size of a particular publication item referred by <code>pubItemName</code> . It follows the following rules in order. <ol style="list-style-type: none"> 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'.
<code>public long getPubItemRecCount (String pubItemName)</code>	This will return the number of records of the publication item referred by <code>pubItemName</code> that will be synced in the next synchronization.

3.7.2.3 PublicationSize Class

The access methods provided by the `PublicationSize` class are listed in [Table 3–30](#):

Table 3–30 PublicationSize Class Access Methods

Parameter	Definition
<code>public String getPubName ()</code>	This will return the name of the publication containing the publication item.
<code>public String getPubItemName ()</code>	This will return the name of the publication item referred to by the <code>PublicationSize</code> object.
<code>public long getSize ()</code>	This will return the total size of the publication item referred to by the <code>PublicationSize</code> object.
<code>public long getNumOfRows ()</code>	This will return the number of rows of the publication item that will be synchronized in the next synchronization.

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 connection
        try
        {
            // Create a ConsolidatorManager object
```

```
        ConsolidatorManager cm = new ConsolidatorManager ();
// Open a Consolidator 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();
}
}

```

3.7.3 Map Table Partition APIs

Consolidator 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 clientid, 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 10g.

3.7.3.1 Create a Map Table Partition

Creates 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.

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 3–31](#).

Table 3–31 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 <i>Oracle9i SQL Reference</i> 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 <i>Oracle9i SQL Reference</i> for more information.

Example

```
Consolidator.PartitionMap("P_SAMPLE1", 5, "tablespace mobileadmin", "initrans 10
pctfree 70");
```

3.7.3.2 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.

Syntax

```
public static void AddMapPartition
```

```
( String pub_item,
  int num_parts,
  String storage,
  String ind_storage) throws Throwable
```

The parameters of `AddMapPartition` are listed in [Table 3–32](#):

Table 3–32 AddMapPartitions Parameters

Parameter	Definition
<code>pub_item</code>	The publication item whose map table is being partitioned.
<code>num_parts</code>	The number of partitions.
<code>storage</code>	A string specifying the storage parameters. This parameter requires the same syntax as the SQL command <code>CREATE TABLE</code> . See the <i>Oracle Database Lite SQL Reference</i> for more information.
<code>ind_storage</code>	A string specifying the storage parameters for indexes on the partition. This parameter requires the same syntax as the SQL command <code>CREATE INDEX</code> . See the <i>Oracle Database Lite SQL Reference</i> for more information.

Example

```
Consolidator.AddMapPartitions("P_SAMEPLE1",5,"tablespace mobileadmin","initrans 10
pctfree 40");
```

3.7.3.3 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 Consolidator.

Syntax

```
public static void DropMapPartition( String partition) throws Throwable
```

Example

```
Consolidator.DropMapPartition("MAP101_1");
```

3.7.3.4 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

```
Consolidator.DropAllMapPartitions("P_SAMPLE1");
```

3.7.3.5 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 Consolidator.

Syntax

```
public static void MergeMapPartitions
( String from_partition,
  String to_partiton) throws Throwable
```

Example

```
Consolidator.MergeMapPartition("MAP101_1", "MAP101_2");
```

3.7.4 Modifying a Publication Item Using AlterPublicationItem

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.
- This feature is supported for all client formats.
- 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 on EPOC.
- 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.

3.7.4.1 Alter Publication Item

This allows additional columns to be added to an existing publication item. The WHERE clause may also be altered, but additional subscription parameters may not be added.

Syntax

```
public static void AlterPublicationItem
( String name,
  String select_stmt)
throws Throwable
```

The parameters for `AlterPublicationItem` are listed in [Table 3–33](#):

Table 3–33 *Alter Publication Item Parameters*

Parameter	Description
name	A character string specifying the publication item name.
select_stmt	A new publication item select statement containing additional columns.

Example

```
Consolidator.AlterPublicationItem("P_SAMEPLE1", "select * from EMP");
```


3.7.5 Fast Refresh and Update Operation for Multi-Table Publications (Views)

The Mobile Server supports fast refresh and update operations for complex multiple table publication items called views, that meet specific criteria. During a fast refresh, incremental changes are synchronized, 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 you must first drop the publication item and recreate it with the appropriate mode.

3.7.5.1 Updatable Parent Tables

For a view to be updatable, it must have a parent table. A parent table can be any one of the view's base tables in which a primary key is included in the view's column list and is unique in the view's row set. If you want to make a view updatable, you must provide the Mobile Server with the appropriate hint and the view's parent table before you create a publication item on the view.

3.7.5.2 Using Parent Table Hints and INSTEAD OF Triggers

To make publication items based on a view updatable, you must use the following two mechanisms:

- Parent table hints
- INSTEAD OF triggers or DML procedure callouts

3.7.5.2.1 Creating a Parent Hint

Parent table hints define the parent table for a given view. Parent table hints are provided through the `ParentHint` function which uses the stoats:

```
public static void ParentHint
    (String owner,
     String store,
     String owner_d,
     String store_d) throws Throwable
```

The parameters for `ParentHint` are listed in [Table 3–34](#):

Table 3–34 *ParentHint Parameters*

Parameter	Description
owner	A string specifying the view owner.
store	A string specifying the view name.
owner_d	A string specifying the base object owner.
store_d	A string specifying the base object name.

Example

```
Consolidator.ParentHint("SAMPLE3", "ADDR0LRL4P", "SAMPLE3", "ADDRESS");
```

3.7.5.2.2 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's base tables.

INSTEAD OF triggers are a function of Oracle database. See the Oracle database documentation for details on INSTEAD OF triggers.

3.7.5.3 Fast Refresh for Views

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's base tables must have a primary key.
- All primary keys from all base tables must be included in the view's 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 `PrimaryKeyHint`.

3.7.5.3.1 PrimaryKeyHint

The syntax for `PrimaryKeyHint` is:

```
public static void PrimaryKeyHint
    (String publication_item,
     String column,
     String b_owner,
     String b_store,
     String b_column) throws Throwable
```

The parameters for `PrimaryKeyHint` are listed in [Table 3–35](#):

Table 3–35 PrimaryKeyHint Parameters

Parameter	Description
<code>publication_item</code>	The name of the publication item the primary key hint is to be mapped to.
<code>owner</code>	A string specifying the view owner.
<code>store</code>	A string specifying the view name.
<code>store_d</code>	A string specifying the base object owner.
<code>b_column</code>	The name of the base table column the hint is using.

Example

```
Consolidator.ParentHint("SAMPLE3", "ADDR0LRL4P", "SAMPLE3", "ADDRESS");
```

3.7.5.4 Complete Refresh for Views

Publication items can be created for complete refresh using the `Complete Refresh` call from the Consolidator 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 on an entire publication on an entire publication via an API call. The complete refresh function forces complete refresh of a publication for a given client.

3.7.5.4.1 CompleteRefresh

The syntax for CompleteRefresh is:

```
public static void CompleteRefresh
    (String client_id,
     String publication) throws Throwable
```

The parameters for CompleteRefresh are listed in [Table 3–36](#):

Table 3–36 *AlterPublicationItem Parameters*

Parameter	Description
client_id	The Consolidator client name.
publication	The name of the publication to be refreshed.

3.7.6 Virtual Primary Key

You can specify a virtual primary key for publication items where the base object does not have a primary key defined. A virtual primary key can be created for more than one column, but the API must be called separately for each column you wish to assign a virtual primary key. The following methods create and drop a virtual primary key.

3.7.6.1 Create Virtual Primary Key Column

This creates a virtual primary key column.

Syntax

```
public static void CreateVirtualPKColumn
    (String owner,
     String store,
     String column) throws Throwable
```

The parameters for CreateVirtualPKColumn are listed in [Table 3–37](#):

Table 3–37 *CreateVirtualPKColumn Parameters*

Parameter	Description
owner	A string specifying a the owner of the base table or view.
store	A string specifying the base table or view.
column	A string specifying the primary key column.

Example

```
Consolidator.CreateVirtualPKColumn("SAMPLE1", "DEPT", "DEPT_ID");
```

3.7.6.2 Drop Virtual Primary Key Column

This allows a virtual primary key to be dropped.

Syntax

```
public static void DropVirtualPKColumn
```

```
(String owner,
String store) throws Throwable
```

The parameters for `DropVirtualPKColumn` are listed in [Table 3–38](#):

Table 3–38 *DropVirtualPKColumn Parameters*

Parameter	Description
owner	A string specifying a the owner of the base table or view.
store	A string specifying the base table or view.

Example

```
Consolidator.DropVirtualPKColumn("SAMPLE1", "DEPT");
```

3.7.7 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 Consolidator 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 Consolidator 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.

3.7.7.1 Enabling Publication Item Query Caching

The following API enables publication item query caching.

Syntax

```
public static void EnablePublicationItemQueryCache(String name)
throws Throwable
```

The parameters for `EnablePublicationItemQueryCache` are listed in [Table 3–39](#):

Table 3–39 *EnablePublicationItemQueryCache Parameters*

Parameters	Description
name	A string specifying the name of the publication item.

Example

```
Consolidator.EnablePublicationItemQueryCache(
    "P_SAMPLE1");
```

3.7.7.2 Disabling Publication Item Query Caching

The following API disables publication item query caching.

Syntax

```
public static void DisablePublicationItemQueryCache(String name)
    throws Throwable
```

The parameters for `DisablePublicationItemQueryCache` are listed in [Table 3–40](#):

Table 3–40 *DisablePublicationItemQueryCache Parameters*

Parameters	Description
name	A string specifying the name of the publication item.

Example

```
Consolidator.DisablePublicationItemQueryCache("P_SAMPLE1");
```

3.7.8 Binding User-Defined PL/SQL Procedures

The Mobile Server synchronization process can be customized in many ways. You can attach application logic to the Mobile Server by binding PL/SQL procedures to publication items. The procedures must expose the `BeforeCompose`, `AfterCompose`, `BeforeApply`, and `AfterApply` methods of the Consolidator API. The Mobile Server calls these methods before and after it:

- Applies client changes to server tables on behalf of Mobile Sync clients.
- Composes fast-refresh changes for a given publication item.

The Mobile Server passes the current Mobile Sync user name information to these methods.

User-defined PL/SQL procedures can cache or pre-compute data. They can also resolve foreign key constraint violation problems. See [Section 3.7.11, "Foreign Key Constraints in Updatable Publication Items"](#) for more information. See [Section 3.7.12, "Callback Customization for Before and After Compose/Apply"](#) for details on using these calls.

3.7.9 Queue Interface for Customizing Replication

Application developers can manage the replication process programmatically by using the [CreateQueuePublicationItem API](#). 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 3.7.9.3, "Queue Interface PL/SQL Procedure"](#) and by creating the queues themselves.

3.7.9.1 Queue Interface Operation

When data arrives from the client it is placed in the publication item in queues. Consolidator calls `UPLOAD_COMPLETE` once the data has been committed. All records in the current synchronization session are given the same transaction

identifier. Consolidator has a [Queue Control Table](#) (C\$INQ+name) 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 Consolidator 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. Consolidator 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 Consolidator refers to C\$PUB_LIST_Q to determine which items will be downloaded to the client.

Similar to in queue, every record in the out queue should be associated with it a transaction identifier (TRANID\$\$). Consolidator 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. Consolidator only downloads records from the out queues when the value of TRANID\$\$ is greater than last_tran. When the data is downloaded, Consolidator calls DOWNLOAD_COMPLETE.

3.7.9.2 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 Consolidator 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:

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),
  ..
)
```

Consolidator 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'+name
(
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 3–41](#):

Table 3–41 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: <ul style="list-style-type: none"> ■ '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.

3.7.9.3 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;
/

```

3.7.9.4 CreateQueuePublicationItem API

This API call 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.

Syntax

```

public static void CreateQueuePublicationItem
( String name,
  String owner,
  String store,
  String select_stmt,
  String pk_columns,
  String cbk_owner,
  String cbk_name) throws Throwable

```

The parameters for `CreateQueuePublicationItem` are listed in [Table 3–42](#):

Table 3–42 *CreateQueuePublicationItem Parameters*

Parameter	Description
name	Defines a new publication item/queue name.
owner	This is the owner of the base table or view.
store	This value specifies the name of the base table or view.
select_stmt	A string specifying a select statement for the new publication item. This statement can include a subscription parameter.
pk_columns	A comma separated list which creates virtual primary keys.
cbk_owner	Specifies the callback package owner. For more information, see Section 3.7.12, "Callback Customization for Before and After Compose/Apply" . This is an advanced feature.
cbk_name	Specifies the callback package name. For more information, see Section 3.7.12, "Callback Customization for Before and After Compose/Apply" . This is an advanced feature.

You must provide Consolidator with the primary key of the `owner.store` in order to create a queue that can be updated or fast-refreshed. If the `store` has no primary key, one can be specified in the `pk_columns` parameter. If `pk_columns` is null, Consolidator uses the primary key of the `store`.

3.7.9.5 Defining a PL/SQL Package Outside the Repository

The PL/SQL package can be defined outside of the Mobile Server repository if necessary, although in order to function it must still refer to the `in queues`, `out queues`, `queue control table` and `temporary table`, which are defined inside the repository. The following API calls are used to retrieve the procedure name, register, or remove a procedure.

3.7.9.5.1 RegisterQueuePkg

This registers the string 'pkg' as the current procedure.

Syntax

```
public String RegisterQueuePkg(String pkg) throws SQLException
```

Example

```
Consolidator.RegisterQueuePkg("ASL.QUEUES_PKG");
```

3.7.9.5.2 GetQueuePkg

This call returns the name of the currently registered procedure.

Syntax

```
public String GetQueuePkg() throws SQLException
```

3.7.9.5.3 UnRegisterQueuePkg

This removes the currently registered procedure.

Syntax

```
public String UnRegisterQueuePkg() throws SQLException
```

3.7.10 Null Sync Callout

Mobile Server makes a callout during synchronization indicating whether the client is attempting a null sync. A null sync refers to the fact that the client has no changes to upload. This callout can be implemented by creating a PL/SQL procedure within the Mobile Server repository. The procedure must have the following specification:

```
create or replace package CUSTOMIZE as procedure
NullSync(p_Client IN varchar2, p_NullSync as boolean);
end CUSTOMIZE;
```

3.7.11 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.

3.7.11.1 Foreign Key Constraint Violation Example

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

3.7.11.2 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. 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 DEFERRED 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;

```

3.7.11.3 Avoiding Constraint Violations with Table Weights

Mobile Server uses table weight to determine 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 3.7.11.1, "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)
```

3.7.12 Callback Customization for Before and After Compose/Apply

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.

These procedures enable you to incorporate customized code into the process. The `clientname` and `tranid` are passed to allow for customization at the user and transaction level.

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

This procedure must be called after all client's data is applied.

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

This procedure must be called before client's data with `tranid` is applied.

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

This procedure must be called after client's data with `tranid` is applied.

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

This procedure must be called before out queue is composed.

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

This procedure must be called after out queue is composed.

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

3.7.13 Callback Customization for DML Operations

Once a publication item has been created, 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 that publication item. There can be only one mobile DML procedure for each publication item. The procedure should be created with the following structure:

```
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 3–43](#):

Table 3–43 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 in the publication item query. If the publication item was created with "SELECT * FROM example", the column order must be the same as they appear in the table "example".
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.

For example, if you want to have a DML procedure for publication item "example", which is defined by the following query:

```
select A,B,C from publication_item_example_table
```

Assuming "A" is the primary key column for "example", 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 will be called 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:

```
Consolidator.AddMobileDmlProcedure("PUB_example", "example", "any_schema.any_
package.any_name")
```

where "example" is the publication item name and "PUB_example" is the publication name.

Please refer to the *Consolidator Admin API Specification* for more information on calling this API.

3.7.13.1 DML Procedure Example

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

SQL Statement

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:

```
Consolidator.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.

3.7.14 Restricting Predicate

A restricting predicate can be assigned to a publication item as it is added to a publication. When a client is synchronizing in high priority mode, the predicate is used to limit data downloaded to the client. This parameter can be null. This parameter is for advanced use. For using a restricting predicate in high-priority replication, see [Section 3.7.15, "Priority-Based Replication"](#).

3.7.15 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 Admin 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.

3.7.16 Shared Maps

This section discusses the shared maps feature in terms of concepts and performance attributes.

3.7.16.1 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 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.

3.7.16.2 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.

3.7.16.3 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 3.7.16.1, "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 client id. If the `group_fnc` is not specified, the default grouping is based on subscription parameters.

3.7.16.4 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 10g 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 10g. 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 API. See the `SetPublicationItemMetadata` API in [Section 3.7.16.3, "Usage"](#).
4. The `ShrinkSharedMaps` consolidator 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.

3.8 Synchronization Errors and Conflicts

With the Mobile Server, a compatibility error with Oracle database advanced synchronization occurs when the client updates a row at the same time that the server deletes it. All other errors, such as nullity violations or foreign key constraint violations, are synchronization errors.

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. Later, Mobile Server database administrators can change these transaction operations and re-execute or purge them from the error queue.

A Mobile Server synchronization conflict occurs if:

- The client and the server update the same row.
- The client and server create rows with the same primary key values.
- The client deletes the same row that the server updates.

See [Section 3.8.3, "Resolving Conflicts Using the Error Queue"](#) for more information on conflict resolution techniques.

3.8.1 Versioning

The Mobile Server uses internal versioning to detect synchronization conflicts. A version number is maintained for each client record as well as for each server record. When a client's changes are applied to the server, the Mobile Server will detect version mismatches and resolve conflicts according to winning rules.

3.8.2 Winning Rules

The Mobile Server uses winning rules to automatically resolve synchronization conflicts. The following winning rules are supported:

- Client wins
- Server wins

When the client wins, the Mobile Server automatically applies client changes to the server. When the server wins, the Mobile Server automatically composes changes for the client.

You can customize the Mobile Server's conflict resolution mechanism by setting the winning rule to "Client Wins" and attaching BEFORE INSERT, UPDATE, and DELETE triggers to database tables. The triggers compare old and new row values and resolve client changes as specified.

3.8.3 Resolving Conflicts Using 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 she may attempt to re-apply the transaction via the `ExecuteTransaction` API call. The administrator may also purge the error queues through the `PurgeTransaction` API call. The Mobile Server error queue is C\$EQ, the data is stored in CEQ\$.

3.8.3.1 Execute Transaction

The execute transaction function re-executes transactions in the Mobile Server error queue.

Syntax

```
public static void ExecuteTransaction
(String clientid,
 long tid) throws Throwable
```

The parameters for `ExecuteTransaction` are listed in [Table 3–44](#):

Table 3–44 *ExecuteTransaction Parameters*

Parameter	Description
<code>clientid</code>	The Mobile Sync Client name.
<code>tid</code>	The transaction ID. These are generated strings which appear in the error queue.

Example

```
Consolidator.ExecuteTransaction("DAVIDL", 100002);
```

3.8.3.2 Purge Transaction

The purge transaction function purges a transaction from the Mobile Server error queue.

Syntax

```
public static void PurgeTransaction
    (String clientid,
     long tid) throws Throwable
```

The parameters for `PurgeTransaction` are listed in [Table 3–45](#):

Table 3–45 *PurgeTransaction Parameters*

Parameter	Description
<code>clientid</code>	The Mobile Server user name.
<code>tid</code>	The transaction ID. These are generated strings which appear in the error queue.

Example

```
Consolidator.PurgeTransaction("DAVIDL", 100001);
```

3.8.4 Space Constraints

All synchronization parameters must be set in the `POLITE.INI` or `polite.txt` file. To counter space constraints for the storage card on the WinCE platform, you can utilize the `Temp` directory. To begin using the `TEMP` directory, add the following entry under the `ALL DATABASES` section.

```
TEMPDIR=\Storage Card\Temp
```

3.9 Mapping Datatypes Between the Oracle Server and Clients

The Oracle database and Oracle Database Lite tables that the Mobile Server synchronizes must use compatible datatypes. Oracle database datatypes are compatible with Oracle Database Lite datatypes.

3.9.1 Oracle Database Lite Datatypes

All Oracle Database Lite based snapshots are created by the Mobile Sync during synchronization. The Mobile Server automatically selects Oracle Database Lite datatypes depending on data precision in the Oracle database. The data conversion

values are listed in [Table 3–46](#). The table lists the Oracle database datatypes in the left column and displays the Oracle Database Lite datatypes across the top row.

For Oracle Database Lite Datatypes, see Appendix B in the *Oracle Database Lite SQL Reference*.

Table 3–46 Oracle Database Lite Datatypes

Oracle Database Datatypes	1 B	2 B	4 B	FLOAT	DOUBLE	NUMBER	DATE TIME	LONG- VAR BINARY	VARCHAR
INTEGER	Y	Y	Y	Y	Y	Y	N	N	N
VARCHAR2	N	N	N	N	N	Y	N	N	Y
VARCHAR	N	N	N	N	N	Y	N	N	Y
CHAR	N	N	N	N	N	Y	N	N	Y
SMALLINT	Y	Y	Y	Y	Y	Y	N	N	N
FLOAT	Y	Y	Y	Y	Y	Y	N	N	N
DOUBLE PRECISION	Y	Y	Y	Y	Y	Y	N	N	N
NUMBER	Y	Y	Y	Y	Y	Y	N	N	N
DATE	N	N	N	N	N	Y	Y	N	N
LONG RAW	N	N	N	N	N	Y	N	Y	N
LONG	N	N	N	N	N	Y	N	N	Y
BLOB	N	N	N	N	N	Y	N	Y	N
CLOB	N	N	N	N	N	Y	N	N	N

"Y" indicates *unconditionally supported* and "N" indicates *not supported*. In the first three columns that are labeled (because of space limitations), 1 B represents **TINYINT**, 2 B represents **SMALLINT**, and 4 B represents **INTEGER**.

Developing Mobile Web Applications

This document describes how to develop and test web applications. Topics include:

- [Section 4.1, "Setting up the Mobile Client"](#)
- [Section 4.2, "Developing and Testing the Application"](#)

4.1 Setting up the Mobile Client

To install and set up the Mobile Client, see [Chapter 12, "Building Mobile Web Applications: A Tutorial"](#), [Section 12.6.1, "Step 1: Installing the Mobile Client for Web-to-Go"](#).

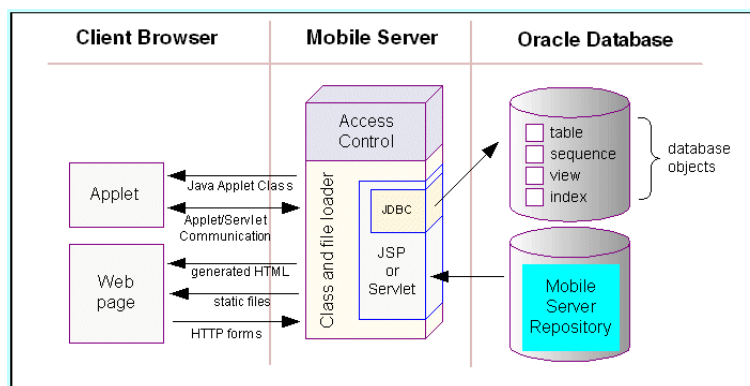
4.2 Developing and Testing the Application

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 4-1](#) displays the development architecture of the Mobile Server and the Oracle database.

Figure 4-1 Development Architecture



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

- [Section 4.2.1, "Building Web-to-Go Applications"](#)
- [Section 4.2.2, "Application Roles"](#)
- [Section 4.2.3, "Developing JavaServer Pages"](#)
- [Section 4.2.4, "Developing Java Servlets for Web-to-Go"](#)
- [Section 4.2.5, "Using Web-to-Go Applets"](#)
- [Section 4.2.6, "Developing Applet JDBC Communication"](#)
- [Section 4.2.7, "Developing Applet Servlet Communication"](#)
- [Section 4.2.8, "Debugging Web-to-Go Applications"](#)
- [Section 4.2.9, "Customizing the Workspace Application"](#)
- [Section 4.2.10, "Using the Mobile Server Admin API"](#)

4.2.1 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 4–1](#) provides examples of each database component type.

Table 4–1 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

4.2.1.1 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).

4.2.1.2 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.

4.2.1.3 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.

4.2.1.4 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.

4.2.2 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.

4.2.3 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.

4.2.3.1 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.

4.2.3.2 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.

4.2.4 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.

4.2.4.1 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`.

4.2.4.2 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.

4.2.4.3 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();
    }
}
```

4.2.4.3.1 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 "Using the `oracle.lite.web.applet` Package" in the *Web-to-Go API Specification*.

4.2.4.3.2 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");
    .
    .
    .
}
```

4.2.4.3.3 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() ;
```

4.2.4.3.4 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();
.
.
.
}

```

4.2.4.4 Running a Servlet

After you create the Web-to-Go servlet, you must run the servlet.

4.2.4.4.1 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.

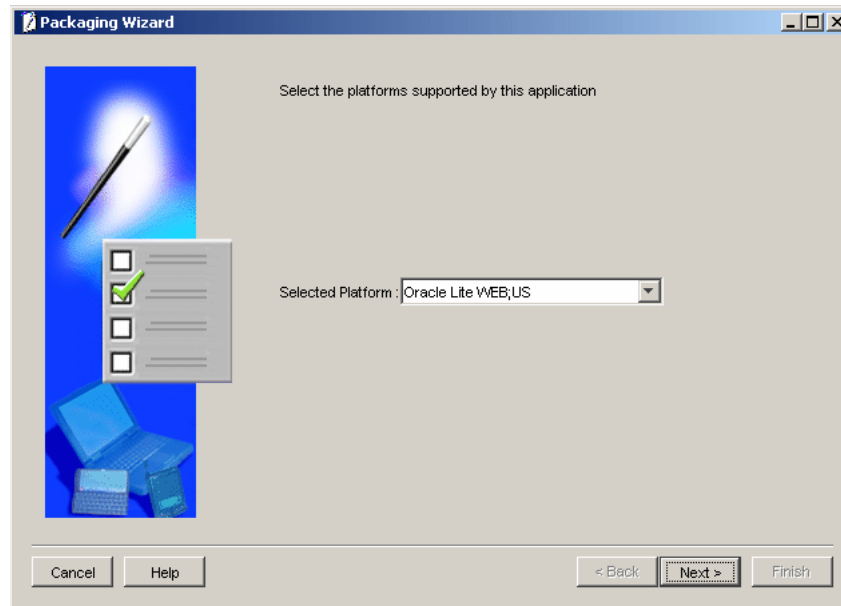
The Mobile Development Kit for Web-to-Go includes the Packaging Wizard, a tool for registering applications and servlets. You can invoke the Packaging Wizard by entering the following at the command line.

```
C:\> wtgpack -d
```

Initially, you select whether to create a new application or to continue work on an existing application.

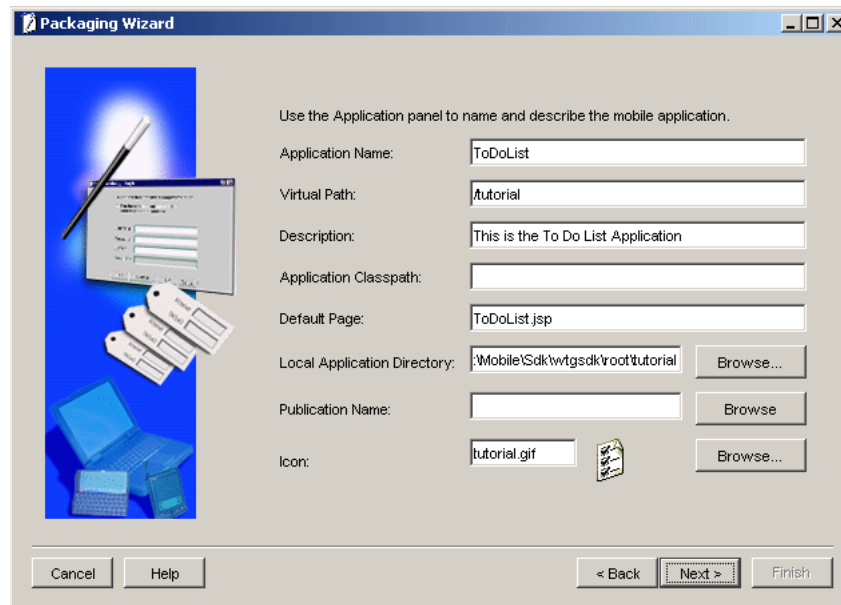
[Figure 4-2](#) displays the Make a Selection dialog.

Figure 4–2 Make a Selection Panel



After you make your selection and click **OK**, the Applications dialog appears. [Figure 4–3](#) displays the Applications dialog.

Figure 4–3 Applications Panel



For detailed instructions on how to use the Packaging Wizard, see [Chapter 12, "Building Mobile Web Applications: A Tutorial"](#).

4.2.4.4.2 The webtogo.ora File The configuration information for the web server and the Packaging Wizard is stored in the **webtogo.ora** file.

[Table 4–2](#) describes webtogo.ora parameters.

Table 4–2 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 <code>ROOT_DIR</code> 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*.

4.2.4.4.3 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`.

4.2.4.4.4 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
<Oracle_home>\mobile\classes\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 4.2.5, "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
```

4.2.4.4.5 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 4–3](#) describes Mobile Client Web Server properties.

Table 4–3 Property Controls

Property	Definition
config_file	The configuration file to use. For more information, refer Section 4.2.4.4.2, "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 <code>webtogo.ora</code> . For more information, refer Section 4.2.4.4.2, "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");
```

4.2.4.4.6 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")
```

4.2.4.5 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 4.2.8.1, "Running Sample 1 Using Oracle9i JDeveloper"](#).

4.2.4.6 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/x@jdbc:polite:webtogo
```

4.2.5 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.

4.2.5.1 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.

4.2.5.2 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.

4.2.5.2.1 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`.

4.2.5.2.2 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();
}
}

```

4.2.6 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 4.2.5.1, "Creating the Web-to-Go Applet"](#).

4.2.6.1 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;
}

```

4.2.6.2 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()
```

4.2.7 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.

4.2.7.1 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();
    }
}
```

4.2.7.1.1 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;
}
```

4.2.7.1.2 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.openConnection();
    ..
    ..
    ..
    // 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();
}
```

4.2.7.1.3 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);  
}
```

4.2.8 Debugging Web-to-Go Applications

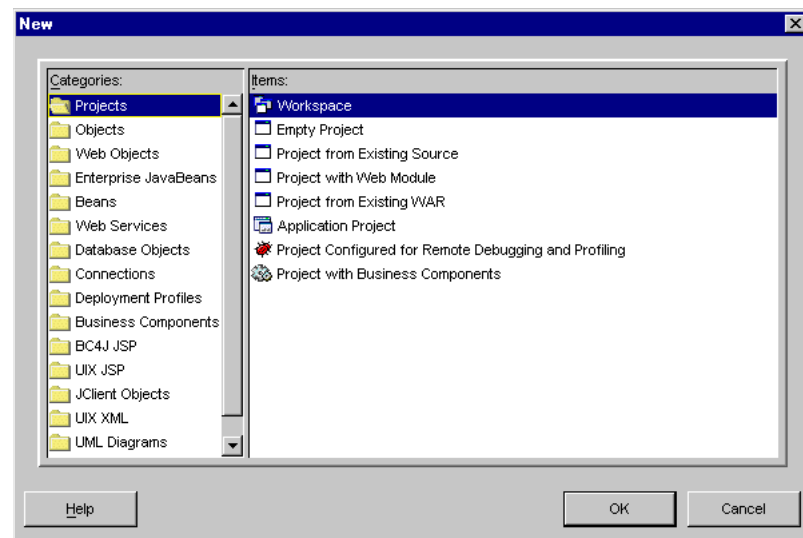
You can run Web-to-Go applications inside a Java debugger if you have already installed the Mobile Development Kit for Web-to-Go and a Java debugger, such as the Oracle9i JDeveloper, Borland's JBuilder, or Visual J++. The example in this section assumes you are using Oracle9i JDeveloper. However, most of the information provided is also relevant to other debuggers.

4.2.8.1 Running Sample 1 Using Oracle9i JDeveloper

This section discusses how to configure the Oracle9i 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 Oracle9i JDeveloper, consult the online help in Oracle9i JDeveloper and Oracle9i JDeveloper's documentation.

4.2.8.1.1 Creating a Debug Project To create a new debug project in Oracle9i JDeveloper, perform the following steps.

1. Start Oracle9i JDeveloper.
2. To create a new project in Oracle9i JDeveloper, click **File**, then click **New** (assuming you have defined a workspace in Oracle9i JDeveloper).
3. From the **Directories** menu in the left panel, select **Projects**, as displayed in [Figure 4-4](#), then select **Empty Project**.

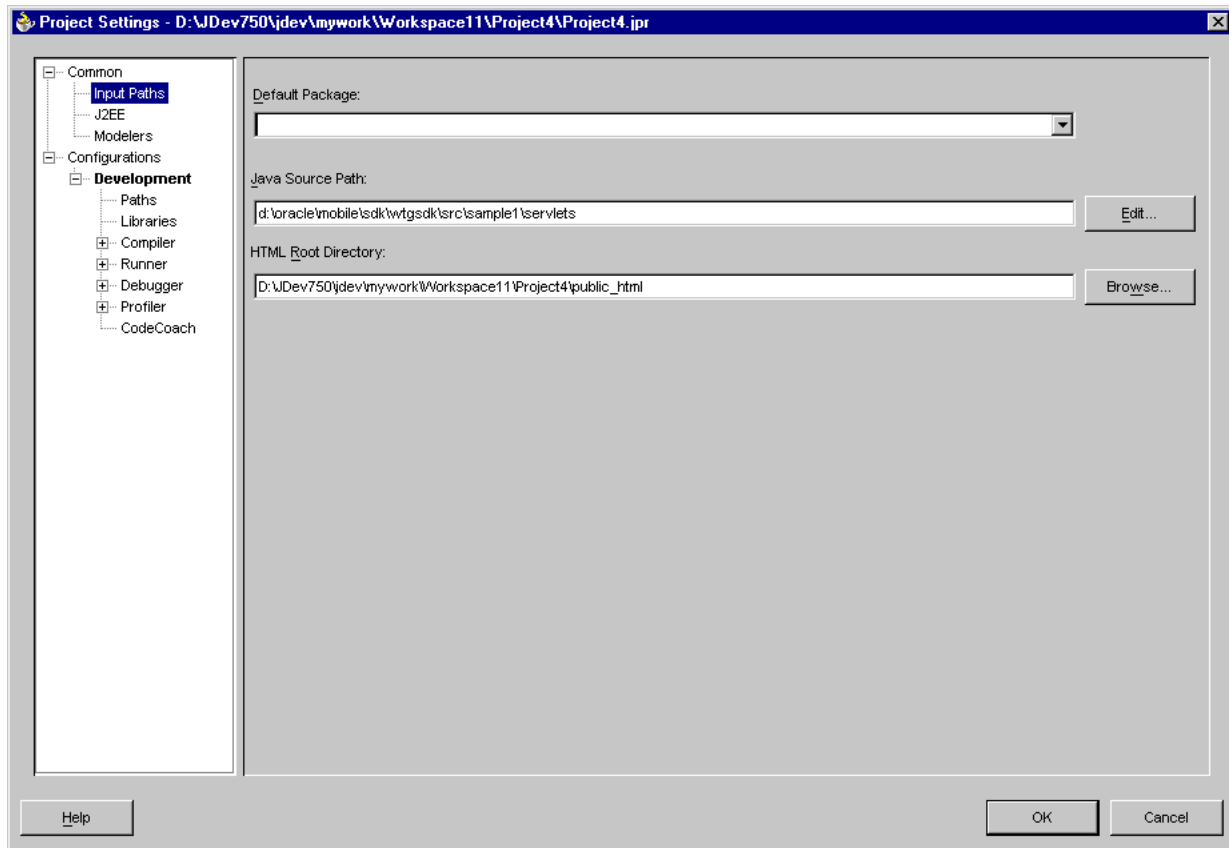
Figure 4–4 Creating a New Project

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 4–5](#).

```
<Oracle_home>\mobile\sdk\wtgSDK\src\sample1\servlets
```

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

Figure 4–5 Project Settings - Input Paths



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
```

4.2.8.1.2 Creating a Library Oracle9i 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.

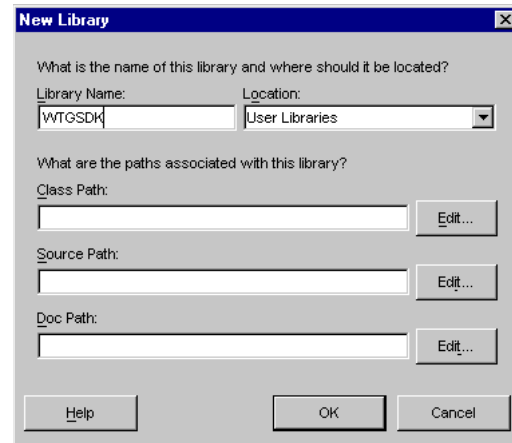
```
<Oracle_home>\mobile\classes\ojsp.jar  
<Oracle_home>\mobile\classes\olite40.jar  
<Oracle_home>\mobile\sdk\bin\webtogo.jar  
<Oracle_home>\mobile\classes\servlet.jar  
<Oracle_home>\mobile\classes\xmlparser.jar  
<Oracle_home>\mobile\classes\classgen.jar  
<Oracle_home>\mobile\classes\wtgpack.jar
```

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 4–6](#). In the **Library Name** field, enter WTGSDK.

Figure 4–6 The New Library Dialog



3. Click **Edit...** next to the Class Path field. The **File** dialog appears.
4. From the appropriate directory, select the six `.jar` files that are listed above.
5. To add the files, click **OK**.

4.2.8.1.3 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 Oracle9i 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\sd\wtgSDK\src\sample1\servlets`, and click **Open**.
3. Also, add the file `RunWebServer.java`, which is located in the directory `<Oracle_home>\mobile\sd\wtgSDK\src`, to the project.
4. A dialog appears prompting you to update the project source path. Click **No**.

4.2.8.1.4 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 Oracle9i JDeveloper.

4.2.8.1.5 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.

4.2.9 Customizing the Workspace Application

The Mobile Development Kit for Web-to-Go includes a set of APIs that contain a basic Web-to-Go workspace application. Developers can use these APIs to replace the standard Web-to-Go workspace application with a customized version. 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
```

After developing the customized Web-to-Go workspace application, the developer must create an Oracle Database Lite database called `webtogo` and load the newly created 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 the file `crclient.bat`, which is included in the sample Web-to-Go workspace application.

The developer must then 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 4.2.9.1, "Web-to-Go Parameters"](#).

As a developer, you must load the `webtogo.odp` 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 itself 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.

To instruct the Mobile Server to use the new Web-to-Go workspace application, the administrator must then modify the `webtogo.ora` file on the server. For the correct parameter settings in the `webtogo.ora`, refer the section [Section 4.2.9.1, "Web-to-Go Parameters"](#).

4.2.9.1 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 4–4 describes `webtogo.ora` parameter settings.

Table 4–4 *Setting webtogo.ora Parameters*

Parameter	Setting
CUSTOM_WORKSPACE	YES
CUSTOM_DIRECTORY	Repository directory of the Web-to-Go workspace application. For example, <code>/myworkspace</code> .
DEFAULT_PAGE	The entry point of the Web-to-Go workspace application. For example, <code>myfirstpage.html</code> .
CUSTOM_FIRSTSERVLET	The name of the servlet that you want to use in your customized workspace. For example, <code>CUSTOM_FIRSTSERVLET=HelloWorld;/hello</code>

Note: Web-to-Go supports only one workspace application per Mobile Server.

4.2.9.2 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 4–5 describes scripts available for JSP compilation.

Table 4–5 *Scripts for JSP Compilation*

Script Name	Description
<code>compile.bat</code>	Compiles <code>.jsp</code> files and copies the class files to the <code>myworkspace/out</code> directory.
<code>crclient.bat</code>	Copies all files in the <code>myworkspace/out</code> directory into the <code>webtogo.odb</code> file.
<code>crserver.bat</code>	Copies all files in the <code>myworkspace/webtogo</code> directory to the Mobile Server Repository, including the <code>webtogo.odb</code> and <code>webtogo.ora</code> files.

4.2.10 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*.

Native Application Development

This document discusses mobile application development for native platforms. The discussion covers the following topics:

- [Section 5.1, "Supported Platforms"](#)
- [Section 5.2, "Java Support"](#)
- [Section 5.3, "Data Source Name"](#)
- [Section 5.4, "Mobile Sync Application Programming Interfaces \(APIs\)"](#)
- [Section 5.5, "Using the Packaging Wizard"](#)

5.1 Supported Platforms

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

- Microsoft Windows NT/2000/XP
- Windows CE

The following Windows CE chipsets are supported:

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

5.2 Java Support

[Table 5–1](#) lists the Java support provided in Oracle Database Lite 10g. The heading row in the table identifies each of the four platforms in which the Java support is available.

Table 5–1 Java Support

Category	Windows 32 Web	Windows 32 Native	Windows CE	Linux (1)
JDBC	Yes Oracle Database Lite offer three JDBC drivers. Refer the section JDBC Drivers given below.	Yes	Yes	Yes On Linux, only JDBC and ODBC access is supported.

Table 5–1 (Cont.) Java Support

Category	Windows 32 Web	Windows 32 Native	Windows CE	Linux (1)
Java SP/Triggers	Yes Java SP/Triggers are not supported in the Web-to-Go application model. However Java SP can be replicated using the Consolidator API.	Yes	NA	Yes
Java Server Pages	1.1	NA	NA	NA
Java Servlet	2.2	NA	NA	NA
BC4J	Yes Latest version of Oracle JDeveloper 10g.	NA	NA	NA
Struts	Yes	NA	NA	NA

JDBC Drivers

Oracle Database Lite offers the following JDBC drivers.

- Embedded (native) JDBC driver: JDBC 1.2.2 compliant. Allows Java applications to communicate directly with Oracle Lite's database engine. Oracle Lite provides a limited number of extensions specified by JDBC 2.0. These extensions are compatible with the Oracle Database JDBC implementation.
- Type 2 driver.
- Type 4 driver : 100% Java implementation. Requires the multi-user database version.

5.3 Data Source Name

In Windows 32, the Consolidator on the client side creates the data source name (DSN) as <username_dbname> after the first synchronization. The Consolidator program takes the values for <username_dbname> from the publication.

In Windows CE and Palm, DSN = <dbname>.

The DSN does not change after the Consolidator creates it.

5.4 Mobile Sync Application Programming Interfaces (APIs)

There are three methods that a developer can use to create native applications which invoke synchronization using the underlying libraries, **ocapi.dll**. These APIs are provided for application development where a different approach is required than that provided by the Mobile Sync client, **msync.exe**. The Palm OS interface is part of SODA.

- [Section 5.4.1, "COM Interface"](#)
- [Section 5.4.2, "C/C++ Interface"](#)

Note: There are currently no client-side synchronization programming interfaces for Sun SPARC Solaris. It is recommended to use the Windows Operating System for programming with these interfaces.

5.4.1 COM Interface

The COM Interface is used to program applications that can start the synchronization process and enable a variety of settings. This interface is modular and extensible. It uses the **ocapi.dll** through a wrapper style interface. The interface is designed to allow applications to be written in Visual Basic, but allows other programming methods supported by the COM interface including VBScript.

5.4.1.1 Features and Components

The COM Interface supports the following features:

- Enables users to start the synchronization process.
- Track progress of the synchronization process.
- Enables setup of client-side user profiles containing data such as user name, password, and server.
- Assign table level synchronization options.
- Allow a choice of transport.

COM Interface API and samples are now installed in the <ORACLE_HOME>\Mobile\SDK\Examples\mysncCom subdirectory. The following classes are contained in the **mSync_com.dll** library:

- [Section 5.4.1.2, "ISync Interface"](#)
- [Section 5.4.1.3, "ISyncOption Interface"](#)
- [Section 5.4.1.7, "ISyncProgressListener Interface"](#)

The interface is contained in the MSync library. When using the ISync interface, you should use **MSync.ISync** as the interface name.

5.4.1.2 ISync Interface

The ISync interface, `MSync.ISync` allows the user to initiate the synchronization process. The format for the ISync interface is listed in a table below.

[Table 5–2](#) lists Sync Interface Abstract Method names and their corresponding description.

Table 5–2 ISync Interface Abstract Methods

Name	Description
<code>HRESULT doSync()</code>	Start the synchronization process. This blocks access until the synchronization process is completed.
<code>void abort()</code>	Aborts the current synchronization. This can be called from a progress listener callback.
<code>HRESULT setOption(ISyncOption *syncObj)</code>	Sets the pointer to the <code>SyncOption</code> to use for the next synchronization. If this function is not called before <code>doSync()</code> , the last saved option will be used.

Example

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

```
Dim sync As Msync.sync
Set sync = CreateObject("MSync.Sync")
sync.DoSync
```

In case no `SyncOption` is used, the interface loads the last saved information to perform synchronization.

5.4.1.3 ISyncOption Interface

The `ISyncOption` class `MSync.SyncOption` defines the parameters of the synchronization process. It can be constructed manually. Alternatively, you can use the data that is loaded or saved from the user profile. The public methods for the `ISyncOption` class are listed in a table below.

[Table 5–3](#) lists `SyncOption` Public Method names and their corresponding description.

Table 5–3 ISyncOption Public Methods

Name	Description
<code>void load()</code>	Loads the profile of the last synchronization user.
<code>void save()</code>	Saves settings to the user profile.
<code>void getPublication</code> (<code>BSTR app_name</code> , <code>BSTR</code> * <code>pub_name</code>)	Uses the Web-to-Go application name and returns the publication name.
<code>void setSyncFlag</code> (<code>BSTR pub_name</code> , <code>BSTR tbl_name</code> , <code>short syncFlag</code>)	Sets selective sync on table level. Passing <code>pub_name</code> , null <code>tbl_name</code> , <code>syncFlag</code> = 0 will turn off <code>syncFlag</code> for everytable in that publication. Passing <code>pub_name</code> , <code>tbl_name</code> , <code>syncFlag</code> = 1 will turn on <code>syncFlag</code> for that table.

The public properties for the `ISyncOption` class are listed in a table below.

[Table 5–4](#) lists names of `ISyncOption` Public Properties and their corresponding description.

Table 5–4 ISyncOption Public Properties

Name	Description
<code>username</code>	Name of the user.
<code>password</code>	User's password.
<code>syncParam</code>	Synchronization preferences. For more information, see Section 5.4.1.5, "COM Interface SyncParam Settings" .
<code>transportType</code>	Type of transport to use. Currently, only "HTTP" type is supported.
<code>transportParam</code>	Parameters of the transport.
<code>BSTR app_name(in)</code>	Web-to-Go application name.
<code>BSTR& pub_name(out)</code>	Publication name.

Example

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
```

5.4.1.4 Selective Synchronization

This feature allows the mobile application to select the way specific tables are synchronized.

You can implement selective synchronization at the publication level and the table level by using the `mSync.SyncOption` interface to determine which publication and publication items need to be synchronized. The list of tables therefore can be changed dynamically during runtime allowing the application developer to programmatically control selective synchronization.

You can use the following method to set selective synchronization:

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

The first parameter, `pub_name`, which is for the publication name, is optional. If it is set to null, the parameter means all publications.

The second parameter, `tbl_name`, which is for the table name (in the form `<client database>.<table name>`), is optional. If it is set to null, the parameter means all tables.

The third parameter, `syncFlag`, which is for the synchronization flag, is set to 1 to turn ON the `syncFlag` or to 0 to turn OFF the `syncFlag`.

See the sample code below for an illustration.

Sample Code:

The following sample code shows how to turn OFF synchronization for all but one table. The table name in this sample is `ORD_DETAIL`. Note that first the synchronization flag is set to 0 and then in the next line of code it is set to 1 for the specified table on which selective synchronization is to be implemented.

```
Dim syncOpt As MSync.SyncOption

syncOpt = CreateObject("MSync.SyncOption")
syncOpt.setSyncFlag("", "", 0) //Turn off sync flag for all tables.
syncOpt.setSyncFlag("", "OrdersODB.ORD_DETAIL", 1) //Turn on sync
```

flag only for the OrdersODB.ORD_DETAIL table.

5.4.1.5 COM Interface SyncParam Settings

The `syncParam` is a string that 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 a table below.

Table 5–5 COM Interface SyncParam Settings

Name	Value/Options	Description
"reset"	N/A	Clears all entries in the environment before applying any remaining settings.
"security"	"SSL" "CAST5"	Use the appropriate selection to choose either SSL or CAST5 stream encryption.
"pushonly"	N/A	Use this setting to 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" "ReceiveOnly"	"SendOnly" is the same as "pushonly". "ReceiveOnly" allows no changes to be posted to the server.
"noNewPubs"	N/A	This setting prevents creation of any new publications, since the last synchronization from being sent, and only synchronizes data from current publications.
"fullrefresh"	N/A	Forces a complete refresh.
"clientDBMode"	"EMBEDDED" "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 illustrates selective synchronization.

```
//turn off the syncFlag for all the tables
syncOpt.setSyncFlag("", "", 0)
```

```
//turn on the syncFlag for table OrdersODB.ORD_DETAIL
```



```
syncOpt.setSyncFlag("", "OrdersODB.ORD_DETAIL", 1)
```

5.4.1.6 COM 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 a table below.

[Table 5–6](#) lists names, values, and the corresponding description of COM Interface `TransportParam` parameters.

Table 5–6 *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.

Example

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.

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

5.4.1.7 ISyncProgressListener Interface

`ISync` implements a connection point container to allow the synchronization status information to be tracked. `ISyncProgressListener` must be implemented to return updates from the `ISync` interface. The abstract method for the `ISyncProgressListener` is listed in a table below.

[Table 5–7](#) lists the name and corresponding description of the `ISyncProgressListener` Abstract Method.

Table 5–7 *ISyncProgressListener Abstract Method*

Name	Description
<code>HRESULT progress([in] int progressType, int param1, int param2);</code>	Called by the synchronization engine when new progress information is available. The <code>progressType</code> is set to one of the progress type constants defined in the <code>ISyncProgressListener Constants</code> 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 <code>progressType</code> .

The `ISyncProgressListener` is an interface that allows progress updates to be trapped during synchronization. The names of constants which report the synchronization progress are listed in a table below.

[Table 5–8](#) lists names and the corresponding progress type description of `ISyncProgressListener` Constants.

Table 5–8 *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.
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.

Example

The following Visual Basic code 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 call back 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
```

5.4.2 C/C++ Interface

The C/C++ Interface consists of function calls and a control structure, the definitions for which can be found in `ocapi.h` and `ocapi.dll` which are located in the `<Oracle_home>\Mobile\bin` directory. This API allows an application to initiate and monitor synchronization with a database from a client application rather than requiring that it

be started from the Mobile Sync application. The default transport mechanism is HTTP, but other forms of transport can be specified if they are available.

An example C++ program, a makefile, and dependent files are given in the <Oracle_home>\Mobile\Sdk\Examples\msync\src directory. Peruse the source code in SimpleSync.cpp to see how this interface is used. The executable SimpleSync.exe is in the <Oracle_home>\Mobile\Sdk\Examples\msync\bin directory.

5.4.2.1 ocSessionInit

This function is used to initialize the synchronization environment.

Syntax

```
int ocSessionInit( ocEnv *env );
```

The parameter for ocSessionInit function is listed in a table below.

[Table 5–9](#) lists the ocSessioninit parameter and its description.

Table 5–9 ocSessionInit Parameters

Name	Description
env	Pointer to an ocEnv structure buffer to hold the return synchronization environment.

Comments

This call initializes the ocEnv structure and restores any user settings that are saved in the last ocSaveUserInfo() call. A pointer to an ocEnv structure is passed as a parameter, and should be allocated by the caller. If the caller wants to overwrite user preference information after the ocSessionInit() call, it can be done by calling ocSaveUserInfo(). The caller must allocate memory for the ocEnv structure.

5.4.2.2 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 5–10](#) lists the ocSessionTerm parameter and its description.

Table 5–10 ocSessionTerm Parameters

Name	Description
env	Pointer to the environment structure returned by ocSessionInit.

Comments

De-initializes all the structures and memory created by the ocSessionInit() call. Users must ensure that they are always called in pairs.

5.4.2.3 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 5–11](#) lists the `ocSaveUserInfo` parameter and its description.

Table 5–11 *ocSaveUserInfo Parameters*

Name	Description
env	Pointer to the synchronization environment.

Comments

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.

1. Username
2. Password
3. SavePassword
4. NewPassword
5. Priority
6. Secure
7. PushOnly
8. SyncApps
9. SyncNewPublication

For more information on how to use these fields, see [Section 5.4.2.7, "C/C++ Data Structures"](#).

5.4.2.4 ocDoSynchronize

Starts the synchronization process.

Syntax

```
int ocDoSynchronize( ocEnv *env );
```

The parameter for `ocDoSynchronize` function is listed in a table below.

[Table 5–12](#) lists the name and description of the `ocDoSynchronize` parameter.

Table 5–12 *ocDoSynchronize Parameters*

Name	Description
env	Pointer to the synchronization environment.

Comments

This starts the synchronization cycle. A round trip synchronization is activated if `syncDirection` is `OC_SENDRECEIVE` (default). If `syncDirection` is `OC_SENDONLY`, only the upload, or send operation, is performed. If `syncDirection` is `OC_RECEIVEONLY`, only the download, or receive operation is performed. Performing an upload-only synchronization is useful if the client does not want to download data from the server.

Return value of 0 indicates that the function has been executed successfully. Otherwise, the value is an error code.

5.4.2.5 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.

When this option is used, it must occur before `ocDoSynchronize`.

The default `sync_flag` setting for `ocSetTableSyncFlag` is TRUE (1) for all the tables. By default, all the tables are flagged to be synchronized. If you want to selectively synchronize specific tables, you must first disable the default setting for synchronizing all the tables and then enable the selective synchronization for the specific tables that you want to synchronize.

Syntax

```
ocSetTableSyncFlag(ocEnv *env, const char* publication_name,
const char* table_name, short sync_flag)
```

The parameters for the `ocSetTableSyncFlag` function are listed in a table below.

[Table 5–13](#) lists the name and description of the `ocSetTableSyncFlag` parameter.

Table 5–13 *ocSetTableSyncFlag Parameters*

Name	Description
<code>env</code>	Pointer to the synchronization environment.
<code>publication_name</code>	The name of the publication which is being synchronized. If the value for <code>publication_name</code> is "NULL", it means all publications in the database. This string is same as <code>client_name_template</code> parameter of the Consolidator <code>CreatePublication</code> 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" .
<code>table_name</code>	This is the name of the snapshot. It is the same as the name of the store, the third parameter of <code>CreatePublicationItem()</code> . For more information, see Section 3.5.5, "Creating Publication Items" in Chapter 3, "Synchronization" .
<code>sync_flag</code>	If <code>sync_flag</code> is set to "1", you must synchronize the publication. If <code>sync_flag</code> is set to "0", then do not synchronize. The value for <code>sync_flag</code> is not stored persistently. Each time before <code>ocDoSynchronize()</code> , you must call <code>ocSetTableSyncFlag()</code> .

Comments

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)
```

5.4.2.6 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 5–14](#) below. The table lists the name of the `ocGetPublication` parameter and provides a description of it.

Table 5–14 *ocGetPublication Parameters*

Name	Description
<code>ocEnv* env</code>	Pointer to an <code>ocEnv</code> structure buffer to hold the return synchronization environment.
<code>const char* application_name(in)</code>	This is the name of the application.
<code>char* buf(out)</code>	The buffer where the publication name will be stored.
<code>int buf_len(in)</code>	The buffer length. It must be at least 32 bytes.

Comments

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.

Example

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"
charbuf[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;
}

```

5.4.2.7 C/C++ Data Structures

Two data structures are part of the Mobile Sync API, `ocEnv` and `ocTransportEnv`.

5.4.2.7.1 `ocEnv`

The `ocEnv` is the data structure used by all the Mobile Sync module functions to hold internal memory buffers and state information. Before using the structure, the application must pass it to `ocSessionInit` to initialize the environment. The parameters for the structure appear in a table below.

[Table 5–15](#) lists the field name, type, usage, and corresponding description of the `ocEnv` Structure field parameters.

Table 5–15 *ocEnv Structure Field Parameters*

Field	Type	Usage	Description
Username	char[MAX_USERNAME]	Caller MUST set these fields before calling <code>ocSessionInit</code> .	Name of the user to authenticate.
Password	char[MAX_USERNAME]	Caller MUST set these fields before calling <code>ocSessionInit</code> .	User password (clear text).
NewPassword	char[MAX_USERNAME]	Caller can set these fields optionally after calling <code>ocSessionInit</code> .	If first character of this string is not null, in otherwords (char) 0, this string will be sent to the server to request it to change the user's password; the password change will be effective on the next sync session.
SavePassword	Short	Caller can set these fields optionally after calling <code>ocSessionInit</code> .	If set to 1, the password in the password field will be saved locally and will be loaded the next time <code>ocSessionInit</code> is called.
AppRoot	char[MAX_USERNAME]	Caller can set these fields optionally after calling <code>ocSessionInit</code> .	Directory where the application will be copied to. If first character is null, then it will use the default directory.

Table 5–15 (Cont.) ocEnv Structure Field Parameters

Field	Type	Usage	Description
Priority	Short	Caller can set these fields optionally after calling <code>ocSessionInit</code> .	0= OFF (default) 1= ON
Secure	Short	Caller can set these fields optionally after calling <code>ocSessionInit</code> .	If set to 0, no security on transport. If set to <code>OC_DATA_ENCRYPTION</code> , use CAST5 synchronization. If set to <code>OC_SSL_ENCRYPTION</code> , use SSL synchronization (Win32 only).
SyncDirection	Enum	Caller can set these fields optionally after calling <code>ocSessionInit</code> .	If set to 0 (<code>OC_SENDRERECEIVE</code>) then sync is bi-directional (default). If set to <code>OC_SENDOONLY</code> , then pushes changes only to the server. This is to stop the sync after the local changes are collected and sent. Useful for sync that requires the engine to separate the different stages (like floppy based). If set to <code>OC_RECEIVEONLY</code> , 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	Must be set before calling <code>ocSessionInit</code> .	If set to 0 (<code>OC_BUILDIN_HTTP</code>), then use HTTP built-in transport driver. If set to <code>OC_USER_METHOD</code> , then use user provided transport functions.
ExError	ocError	Read only information updated by OCAPI.	Extended error code - either OS or OKAPI error code.
TransportEnv	ocTransportEnv		Transport buffer. See Section 5.4.2.7.2, "ocTransportEnv" .
ProgressProc	FnProgress	Caller can set these fields optionally after calling <code>ocSessionInit</code> .	If not null, points to the callback for progress listening.
TotalSendDataLen	Long	Read only information updated by OCAPI.	Set by OCAPI informing transport the total number of bytes sent; set before first <code>fnSend()</code> is called.
TotalRecieveDataLen	Long	Read only information updated by OCAPI.	Set by OCAPI information transport total number of bytes to receive; should be set at first <code>fnReceive()</code> call.
UserContext	Void*	Caller can set these fields optionally after calling <code>ocSessionInit</code> .	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	Caller can set these fields optionally after calling <code>ocSessionInit</code> .	If set to 1, then only push changes to the server.

Table 5–15 (Cont.) ocEnv Structure Field Parameters

Field	Type	Usage	Description
SyncApps	Short	Caller can set these fields optionally 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	Caller can set these fields optionally after calling ocSessionInit.	If set to 1 (default), then 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	Caller can set these fields optionally after calling ocSessionInit.	If set to OC_DBMODE_EMBEDDED (default), it will use local Oracle Database Lite ODBC driver. If set to OC_DBMODE_CLIENT, it will use the Branch Office driver.
SyncTimeLog	Short	Caller can set these fields optionally after calling ocSessionInit.	If set to 1, log sync start time is recorded in the "conscli.odt" file.
UpdateLog	Short	Caller can set these fields optionally after calling ocSessionInit.	Debug only. If set to 1, logs server side insert and update row information to the publication's odb.
Options	Short	Caller can set these fields optionally after calling ocSessionInit.	Debug only. A bitset of the following flags: <ul style="list-style-type: none"> ■ OCAPI_OPT_SENDEMADATA Sends meta-info to the server. ■ or OCAPI_OPT_DEBUG Enables debugging messages. ■ OCAPI_OPT_DEBUG_F Saves all bytes sent and received for debugging. ■ OCAPI_OPT_NOCOMP Disables compression. ■ OCAPI_OPT_ABORT If set, OCAPI will try to abort the current sync session. ■ OCAPI_OPT_FULLREFRESH Forces OCAPI to purge all existing data and do a full refresh.

The environment structure also contains fields that the caller can update to change the way Mobile Sync module functions work.

```
typedef struct ocEnv_s {
    // User infos
    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_SENDFILE,                // 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_BUILDDIN_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
} ocEnv;

```

5.4.2.7.2 ocTransportEnv

This structure is used to override built-in transport functions. By providing the list of functions in the structure, applications can define their own implementation for the transport layer used by the synchronization engine.

```

typedef struct ocTransportEnv_s {
void* ocTrInfo;              // transport internal context
                                // for built-in Http, mapped to ocTrHttp
connectProc fnConnect;       // plug-in callback to establish a connection from
                                // device to server
disconnectProc fnDisconnect; // plug-in callback to dismantle connection
                                // from device to server
sendProc fnSend;             // plug-in callback to send data
receiveProc fnReceive;       // plug-in callback to receive data
}ocTransportEnv;

```

5.5 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 Control Center. The Packaging Wizard also enables you to create SQL scripts which can be run to create base tables in the Oracle database.

For detailed information on how to use the Packaging Wizard, see the *Oracle Database Lite Tools and Utilities Guide*.

Oracle Database Lite 10g ADO.NET Provider

This document discusses 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.

The topics that are discussed in this document are the following:

- [Section 6.1, "Classes"](#)
- [Section 6.2, "Running the Demo"](#)
- [Section 6.3, "Limitations"](#)

6.1 Classes

This section describes the classes in the Oracle Database Lite 10g ADO.NET provider.

6.1.1 OracleConnection

This is the primary interface to establish a connection to Oracle Database Lite. This class implements the **System.data.IDBConnection** interface. You can pass an ODBC data source name when constructing an instance of the **OracleConnection** class. For example:

```
IDBConnection conn = new OracleConnection ("POLITE");  
  
conn.Open();
```

In a general case, it is possible to pass a full connection string as described in Oracle Database Lite ODBC documentation for the `SQLDriverConnect` API. For example:

```
OracleConnection conn = new OracleConnection  
("DataDirectory=\\orace;Database=polite;DSN=*;uid=system;pwd=manager");  
  
conn.Open();
```

You can also construct an empty connection object and set **ConnectionString** property later.

With an embedded database, it's recommended to open the connection at the beginning and leave it open for the lifetime of the program. Note that closing the connection will also close all the `IDataReader` cursors using it.

6.1.2 Transaction Management

By default, Oracle Database Lite connection is in autocommit mode. To begin a transaction, use the **BeginTransaction()** method in the **OracleConnection** object.

Commit or **Rollback** on the returned **IDbTransaction** will do the requested action and return the database to autocommit mode. You can use SQL syntax to set up, remove and undo savepoints within a transaction.

For Microsoft Pocket PC-based devices, Oracle Database Lite only supports one process accessing a given database. When a process tries to connect to a database in use, the **OracleConnectionOpen** method will throw an **OracleException**. You can temporarily close a connection to allow another process to connect.

6.1.3 OracleCommand

The **OracleCommand** class implements the **System.Data.IDbCommand** interface. The recommended way to create commands is through the **CreateCommand()** method of the **OracleConnection** class. **OracleCommand** does have constructors recommended by the ADO.NET manual, for example `OracleCommand(OracleConnection conn, string cmd)`.

However this use will make it difficult to port the code to, for example, the ODBC provider on Windows 32. Creating a connection and then using interface methods to derive other objects will make changing the provider trivial at compile time (or even using reflection API at runtime).

6.1.4 OracleParameter and Prepared Statements

Parsing a new SQL statement takes a significant time. It's important to use prepared statements for any performance-critical operations. Although, **IDbCommand** has an explicit **Prepare** method, a statement will also be prepared on the first use. Just reuse the object repeatedly without calling **Dispose** or changing the **CommandText** property.

6.1.4.1 Parameters

Oracle Database Lite uses ODBC-style parameters, such as the "?" character (without the quotation marks) in the SQL string. Parameter names and data types are ignored by the driver and are only for the programmer's use.

Example:

Let us assume that you have created the following table:

```
create table t1(c1 int, c2 varchar(80), c3 data)
```

You can use the following parameters in the context of the table that you have created.

```
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();
```

The relevant class names are **OracleParameter** and **OracleParameterCollection**.

6.1.5 OracleBlob and Large Object Support

Oracle Database Lite 10g includes in its classes the **OracleBlob** class to support large objects. Currently, the Oracle Database Lite ADO.NET implementation supports only the BLOB data type. Create a new **OracleBlob** object to instantiate (or insert) a new BLOB object in the database, as follows:

```
OracleBlob blob = new OracleBlob(conn);
```

This object can be used in the same way as the objects for the other datatypes. You can use it 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();
```

The **Oracle Blob** object also can be retrieved using the data reader to query a table with a BLOB column:

```
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);
```

The **OracleBlob** class supports reading and writing to the underlying BLOB, as well as retrieving and modifying the BLOB's size. Use the **Length** property of **OracleBlob** to get or to set its size and 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:

```
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
```

You can use the **Connection** property of **OracleBlob** to retrieve the current **OracleConnection**. You can also 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.

6.1.6 OracleSync and Data Synchronization

To programmatically synchronize databases, you can use the **OracleSync** class. Instantiate an instance of the **OracleSync** class, set relevant properties, and call the **Synchronize** method to trigger data synchronization.

For example,

```
OracleSync sync = new OracleSync();
sync.UserName = "JOHN";
sync.Password = "JOHN";
sync.serverURL = "mobile_server";
sync.synchronize();
```

If you want to get Synchronization Progress information, you must set the **SyncEventHandler** attribute of the **OracleSync** class.

For example,

```
sync.SetEventHandler (new OracleSync.SyncEventHandler (MyProgressHandler), true);
```

The **MyProgress** method must have the following signature.

```
Void MyProgress(SyncStage stage, int Percentage)
```

The Oracle Database Lite ADO.NET provider offers basic support for synchronization with the Oracle server database through the launching of the **mSync** tool. To bring up the **mSync** tool's user interface and to enable the user to modify settings before doing a synchronization, call:

```
OracleEngine.synchronize(false)
```

If the settings are already correct and you want to do an automatic synchronization, call:

```
OracleEngine.synchronize(true)
```

To do a regular synchronization with a specific server, you can use the following:

```
OracleEngine.synchronize("S11U1", "manager", "myserver.mydomain.com")
```

Finally, you can call the following:

```
OracleEngine.synchronize(args)
```

Note: Construct "args" using the options that are listed in [Table 6–1](#).

[Table 6–1](#) lists the command line options and "args" (or arguments) that are recognized.

Table 6–1 Command Line Options

Option	Description
username/password@server[:port] [@proxy:port]	Automatically synchronize to the specified server.
/a	Automatically sync 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.
/cast5	Synchronize with CAST5 encryption.
/force	Force refresh.
/noapp:(application_name)	Do not synchronize specific Web-to-Go application data. Synchronize with other applications.

Table 6–1 (Cont.) Command Line Options

Option	Description
/nopub: (publication_name)	Do not synchronize specific publication data. Synchronize with other publications.
/notable: (table_name)	Do not synchronize specific table data. Synchronize with other tables.
/notable: (odbc_name) . (table_name)	
/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)	Synchronize only specific table data. Do not synchronize with other tables.
/onlytable: (odbc_name) . (table_name)	
/high_priority	Enable high priority data synchronization.

InvalidOperationException will be thrown if synchronization fails. Note that you need to close all database connections before doing a synchronization.

6.2 Running the Demo

This release comes with a sample code demo that illustrates working code using the Oracle Database Lite ADO.NET provider. To run the demo, follow these steps:

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, for example **olite.us.pocket_pc.arm.CAB** from the following directory:
`<Oracle_home>\Mobile\Sdk\wince\Pocket_PC\cabfiles`
3. Open **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.

The demo is a minimalist timecard application for a cable technician who might install, remove, or repair service and keep track of the hours worked. 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's row to remove it. You can also navigate to a different date to review past work (change date on the device to create some work items first).

Examine **MainForm.cs** in ClockIn subdirectory. Pay special attention to the following items:

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.

Now make some changes to become familiar with ADO.NET development:

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 will need to define a primary key on ClockIn table (use a sequence).

To use the Oracle Database Lite ADO.NET provider from your own project, add a reference to **Oracle.DataAccess.Lite_wce.dll**.

6.3 Limitations

In this release of the Oracle Database Lite ADO.NET provider, **GetSchemaTable** only returns partial data. For example, it claims that all the columns are primary key, doesn't report unique constraints, and returns null for **BaseTableName**, **BaseSchemaName** and **BaseColumnName**. It is recommended that you use **ALL_TABLES** and **ALL_TAB_COLUMNS** instead of this call to get Oracle Database Lite meta information.

6.3.1 Thread Safety

To build a thread-safe program, make sure that different threads use different **IDbCommand** and **IDataReader** objects. The **OracleConnection** and **IDbTransaction** methods can be called concurrently, except for opening and closing the connection.

Developing Mobile Applications for Palm OS Devices

This document discusses building Oracle Database Lite 10g applications for Palm devices. Oracle Database Lite 10g 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 10g applications using Metrowerks CodeWarrior 9. It includes the following topics:

- [Section 7.1, "Installing Oracle Database Lite Runtime on the Device"](#)
- [Section 7.2, "Uninstalling or Replacing Oracle Database Lite Runtime"](#)
- [Section 7.3, "Running Oracle Database Lite on Palm OS Emulator"](#)
- [Section 7.4, "Running Oracle Database Lite on Palm OS Simulator"](#)
- [Section 7.5, "Using Oracle Database Lite Base Libraries"](#)
- [Section 7.6, "Building a SODA Application"](#)
- [Section 7.7, "Building a SODA Forms Application"](#)
- [Section 7.8, "Building an ODBC Application"](#)
- [Section 7.9, "Packaging your Application with Oracle Database Lite Runtime"](#)
- [Section 7.10, "Customizing Oracle Database Lite Runtime"](#)
- [Section 7.11, "Palm Shared Library Manager \(PSLM\)"](#)

7.1 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 `olSetup` icon in the "Oracle Lite" program group. If you are installing on the device using HotSync, `olSetup` will be run automatically.

Note that a sync with mobile server will replace Oracle Database Lite runtime on the device with whatever is installed on the server. Choose "Ignore apps" option in synchronization settings to suppress this behavior.

The file **tutorial.html** walks you through building a small application for Palm. The file is located in the following directory:

```
<Oracle_home>\Mobile\Sdk\Palm\doc\tutorial.html
```

The top-level list of the Palm-specific documentation which contains the links from which you can access these various documents (including the tutorial and the SODA API Reference) is located in the following directory:

<Oracle_home>\Mobile\Sdk\Palm\doc\index.html

7.2 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 an Oracle Database Lite became corrupted (the application crashes or displays invalid data). Do a sync 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 an up to date version from the specified mobile server (by default, the same one used for sync. This can be used to reset the device to known version of Oracle Database Lite or upgrade from a version prior to 5.0.2.9.0, that doesn't support automatic upgrade.

7.3 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 mobile client or mobile SDK directory. Run olSetup once to install the Oracle Database Lite runtime. Now go to emulator debug options and make sure "MemMgr semaphore" and "Proscribed function calls" checkboxes are cleared. 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 will allow you to synchronize with the Mobile Server using the emulator.

7.4 Running Oracle Database Lite on Palm OS Simulator

Oracle Database Lite works properly on PalmOS 5.x devices, however the debug version of the PalmOS Simulator crashes when running **olSetup** because of a bug in the simulator. The release version of the simulator works properly.

7.5 Using Oracle Database Lite Base Libraries

Oracle Database Lite comes with 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 an explanation.
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 8K.
6. Oracle Database Lite interfaces may change between versions. To avoid compatibility problems, Oracle Database Lite shared libraries will return errors if the application is not linked with the matching version of the stub library. When upgrading Oracle Lite runtime, you need to re-link your application with the new stubs.

These steps allow access to Oracle Database Lite C library, which provides many of the ANSI C functions which are otherwise missing from the PalmOS platform. Examine `libc.h` for a list. `olstd.h` defines C++ classes such as a hash table, which are documented as a part of SODA.

`Libc.h` 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.

7.6 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.

The SODA documents are located in the following directory.

```
<Oracle_home>\Sdk\soda\index.html
```

7.7 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
```

7.8 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`.

7.9 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.

7.10 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 7-1

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 odbcc.prc	If you are only using SODA
Remove msqll.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.

7.11 Palm Shared Library Manager (PSLM)

For detailed information on the Palm Shared Library Manager (PSLM), refer [Chapter 8, "Palm Shared Library Manager \(PSLM\)"](#).

Palm Shared Library Manager (PSLM)

This document discusses the Palm Shared Library Manager (PSLM). It includes the following topics:

- [Section 8.1, "Overview"](#)
- [Section 8.2, "Trying out PSLM"](#)
- [Section 8.3, "Writing a PSLM Library"](#)
- [Section 8.4, "Building a Shared Library Project"](#)
- [Section 8.5, "Calling a PSLM Library from Your Application"](#)
- [Section 8.6, "Building an Application Using PSLM"](#)
- [Section 8.7, "Exceptions Across Modules"](#)
- [Section 8.8, "Cloaked Shared Libraries"](#)
- [Section 8.9, "Patching the CodeWarrior Runtime"](#)

8.1 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 to 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.

8.2 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.

8.3 Writing a PSLM Library

A PSLM library is a C++ class that extends `PSLibrary`. It exposes all its 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 its `startup` and `cleanup` methods rather than constructor and destructor to manage its 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 its 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 specify 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 constructor of a `PSLibContext` sets the context to that of the library passed as an argument. This is what enables a library to use its 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.

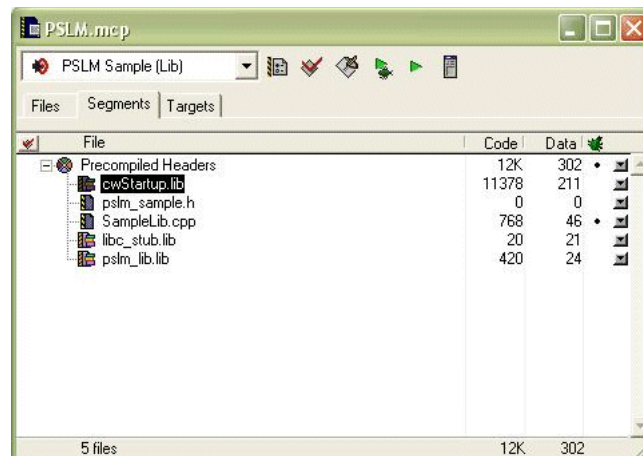
The last piece to consider is the library's `PilotMain`, which is very simple:

```
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.

8.4 Building a Shared Library Project

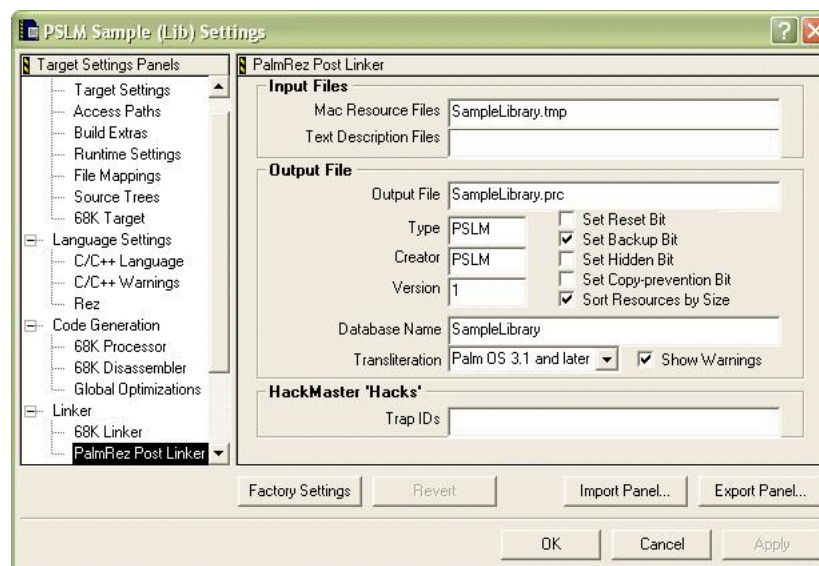
The following illustration, [Figure 8–1](#), shows the CodeWarrior project for `SampleLibrary`:

Figure 8–1 The CodeWarrior Project for SampleLibrary

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 case, I included `SampleLib.cpp` and `libc_stub.lib` (which is a static helper for ANSI C shared libraries). Another important point is that 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 8–2](#), shows the "PalmRez Post Linker" section of the SampleLibrary project:

Figure 8–2 The PalmRez Post Linker Section

A PSLM library is linked as an application, but we don't 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 planning to use when you load the library.

8.5 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 is 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;
```

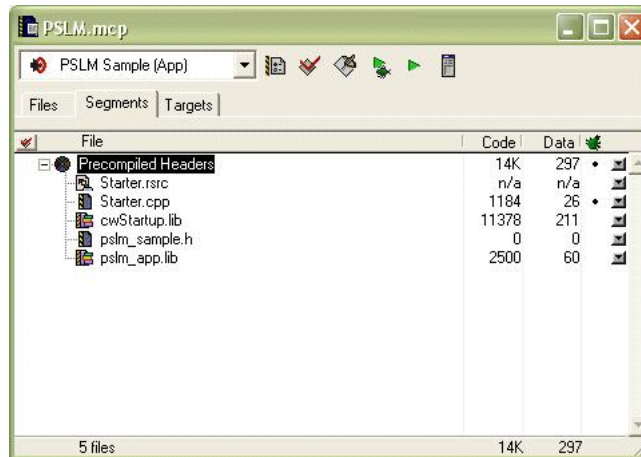
In the other extreme, you can have a function that loads a plugin, lets it do some work and then unloads it before returning. In this case, you can declare a local variable of type `PSLibObject` and even pass a dynamic argument as a library name to support user-defined plugins.

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 its 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.

8.6 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 8–3](#) shows a PSLM sample application.

Figure 8–3 PSLM Sample Application



8.7 Exceptions Across Modules

You are free to use exceptions inside the shared library, as long as they are also caught inside. Unfortunately, it's 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 its caller, you must link them with both. For this case, it's the easiest to use an inline method for re-throwing the exception.

8.8 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 semaphore 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.

8.9 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 not disable the support for old devices in build options.

Using Mobile Sync for Palm

This document discusses using Mobile Sync (mSync) for Palm. mSync for PalmOS allows a user or a developer to synchronize data with the Mobile Server. User can run the application (which can be found in Oracle Database Lite program group) manually and configure various settings. He or she can then tap the Sync button to manually initiate synchronization over the default network connection configured on the PDA. There are several other ways to invoke sync with pre-configured settings:

- HotSync will automatically synchronize 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, will launch mSync and attempt to synchronize over the network connection.
- SODA Forms applications have "Sync data with the mobile server" option in their "Form" menu.

9.1 Configuring mSync

When mSync is run, it displays a screen with the most commonly used synchronization settings. [Table 9–1](#) shows the controls for the synchronization settings that are displayed by mSync.

Table 9–1 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 will be 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 host name or hostname:port of your mobile server. When syncing over network, you might consider entering the IP address instead to work around DNS configuration problems.

Table 9–1 (Cont.) List of Controls for Synchronization Settings

Option	Use
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	This checkbox activates 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	This button will start a sync over the default network connection.
Cancel button	This button will exit mSync and re-start the SODA or SODA Forms application, if any, that initiated it.

mSync also provides the synchronization settings item under its Options menu, which can be used to adjust some less frequently used settings, as described in [Table 9–2](#).

Table 9–2 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 don't 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.

9.2 Using HotSync to Synchronize Data with the Mobile Server

This section discusses using HotSync to synchronize data with the Mobile Server.

9.2.1 Configuring HotSync for a PalmOS Device

To synchronize Oracle Database Lite databases over HotSync, first install Mobile client for PalmOS on a PC that already has Palm Desktop. Start mSync on PDA and configure all the sync options, including "Save password" checkbox. Also, find and enable "Stay on in Cradle" options in PalmOS Prefs application. Now every HotSync will automatically synchronize Oracle Database Lite data with the Mobile Server. After HotSync finished, refer to LiteLog on the device or HotSync log on the desktop to check for errors.

9.2.2 HotSync Timeout Errors

If the mobile server sends a large volume of data to the PDA, Palm Desktop may timeout during HotSync and you will see a message box on the desktop while the PDA is still processing sync data. Just dismiss the dialog or let it time out automatically. mSync will detect this condition and automatically do another HotSync, which should go through successfully. 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, do another HotSync manually.

This timeout is an issue in Palm Desktop software rather than Oracle Database Lite, in particular SyncCallRemoteModule API that is used by a conduit to invoke an application on Palm. If the application takes a lot of time to run, 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 it.

9.2.3 Configuring PalmOS Emulator for HotSync

It's possible to do 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.

9.3 Using Network Sync

mSync can also open a direct connection to the mobile server without going through the Palm Desktop. If you have a valid network connection, for example through a modem, integrated cellular phone or Bluetooth, just tap the Sync button on the mSync main screen.

9.3.1 Synchronizing Using a Cradle and Windows Desktop

It's 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.

9.3.2 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.

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 10.1, "Overview"](#)
- [Section 10.2, "Developing Offline Mobile Applications for Win32"](#)

10.1 Overview

To demonstrate the steps involved in building offline mobile applications for the Win32 platform, this tutorial presents a simplified mobile field service example.

10.2 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 `oradbsvr` 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 `oradbsvr` 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 10g 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.

10.2.1 Command Sequence

The following sections describe the command sequence.

10.2.1.1 Step 1. Create TASK Table on the Server Database

We will use the Oracle9i thin JDBC driver to connect to the Oracle database running in the `oradbsvr` 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:@oradbsvr: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
```

10.2.1.2 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:@oradbsvr: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.

10.2.1.3 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.

10.2.1.4 Step 4. Install the Oracle Database Lite 10g 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 10g 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 Command 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).

10.2.1.5 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.

10.2.1.6 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.

10.2.1.7 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
```

10.2.1.8 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.

10.2.1.9 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`.

10.2.1.10 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 10g 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. Enter notes for one of the tasks. Then execute `D:\MFS\Mobile\bin\msync.exe` to synchronize your changes with the server.

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 10g ADO.NET interface for Pocket PC. It enables you to implement offline mobile applications for the Pocket PC using Oracle Database Lite 10g. It provides you with the complete framework to build, deploy, and manage offline mobile applications. Oracle Database Lite 10g supports various application models for the Pocket PC by supporting industry standard interfaces such as ODBC, JDBC, and ADO.NET. Topics include:

- [Section 11.1, "Overview"](#)
- [Section 11.2, "Developing the Application"](#)
- [Section 11.3, "Packaging and Publishing the Application"](#)
- [Section 11.4, "Administering the Application"](#)
- [Section 11.5, "Running the Application on the Pocket PC"](#)

11.1 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.

11.1.1 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.

11.1.1.1 Application Development Computer Requirements

You must configure and install the following components on the development computer.

[Table 11–1](#) lists the configuration and installation requirements for the mobile application development computer.

Table 11–1 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 1.3.1 or higher.
Installed Oracle Database Lite 10g Components	Oracle Database 8.1.7 or higher. The Mobile Server (Oracle Database Lite 10g CD-ROM). The Mobile Development Kit (Oracle Database Lite 10g CD-ROM).
Installed Pocket PC Components	Microsoft Active Sync 3.7.1 or higher. Microsoft eMbedded Visual Toolkit 3.0

11.1.1.2 Client Device Requirements

You must connect the client device to the desktop and install the Oracle Database Lite 10g client for Pocket PC on the device. For more information on how to install the Mobile Client on the device, see [Section 11.5.1, "Installing the Oracle Database Lite Mobile Client for Pocket PC"](#).

11.2 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.

11.2.1 Creating Database Objects in the Oracle Server

During deployment, the Mobile Server automatically creates the Oracle Database Lite 10g 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.

11.2.1.1 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 11–2 lists the columns of packages that enable you to store all information about the package.

Table 11–2 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 11–3 lists the columns of routes that enable you to store all information about a route.

Table 11–3 Routes Table

Column	Description
ROUTE_NO	Route Number (Primary Key)
ROUTE_NM	Route Name
EST_TIME	Estimated Time

Table 11–4 lists columns of trucks that enable you to store all information about the availability status and destination information for a truck.

Table 11–4 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`".

11.2.2 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.

11.2.2.1 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 11.3.2, "Defining the Application Connection to the Oracle Database Server"](#).

11.2.2.2 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.

11.2.2.3 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)
```

11.2.2.4 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()
```

11.2.3 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).

11.2.3.1 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
```

11.2.3.2 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.

11.3 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.

11.3.1 Defining the Application Using the Packaging Wizard

Using the Packaging Wizard, you can select and describe the Transport application.

11.3.1.1 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.

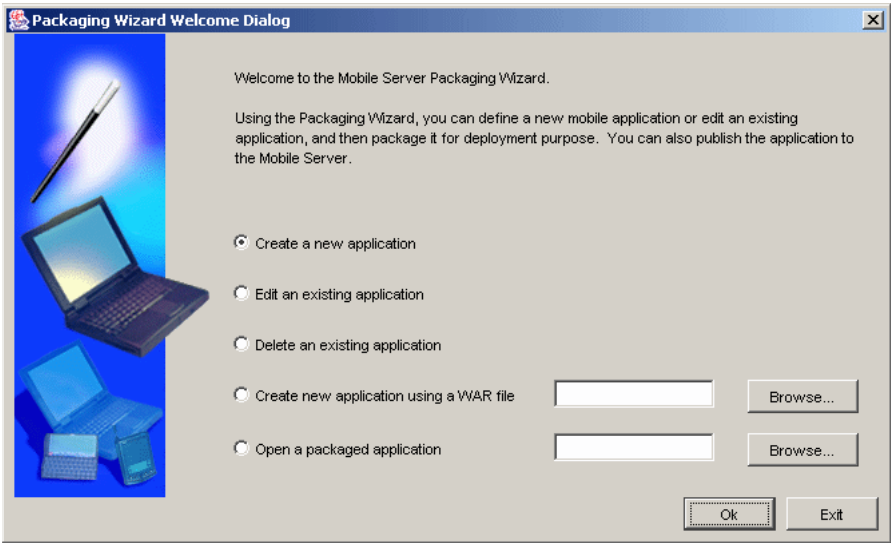
To launch the Packaging Wizard in regular mode, perform the following steps.

1. Using the Command Prompt, enter the following.

```
cd ORACLE_HOME\mobile\sdk\bin
wtgpack
```

As [Figure 11–1](#) displays, the Packaging Wizard displays the Welcome panel. Select the **Create a new application** option and click **OK**.

Figure 11–1 Welcome Dialog



- 2. The Select Platforms panel appears. Choose WinCE from the list displayed and click **Next**.
- 3. The Application panel appears. As Table 11–5 describes, enter the Pocket PC Transport application settings. Figure 11–2 displays the Applications panel.

Figure 11–2 Applications Panel

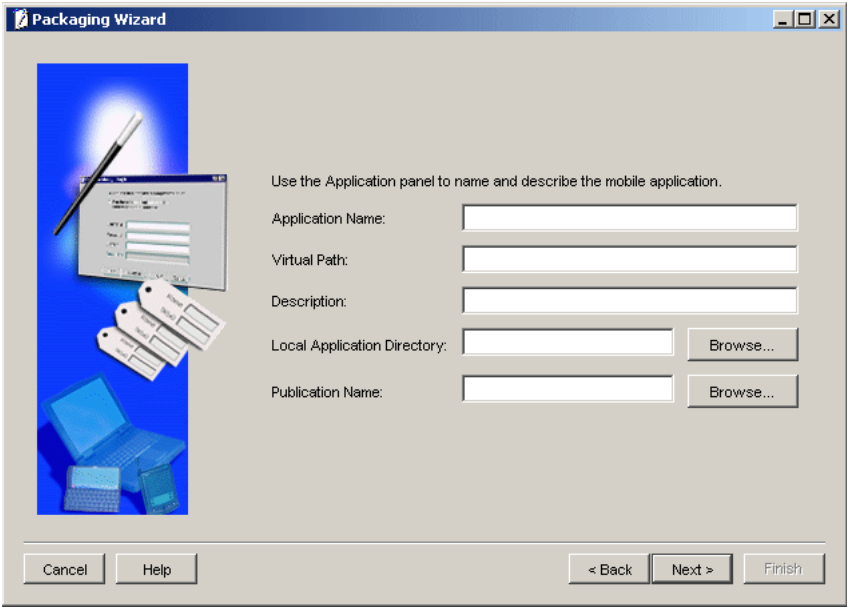


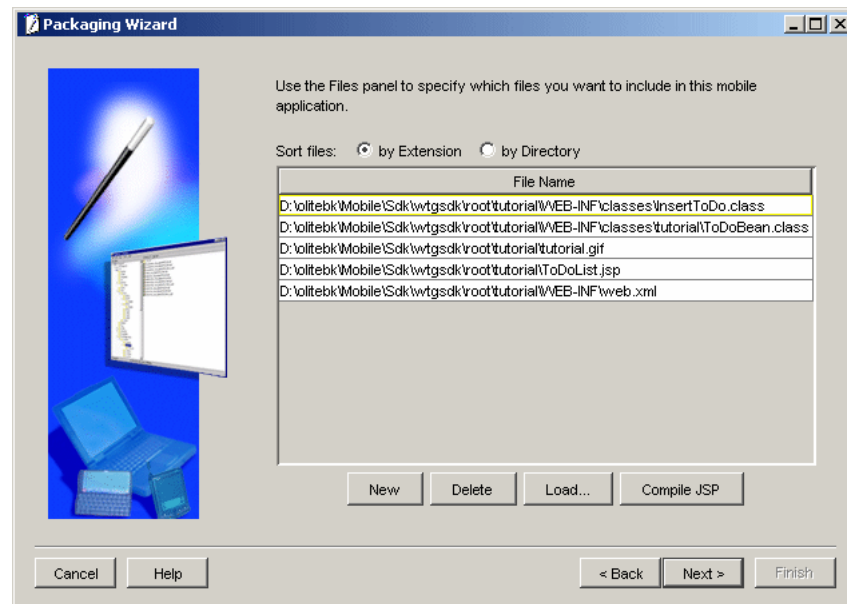
Table 11–5 The Pocket PC Transport Application Settings

Field	Value
Application Name	Transport
Virtual Path	/Transport

Table 11–5 (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 11–3](#) displays, the Files panel appears.

Figure 11–3 Files Panel

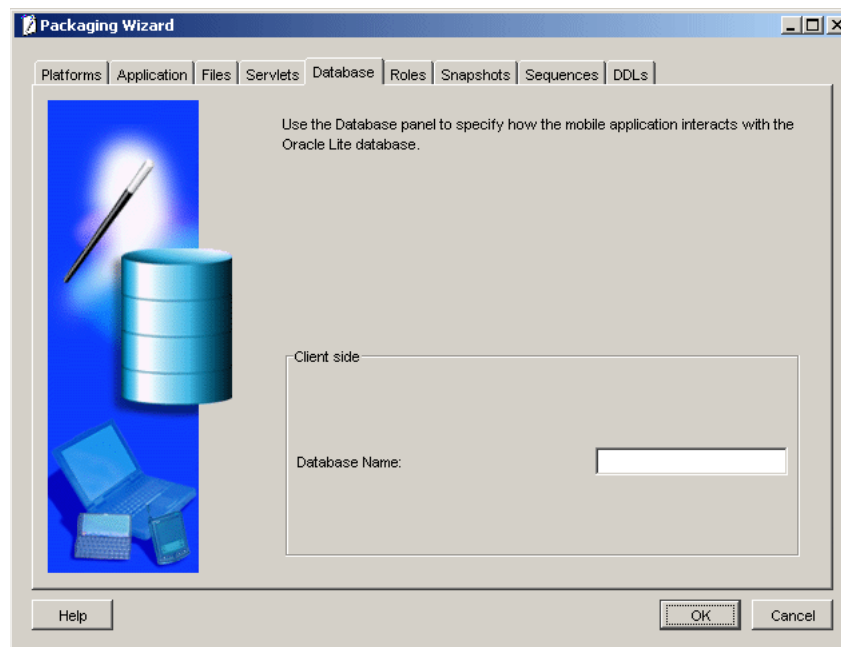
The Files panel automatically lists all files that reside in the directory, based on the 'Local Application Directory' specified in the previous Application panel. Ensure that you select the correct CAB file from the directory in which you saved the CAB file, using the Application Install Wizard.

For example, in this tutorial, you must select the `Transport_PPC.ARM.CAB` because your target device is Pocket PC with the ARM chipset.

11.3.2 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 11–4](#) displays, the Database panel appears. It enables you to define the Transport application's connection information to the Oracle Database Server.

Figure 11–4 Database Panel

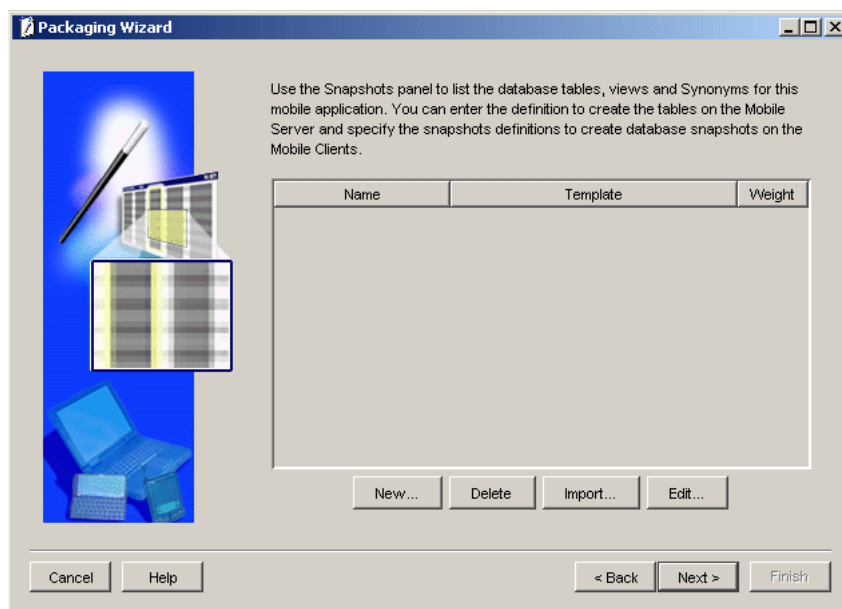
The Client Side Database Name field refers to the Data Source Name (DSN) for the Oracle Database Lite database file, which is automatically created on the device. In this field, enter the value 'transport'.

11.3.3 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 11–5](#) 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 11–6](#).

Figure 11–5 Snapshots Panel**Table 11–6 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 11–6 displays, the Edit Snapshots panel appears.

Figure 11–6 Edit Snapshots Panel

The screenshot shows a 'New Snapshots' dialog box. The 'Server' field is set to 'Oracle Lite VMN32;US'. The 'Snapshot name' field contains 'Snapshot name' and has an 'Import...' button to its right. The 'Weight' field is set to '0'. The 'Owner' field is set to 'master'. There is an unchecked checkbox for 'Generate SQL'. Below this is a large empty text area labeled 'SQL:'. At the bottom of the dialog are 'Ok' and 'Exit' buttons.

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.

11.3.4 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 11–7](#) describes, enter the specified values.

Table 11–7 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.

11.4 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*.

11.4.1 Starting the Mobile Server

To start the Mobile Server in standalone mode, enter the following command using the Command Prompt.

```
> java -jar oc4j.jar
```

11.4.2 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.

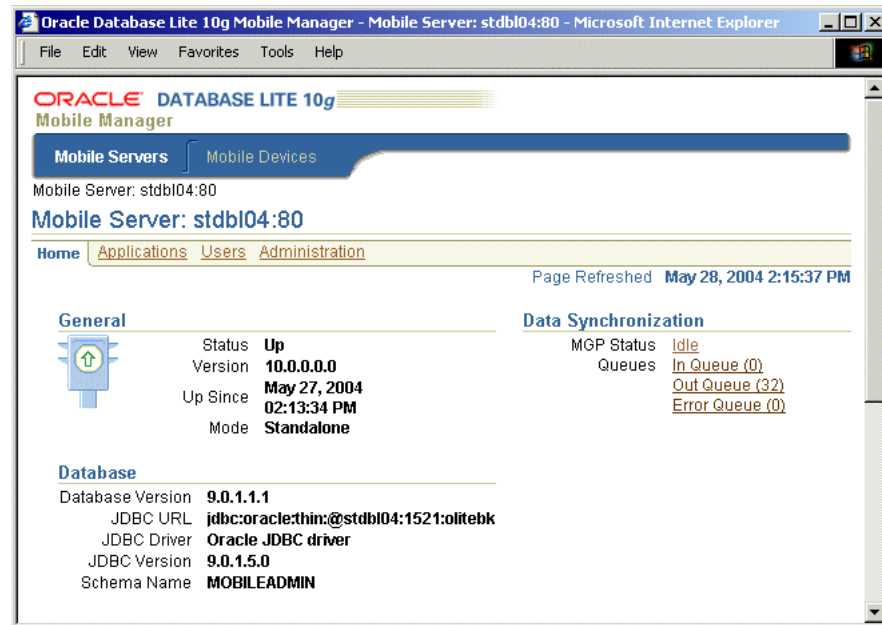
1. Open your web browser and connect to the Mobile Server by entering the following URL.

```
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.

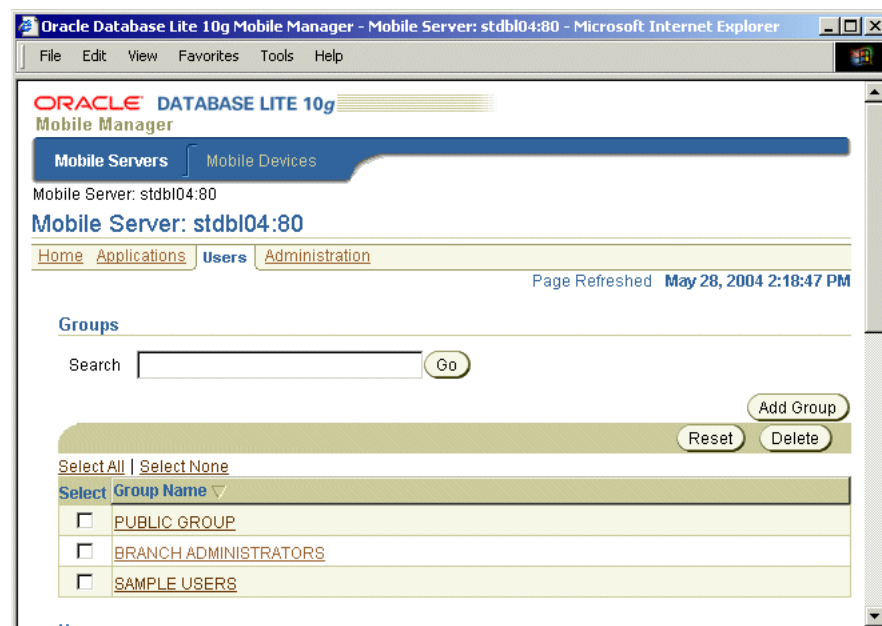
[Figure 11–7](#) displays the Mobile Server home page.

Figure 11–7 Mobile Server Home Page

11.4.3 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**. As [Figure 11–8](#) displays, the Add User page appears.

Figure 11–8 Add User Page

3. Enter data as described in [Table 11–8](#).

4. Click **Save**. The **Mobile Manager** displays a confirmation message.
5. Click **OK**.

Table 11–8 lists the values that you must enter in the **Add User** page.

Table 11–8 The Add User Page Description

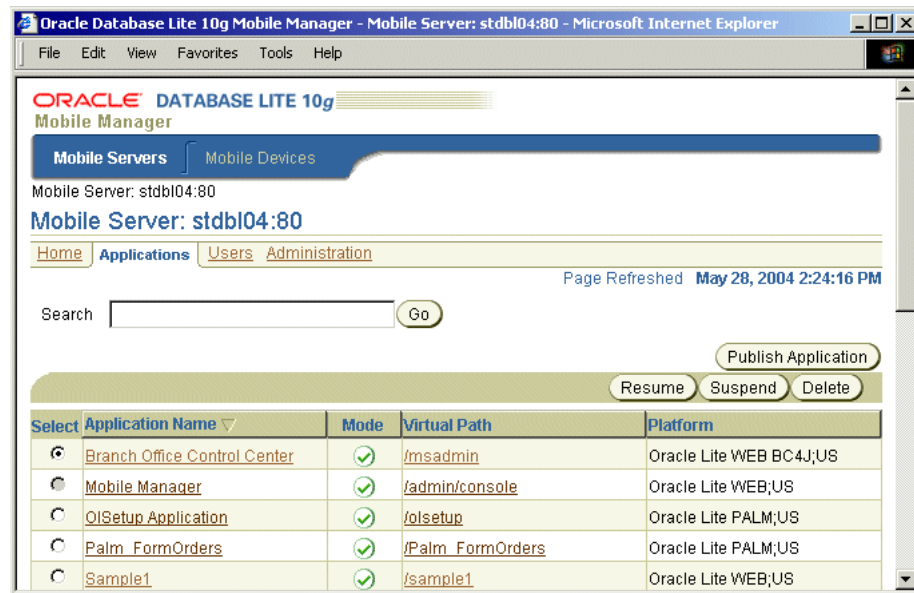
Field	Value
Display Name	bob
User Name	bob
Password	bobhope
Password Confirm	Re-enter the password for confirmation
System Privilege	Select the "User" option

11.4.4 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 11–9 displays, The **Applications** page appears. You can search the list of available applications by application name.

Figure 11–9 Applications Page



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**.

11.4.5 Granting User Access to the Application

To grant user access to the Transport application, perform the following steps.

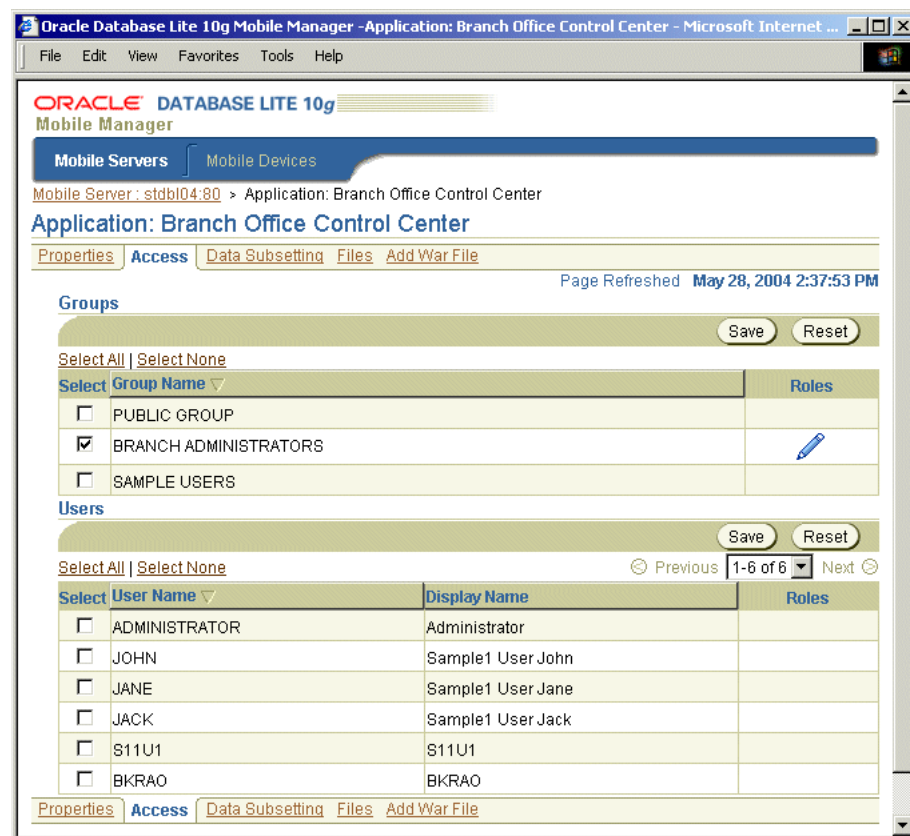
1. In the Transport application page, click the **Access** link. As Figure 11–10 displays, the Access page lists application users and application groups. To grant access to a user or a group of users to the Transport application, select the corresponding boxes.

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 11–10 displays the Access page of the Transport application.

Figure 11–10 Access Page



11.4.6 Starting the Message Generator and Processor (MGP)

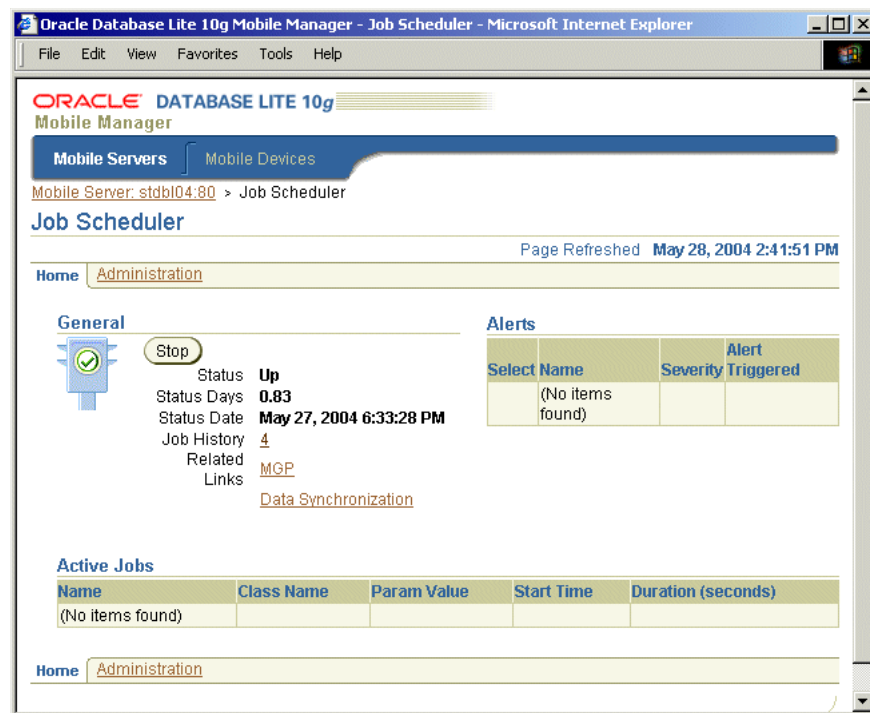
In the Oracle Database Lite 10g Asynchronous replication model, a client does not wait for the server to prepare the payload. A payload contains data that will be synchronized. The Mobile Server prepares the payload for all mobile clients asynchronously by running the MGP process in the background at 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 11–11 displays the **Job Scheduler** page.

Figure 11–11 Job Scheduler Page



11.5 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.

11.5.1 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`

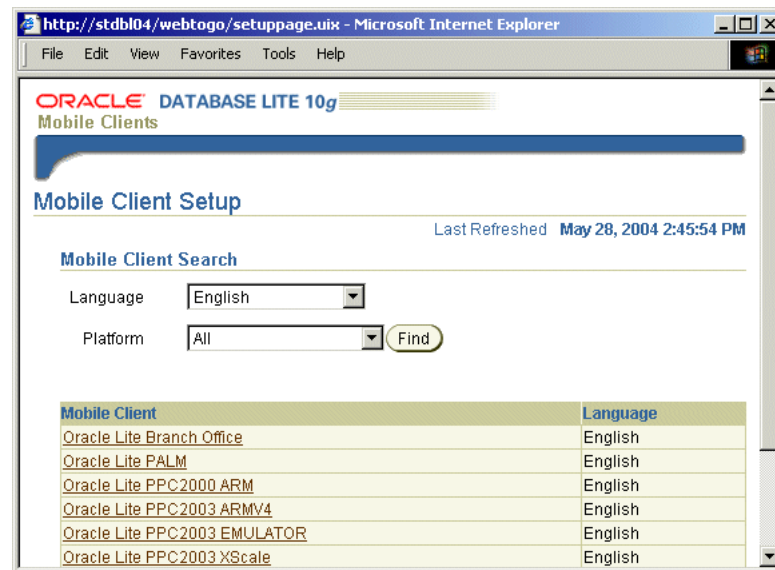
Note: You must replace the <Mobile_Server> variable with the host name or IP address of your Mobile Server.

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 11–12 displays the Mobile Client Setup page.

Figure 11–12 Mobile Client Setup Page



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.

11.5.2 Installing and Synchronizing the Transport Application and Data

To install the Transport application and data, perform the following steps.

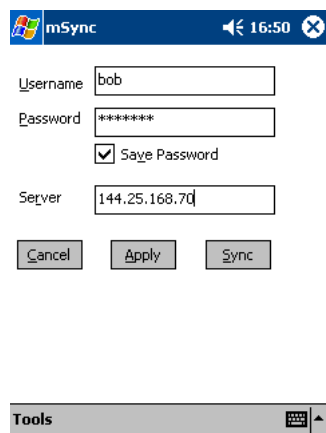
1. On the device, locate and tap the **mSync** application icon in the programs group.
2. The **mSync** dialog appears. To download the Transport application and snapshots for user BOB, enter data as described in [Table 11–9](#).

Table 11–9 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 11–13](#) displays the **mSync** dialog on the Pocket PC.

Figure 11–13 Running mSync on Pocket PC



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.

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 12.1, "Overview"](#)
- [Section 12.2, "Developing the Application"](#)
- [Section 12.3, "Packaging the Application"](#)
- [Section 12.4, "Publishing the Application"](#)
- [Section 12.5, "Administering the Application"](#)
- [Section 12.6, "Running the Application on the Mobile Client for Web-to-Go"](#)

12.1 Overview

Using a simple "To Do List" application, this tutorial guides you through the different phases of implementing a web application for mobile devices, with a detailed description of the creation, deployment, and administration phases. The To Do List application allows the user to maintain a list of To Do items. It maintains a status for each item indicating its completion and stores all items in the Oracle database. The To Do List application can be accessed by multiple users and displays 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 4, "Developing Mobile Web Applications"](#). For more information on Oracle Database Lite concepts, refer the *Oracle Database Lite Concepts Guide*.

12.1.1 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.

12.1.1.1 Development Computer Requirements

As [Table 12-1](#) describes, the development computer must contain the following components.

Table 12–1 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 1.3.1 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)

12.1.1.2 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.

12.2 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 12–2](#) describes, the To Do List application contains the following components.

Table 12–2 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 12.2.1, "Step 1: Creating Database Objects in Oracle Database Lite"](#)
- [Section 12.2.2, "Step 2: Compiling the Application"](#)
- [Section 12.2.3, "Step 3: Defining the Application and Registering the Servlet"](#)
- [Section 12.2.4, "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.

12.2.1 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.

12.2.1.1 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 12-3](#) describes, the To Do Items table contains the following columns.

Table 12-3 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.

12.2.1.2 Required Action

Create the database objects in Oracle Database Lite using MSQL. MSQL 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/xyz@jdbc:polite:webtogo @tutorial.sql`

Note: MSQL requires a user name and password. Enter `system` as the User Name and substitute `xyz` with any alphanumeric string.

There is a mandatory space between `webtogo` and `@tutorial.sql`.

12.2.2 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.

12.2.2.1 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
```

12.2.3 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.

12.2.3.1 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 the *Oracle Database Lite Tools and Utilities Guide*.

12.2.3.2 Required Action

Define the To Do List application and register its servlet by performing the following steps.

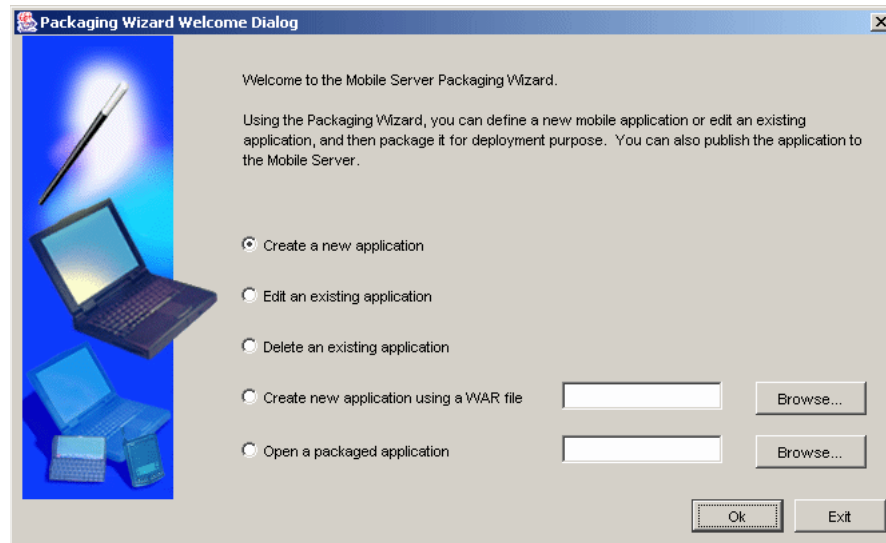
1. Start the Packaging Wizard in debug mode. Using the Command Prompt, enter the following commands.

```
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 12-1](#).

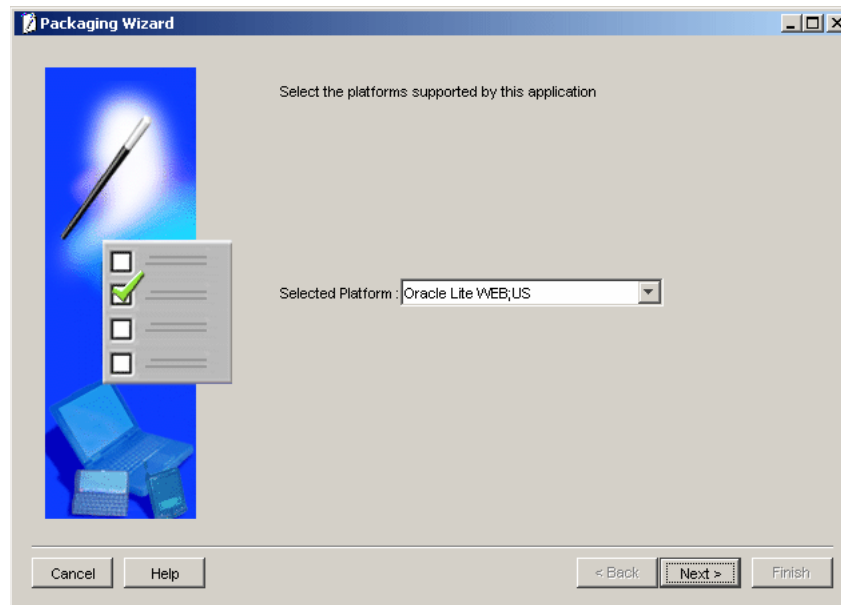
Note: Deleting an existing application merely deletes the application from the XML file and does not remove the application from the Mobile Server.

Figure 12–1 Make a Selection Dialog



2. Select the **Create a new application** option and click **OK**.
3. The **Select a Platform** panel appears. As [Figure 12–2](#) 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 12–2 Selecting a Platform



4. As [Figure 12–3](#) displays, the **Application** panel appears. Use the **Application** panel to modify To Do List application settings. As [Table 12–4](#) describes, enter the specified values in the corresponding fields.

Figure 12-3 Application Panel

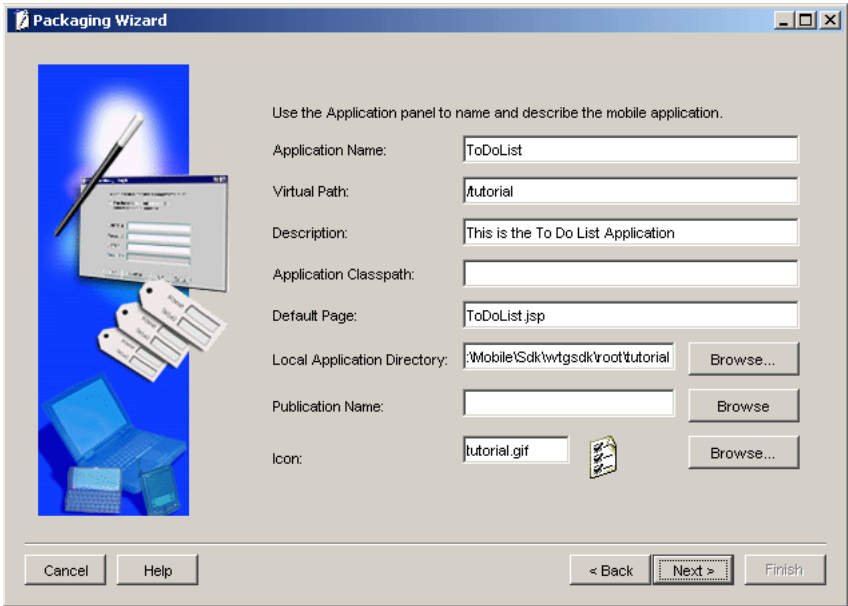


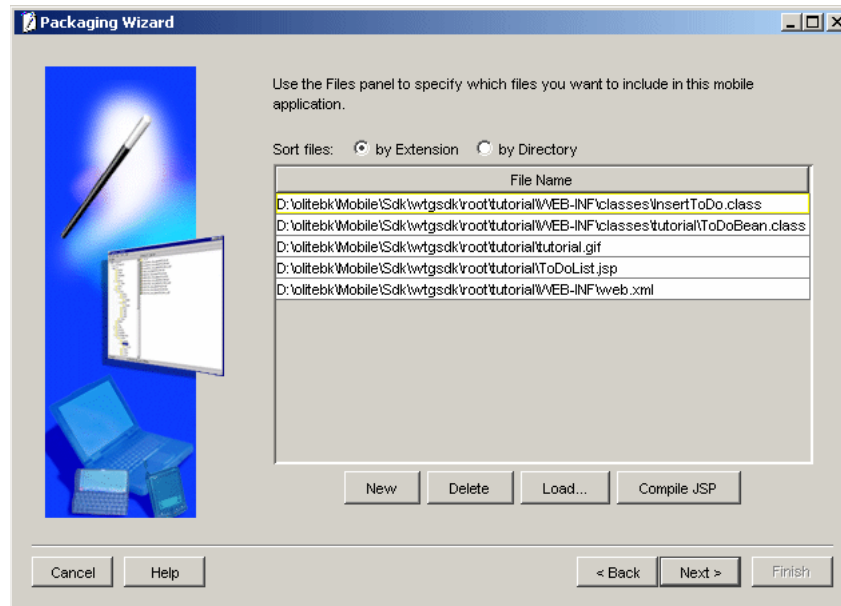
Table 12-4 The To Do List Application Values

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 12-4](#) 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.

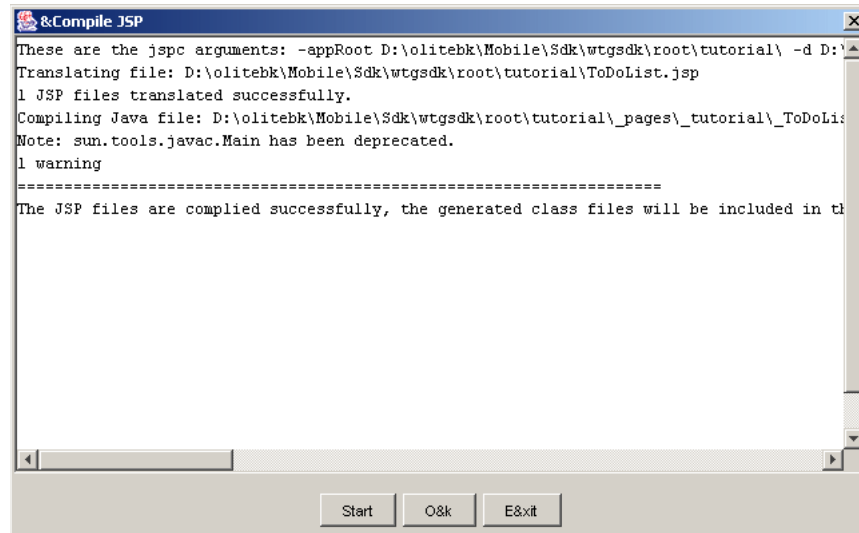
The Files panel identifies files that the Packaging Wizard uploads from the local application directory to the application repository on the Mobile Server.

Figure 12–4 Uploading Application Files

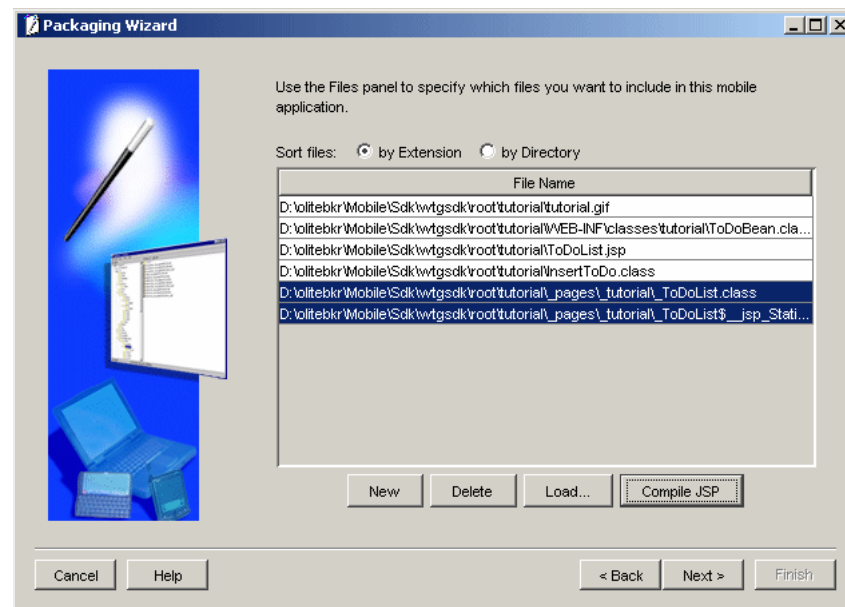


6. Click **Compile JSP**. The Packaging Wizard compiles all your JSP files to Java Servlet classes. As [Figure 12–5](#) displays, the following confirmation page appears.

Figure 12–5 JSP Compilation Completion Message



7. As [Figure 12–6](#) displays, the generated files are automatically added to the list of application files.

Figure 12–6 Including Generated Files to Application Files

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.

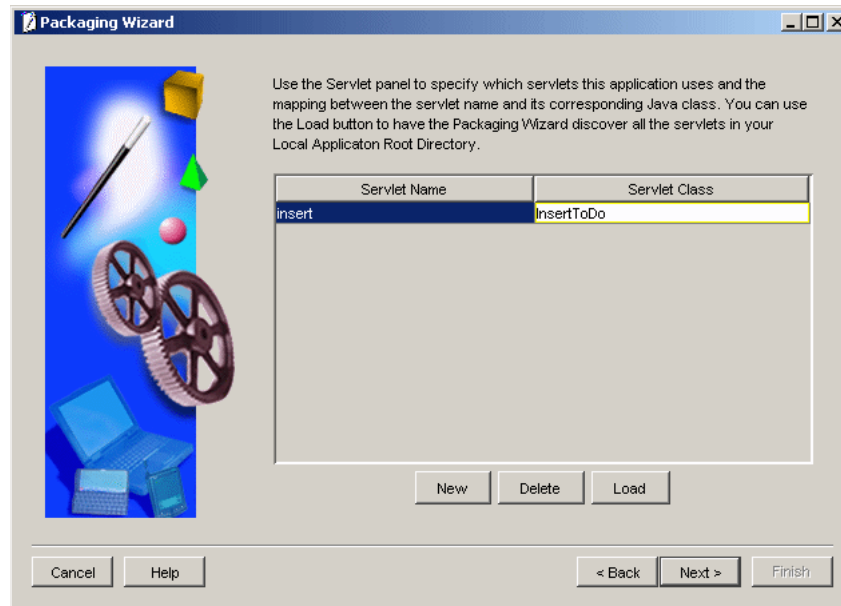
As [Figure 12–7](#) displays, you can view the To Do List application's servlet in the Servlets panel. Since the To Do List application contains only one servlet, the Servlets panel displays a single line.

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 12–7 Registering Servlets



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**.

12.2.4 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.

12.2.4.1 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*.

12.2.4.2 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 12–8 displays, the browser displays the list of applications currently known to the Web-to-Go system.

Figure 12–8 Available Applications Page

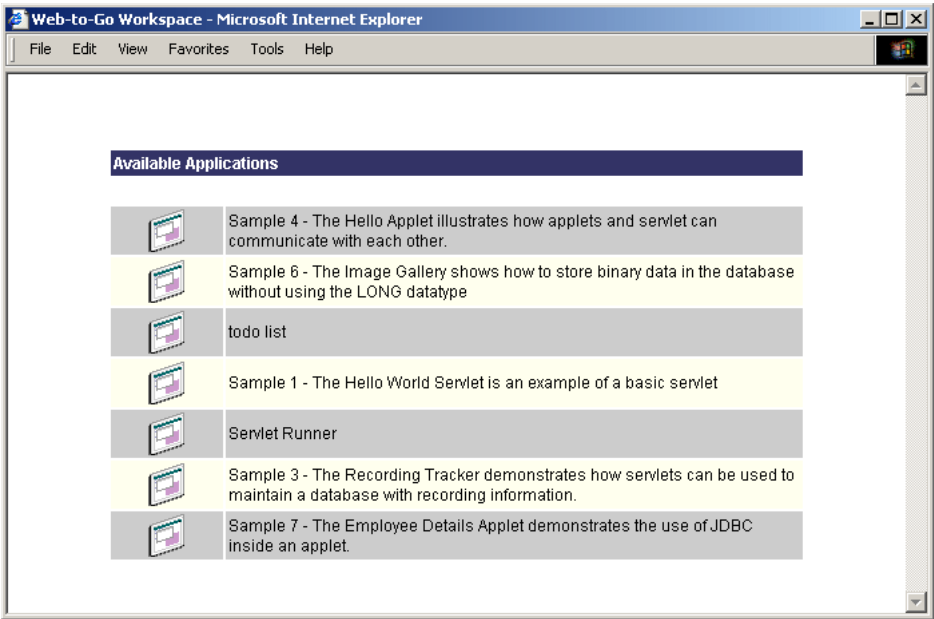


Table 12–5 describes the Available Applications page.

Table 12–5 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. <code>ORACLE_HOME\mobile\sdk\wtgSDK\root\sample4</code>
Sample 6	The Image Gallery shows how to store binary data in the database without using the <code>LONG</code> datatype. The application is located in the following directory. <code>ORACLE_HOME\mobile\sdk\wtgSDK\root\sample6</code>
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. <code>ORACLE_HOME\mobile\sdk\wtgSDK\root\sample1</code>
ServletRunner	The default application containing all published servlets that are not assigned to an application.

Table 12–5 (Cont.) List of Available Applications Description

Application	Description
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. <code>ORACLE_HOME\mobile\sdk\wtgSDK\root\sample3</code>
Sample 7	The Employee Data Applet demonstrates the use of JDBC inside an applet. The application is located in the following directory. <code>ORACLE_HOME\mobile\sdk\wtgSDK\root\sample7</code>

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.

12.3 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](#)

12.3.1 Step 1: Defining the Application

In this step, you select and describe the To Do List application using the Packaging Wizard.

12.3.1.1 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.

12.3.1.2 Required Action

Select and describe the To Do List application by launching the Packaging Wizard in regular mode.

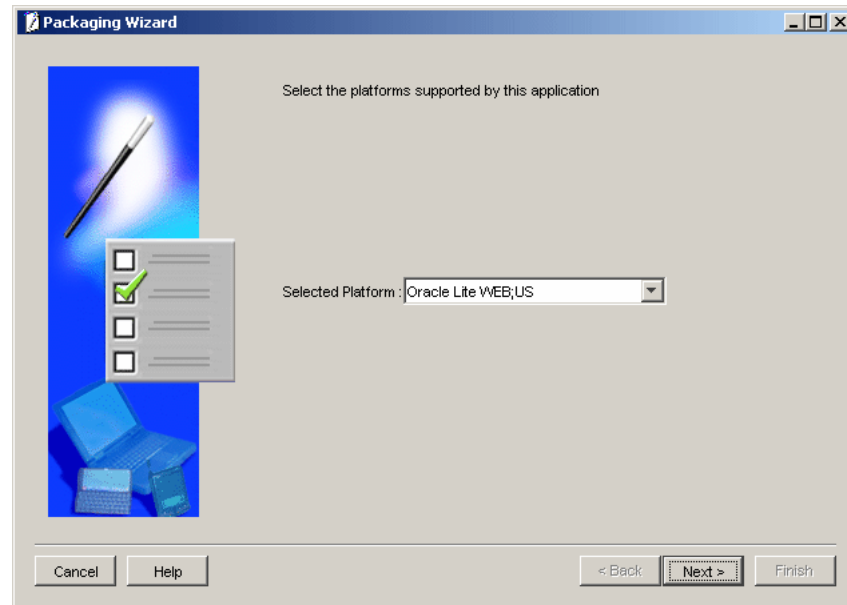
1. Using the Command Prompt, enter the following.

- a. `cd ORACLE_HOME\mobile\sdk\bin`
- b. `wtgpack`

The Packaging Wizard appears.

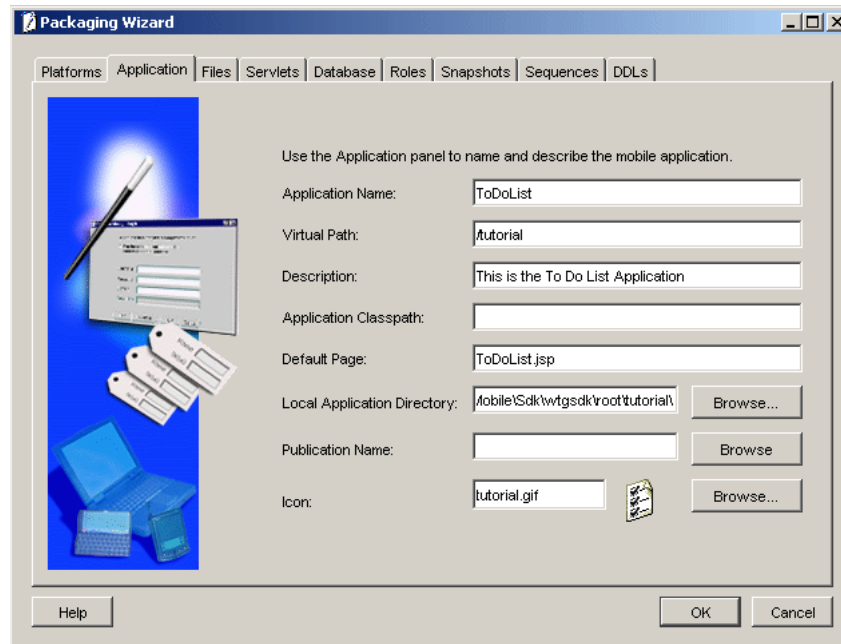
2. Choose **Edit an existing application** and select the To Do List application from the list displayed.
3. Click **OK**. The Platforms panel appears. As [Figure 12–9](#) displays, the Platforms panel contains the same information that you entered in [Section 12.2.3, "Step 3: Defining the Application and Registering the Servlet"](#).

Figure 12–9 *Selecting a Platform*



4. Click the **Application** tab. As [Figure 12–10](#) displays, the Application tab contains the same information that you entered in [Section 12.2.3, "Step 3: Defining the Application and Registering the Servlet"](#).

Figure 12–10 Application Description Panel



5. Describe the To Do List application by performing the following steps.
 - a. As [Table 12–6](#) describes, verify that the specified values in the following fields are correct.

Table 12–6 Application Panel Description

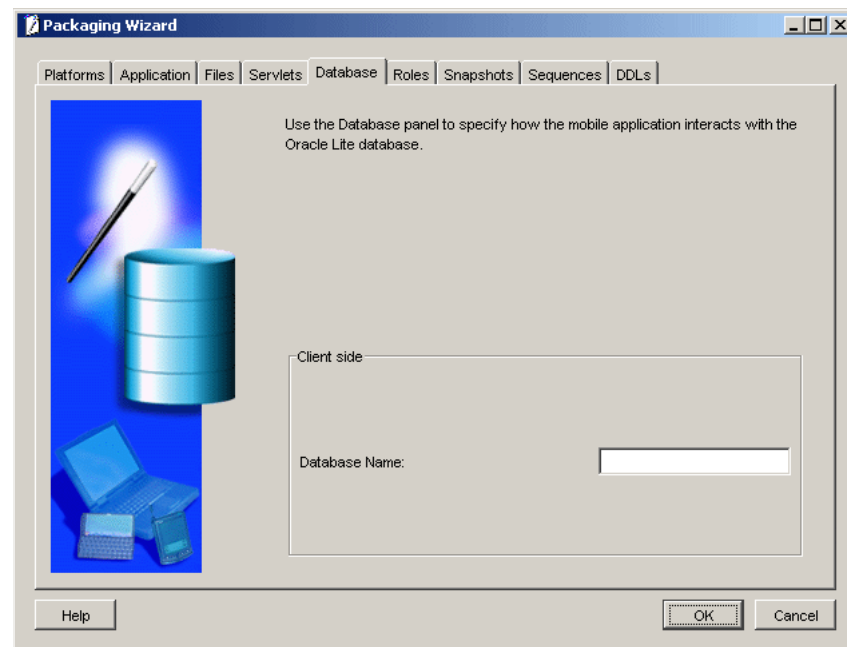
Field	Value
Application Name	ToDoList
Virtual Path	/tutorial
Description	This is the To Do List Application
Application Classpath	
Default Page	ToDoList.jsp
Local Application Directory	ORACLE_HOME\mobile\sdk\wtgSDK\root\tutorial
Publication Name	
Icon	tutorial.gif

- 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.

12.3.2 Step 2: Specifying Database Details

In this step, you will specify the Client Side Database Name.

[Figure 12–11](#) displays the Database tab.

Figure 12–11 Database Tab

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.

12.3.2.1 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 4, "Developing Mobile Web Applications"](#), [Section 4.2.2, "Application Roles"](#).

12.3.3 Step 3: Defining the Snapshot

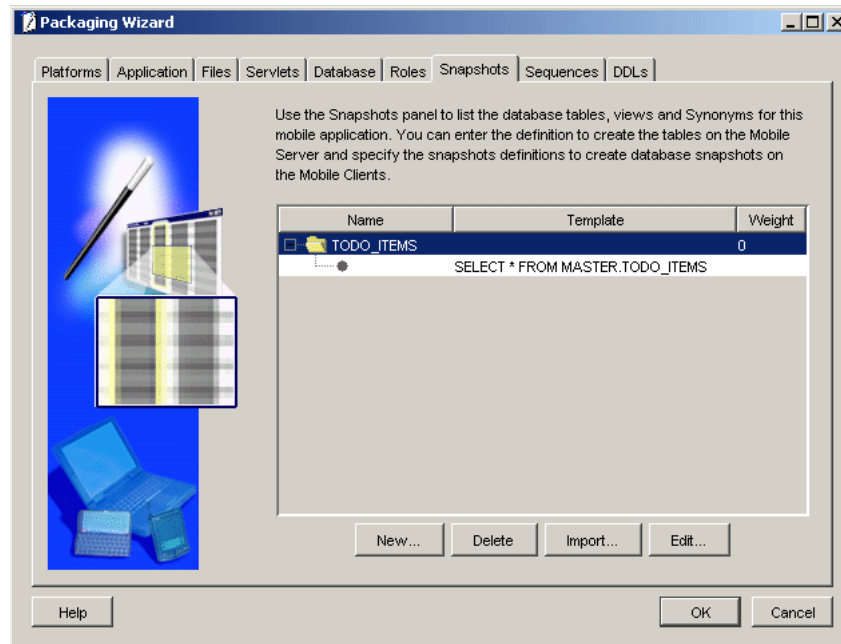
In this step, you will deploy the To Do List application's database schema objects using the Packaging Wizard.

12.3.3.1 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 12–12](#) displays the Snapshots tab.

Figure 12–12 Snapshots Tab



12.3.3.2 Required Action

In the Snapshots tab, import the table definition from the development database by performing the following steps.

1. Click **Import**. As [Figure 12–13](#) displays, the Connect to Database dialog appears. As [Table 12–7](#) describes, enter the following information in the corresponding fields.

Figure 12–13 Connect to Database Dialog

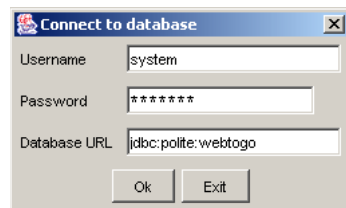


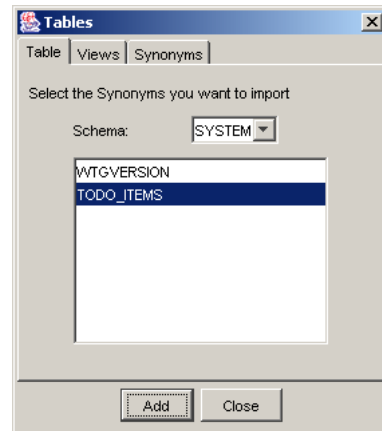
Table 12–7 Connect to Database Dialog Description

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 12-14](#) displays, the Tables dialog appears and displays a list of available tables.

Figure 12-14 *Tables Dialog*



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 12-15](#) displays, the Edit Snapshots panel appears.

Figure 12–15 Edit Snapshots Panel - Server Tab

Edit Snapshots

Server: Oracle Lite WEB;US

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))
```

Ok Exit

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 12–16 displays the SQL statement in the Template field.

Figure 12–16 Edit Snapshots Panel

Edit Snapshots

Server: Oracle Lite WEB;US

☒ Create on client ☒ Updatable?

Base Object Type: ☒ Table ☐ View

Conflict resolution: ☒ Server Wins ☐ Client Wins

Refresh type: ☒ Fast Refresh ☐ Complete Refresh

Parent Hint: Parent Table Name:

Virtual Primary Hint: Base Object Name: Base Object Column Name:

Template:

Primarykey Hint:

Table Name	Table Column Name	View Column Name

Indices

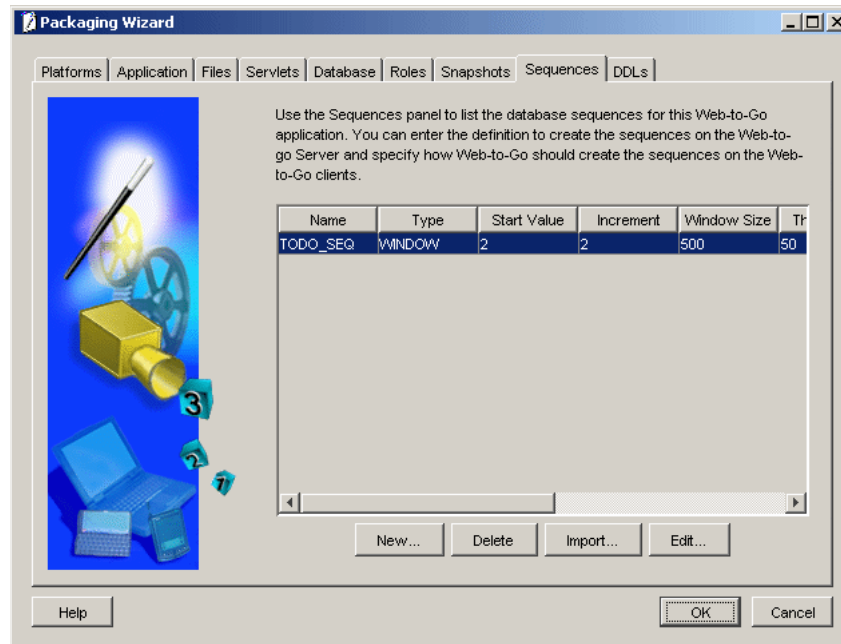
Name	Type	Columns

12.3.4 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.

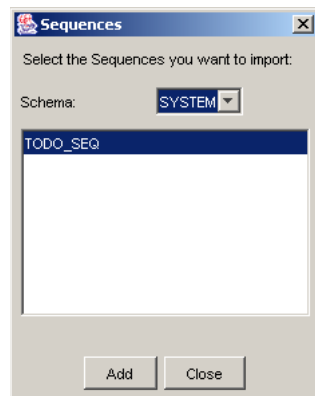
1. Click the Sequences tab. The Sequences panel appears as displayed in [Figure 12–17](#). Using the Sequences tab, you can list database sequences for Web-to-Go applications. To specify how Web-to-Go creates sequences on the Mobile Client for Web-to-Go, you can include sequence definitions. These sequences must exist in the database prior to performing this step.

Figure 12-17 Sequences Tab

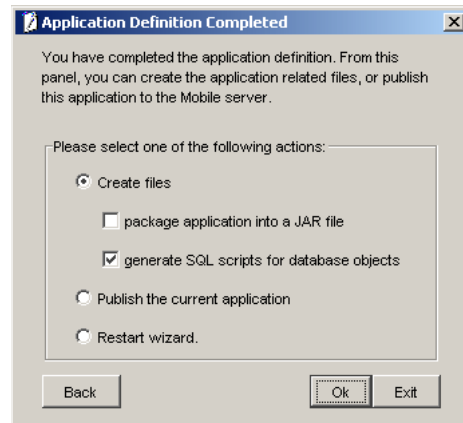


2. Click **Import**. As [Figure 12-18](#) displays, the Sequences dialog appears displaying a list of available sequences.

Figure 12-18 Sequences Dialog



3. Select the **TODO_SEQ** sequence. Click **Add** and click **Close**.
4. Click **OK**. The Application Definition Completed panel appears, as displayed in [Figure 12-19](#).

Figure 12–19 Application Definition Completed Dialog

Note: This tutorial application does not use DDLs and therefore skips the DDLs tab.

12.3.5 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.

12.3.5.1 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 12–8 describes the SQL scripts.

Table 12–8 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.
ddl.sql	This file is empty because no DLLs are defined.

12.3.6 Step 6: Package the Application

Using the Application Definition Completed panel, you can package the To Do List application into a jar file.

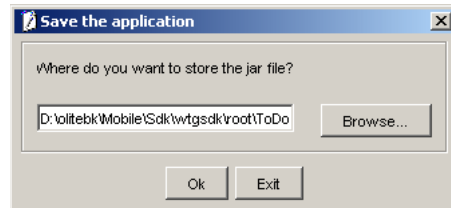
12.3.6.1 Required Action

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 12–20](#) displays. The default location is given below.

`ORACLE_HOME\Mobile\Sdk\wtgSDK\root\ToDoList.jar`

Figure 12–20 Save the Application Dialog



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.

12.4 Publishing the Application

After packaging your application, you are ready to publish it. The following sections describe the steps for publishing the application.

12.4.1 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@webtogo.world  
  
create user master identified by master;  
  
grant connect, resource to master;
```

12.4.2 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.

12.4.2.1 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@webtogo.world @ToDoList.sql
```

This script performs the following actions on the Oracle database.

- Creates the `TODO_ITEMS` table.
- Creates the `TODO_SEQUENCE` sequence.

12.4.3 Step 3: Start the Mobile Server

In this step, you start the Mobile Server.

12.4.3.1 Required Action

To start the Mobile Server, perform the following steps.

1. Using the Command Prompt, go to the following directory.

```
ORACLE_HOME\mobile_oc4j\j2ee\home
```

Note: Mobile Servers installed on *iAS* can access the *ORACLE_HOME\j2ee\home* directory.

2. To start the Mobile Server for the first time and subsequent occasions, enter the following command.

```
java -jar oc4j.jar
```

12.4.4 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**.

12.4.4.1 Required Action

To start the **Mobile Manager**, perform the following steps.

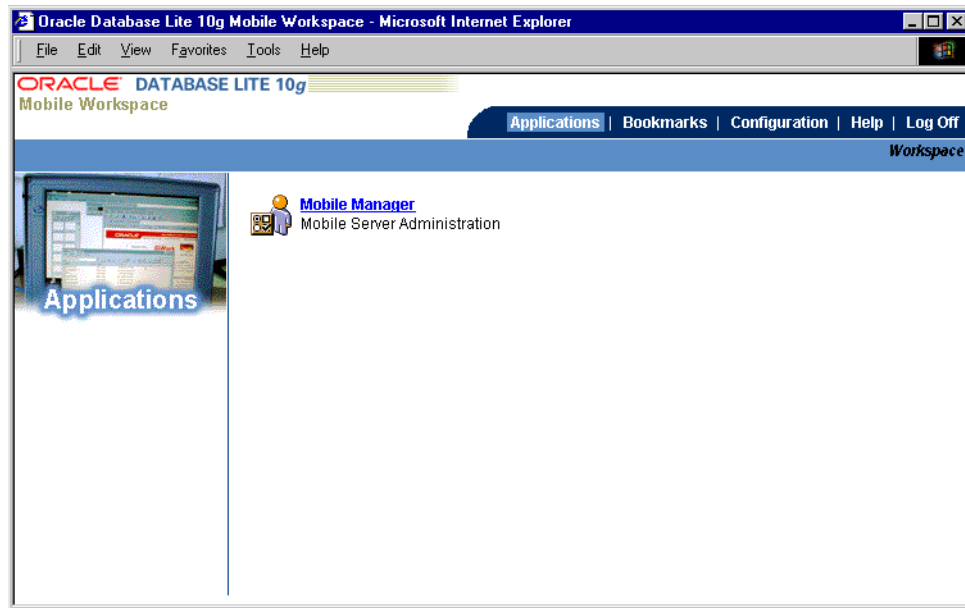
1. Start your web browser and connect to the Mobile Server by enter the following URL.

```
http://<mobile_server>/webtogo
```

Note: Replace the *<mobile_server>* variable with the host name of your Mobile Server.

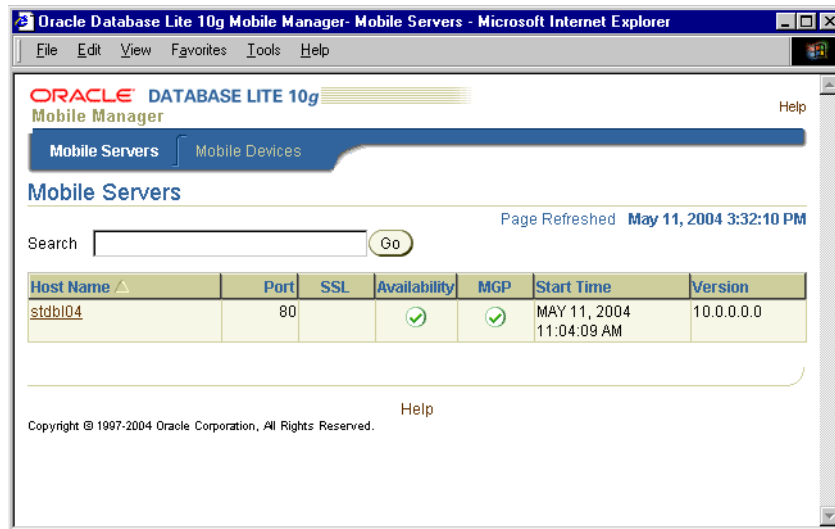
2. Log on as the Mobile Server Administrator using *administrator* as the User Name and *admin* as the Password. As [Figure 12-21](#) displays, the **Mobile Manager** link appears in the workspace.

Figure 12–21 Launching the Mobile Manager



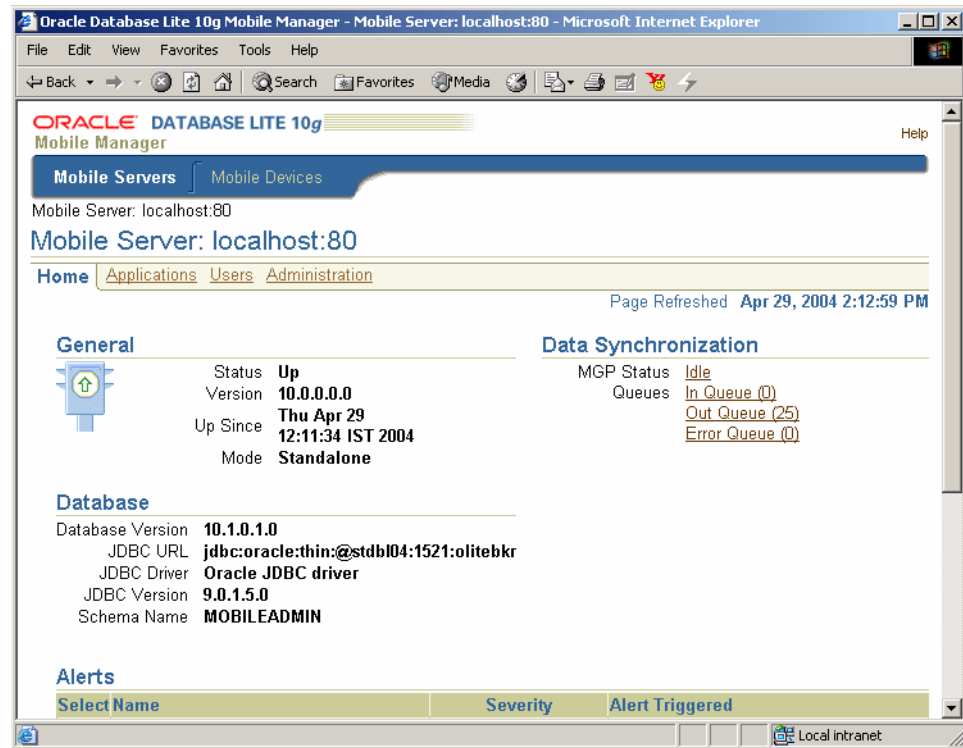
3. To launch the **Mobile Manager**, click the **Mobile Manager** link in the workspace. As [Figure 12–22](#) displays, the Mobile Server Farms page appears.

Figure 12–22 Mobile Server Farms Page



4. Click your Mobile Server link. As [Figure 12–23](#) displays, the corresponding Mobile Server home page appears.

Figure 12-23 Mobile Server Home Page



12.4.5 Step 5: Upload the Application

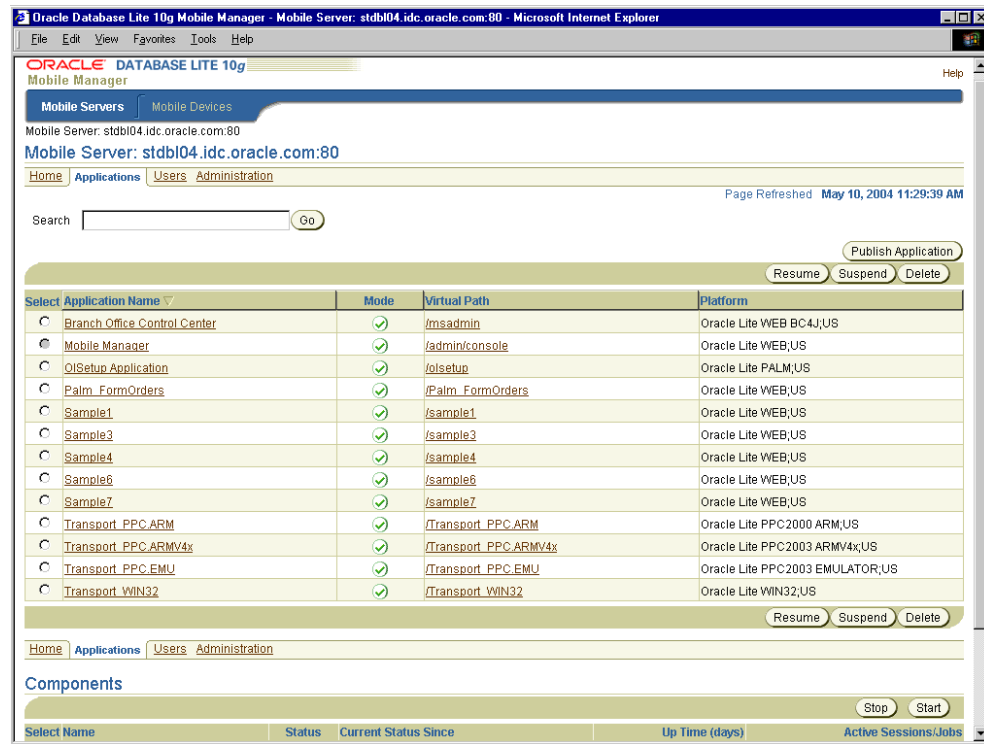
In this step, you upload the jar file containing the To Do List application.

12.4.5.1 Required Action

To upload an application to the Mobile Server, perform the following steps.

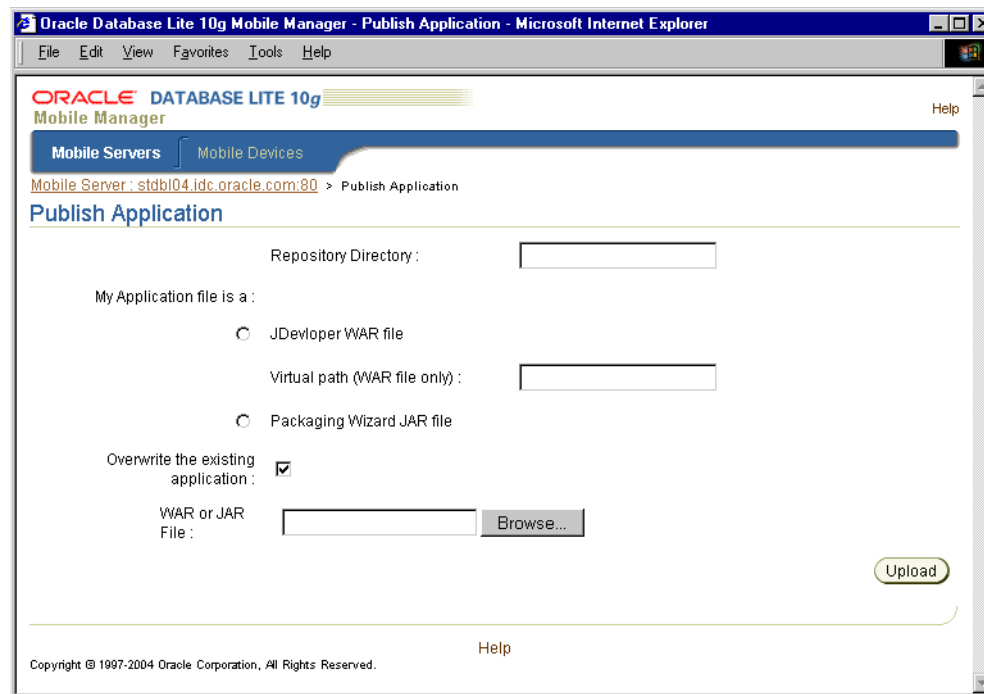
1. Click the **Applications** link. As Figure 12-24 displays, the Applications page appears.

Figure 12–24 Applications Page



2. Click **Publish Application**. As Figure 12–25 displays, the Publish Application page appears.

Figure 12–25 Publish Application Page



3. Enter `/tutorial` as the value for the repository directory.
4. Select the Packaging Wizard JAR File option.
5. Using the **Browse** button, locate the jar file which you created in [Section 12.3.6, "Step 6: Package the Application"](#). The default location of the jar file is given below.

```
ORACLE_HOME\Mobile\sdk\wtgsdk\root\ToDoList.jar
```

6. To upload the application, click **Upload**.

At this stage, your application is published.

Note: You will set the application properties in the following [Section 12.5.3, "Step 3: Setting Application Properties"](#).

12.5 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 12.5.1, "Step 1: Starting the Mobile Manager"](#)
- [Section 12.5.2, "Step 2: Using the Mobile Manager to Create a New User"](#)
- [Section 12.5.3, "Step 3: Setting Application Properties"](#)
- [Section 12.5.4, "Step 4: Granting User Access to the Application"](#)
- [Section 12.5.5, "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*.

12.5.1 Step 1: Starting the Mobile Manager

In this step, you will start the **Mobile Manager**. This web based application enables you to easily administer Mobile Server applications.

12.5.1.1 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.

12.5.2 Step 2: Using the Mobile Manager to Create a New User

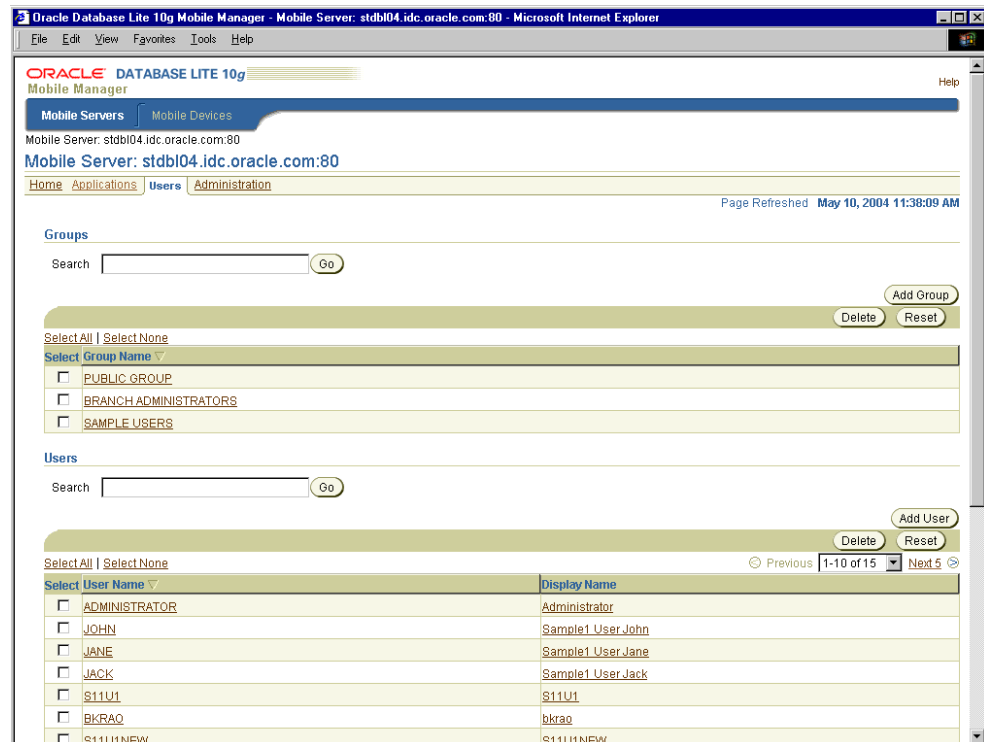
In this step, you will create a new user.

12.5.2.1 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 12-26](#) displays, the Users page appears.

Figure 12-26 Users Page



2. Click **Add User**. As [Figure 12-27](#) displays, the Add User page appears.

Figure 12–27 Add User Page

3. As described in [Table 12–9](#), enter the following information in the Add User page and click **Save**.

Table 12–9 Add User Page Description

Field	Value
Display Name	tutorial
User Name	tutorial
Password	tutorial
Password Confirm	tutorial
Privilege	User

12.5.3 Step 3: Setting Application Properties

In this step, you will set the To Do List application's properties.

12.5.3.1 Required Action

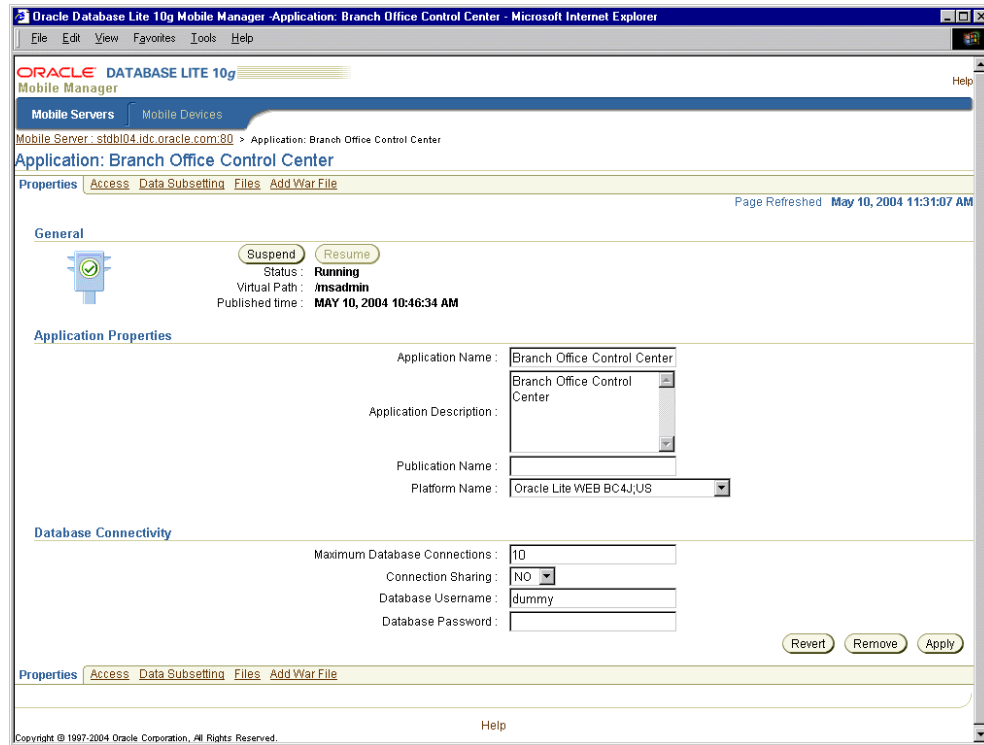
To set the To Do List application's properties, perform the following steps.

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 12–28](#) displays, the Application Properties page lists application properties and database connectivity details.

Figure 12–28 Application Properties Page



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.

12.5.4 Step 4: Granting User Access to the Application

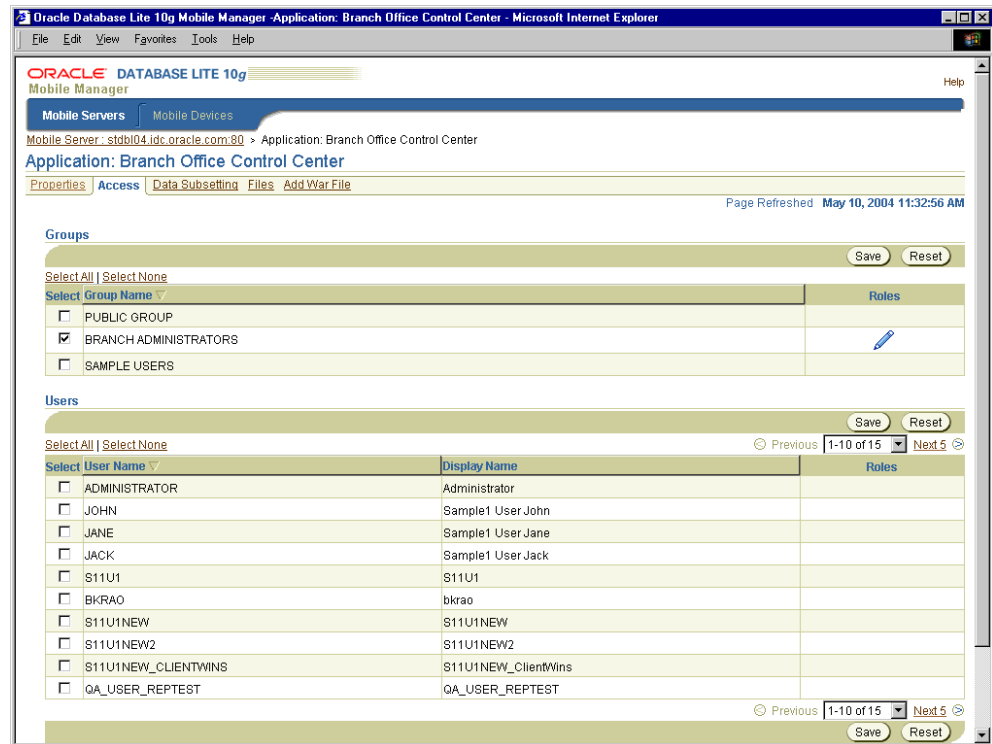
In this step, you grant the user `TUTORIAL` access to the To Do List application.

12.5.4.1 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 12–29](#) 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 12-29 Access Page



- Under the Users table, locate the user TUTORIAL and select the check box displayed against the user, TUTORIAL.
- Click **Save**. The **Mobile Manager** displays a confirmation message. The user TUTORIAL has now been granted access to the To Do List application.

12.5.5 Step 5: Defining Snapshot Template Values for the User

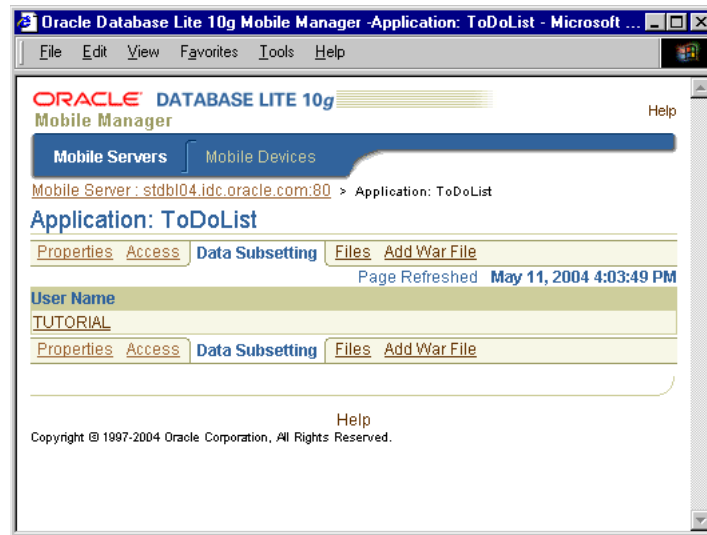
In this step, you will define the snapshot template variable for the user, TUTORIAL. Each Mobile Client for Web-to-Go downloads the same application data when it synchronizes. In some cases, you may want to specify the data your application downloads for each user. You can accomplish this by modifying the user's snapshot template variable.

12.5.5.1 Required Action

To modify a user's Data Subsetting parameters, perform the following steps.

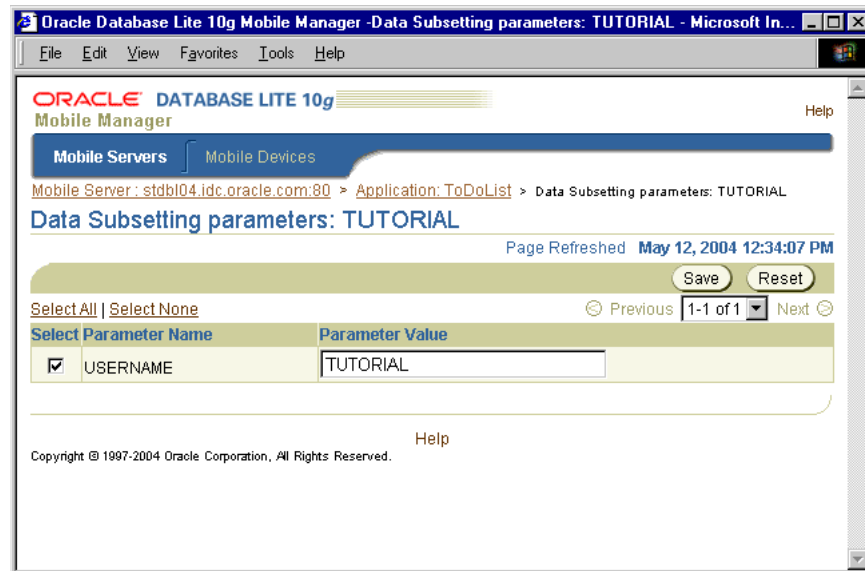
- Navigate to the Applications page and click the **ToDoList** application link. The Application Properties page appears. Click the **Data Subsetting** link. As [Figure 12-30](#) displays, the Data Subsetting page appears.

Figure 12–30 Data Subsetting Page



2. Under the User Name column, click the user name link TUTORIAL. As [Figure 12–31](#) displays, the Data Subsetting Parameters page appears.

Figure 12–31 Data Subsetting Parameters Page



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*.

At this stage, you have successfully administered the To Do List Application.

12.6 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 12.6.1, "Step 1: Installing the Mobile Client for Web-to-Go"](#)
- [Section 12.6.2, "Step 2: Logging into the Mobile Client for Web-to-Go"](#)
- [Section 12.6.3, "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.

12.6.1 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.

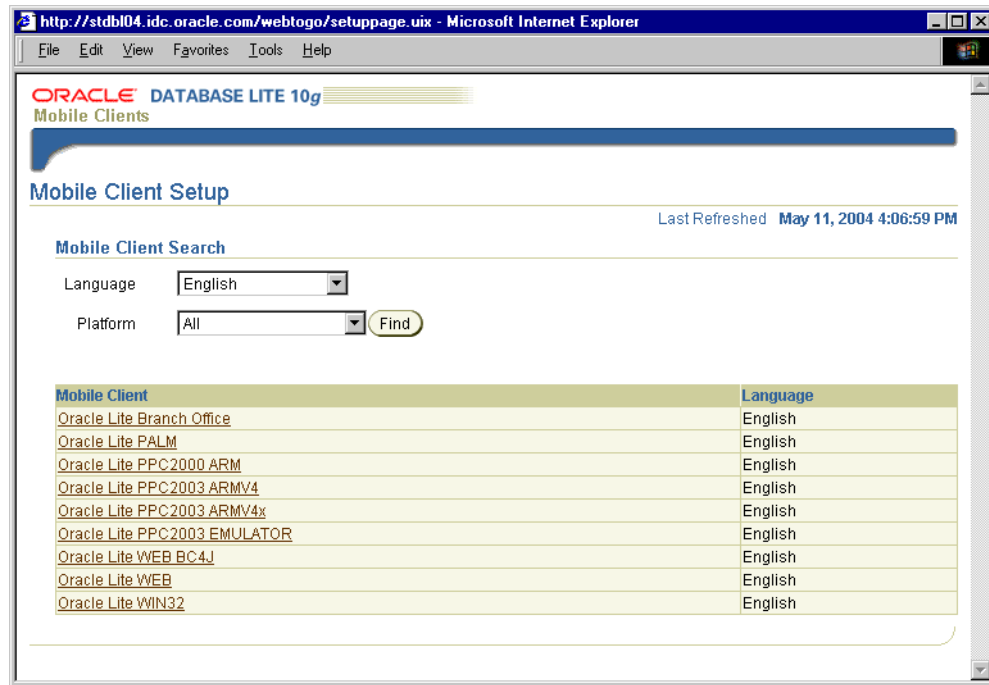
12.6.1.1 Required Action

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.

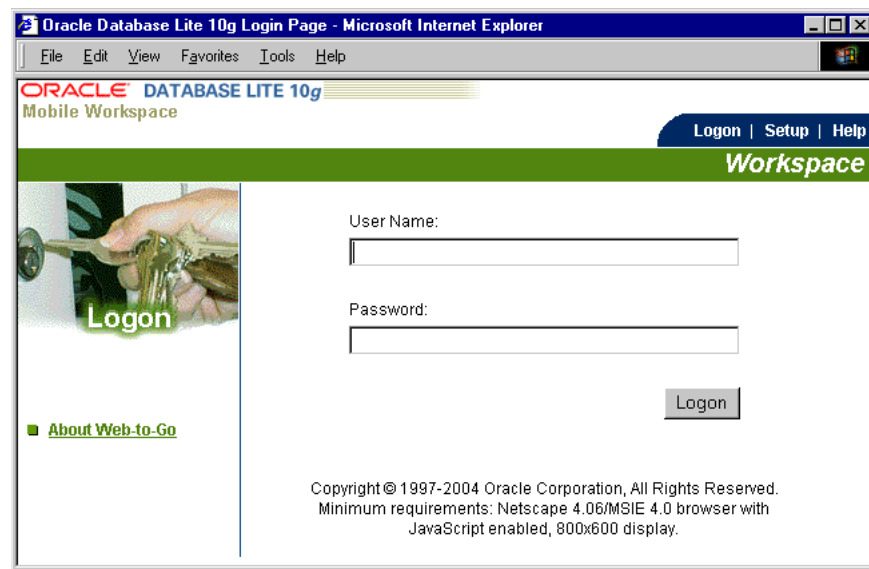
`http://<mobile_server>/webtogo/setup`
2. As [Figure 12-32](#) displays, the Mobile Client Setup page lists a set of mobile clients by platform. To download the Mobile Client for Web-to-Go setup program, click the corresponding Mobile Client link.

Figure 12–32 Mobile Client Setup Page



Note: While installing the Mobile Client, you will be prompted for the User name and Password. Enter *administrator* as the user name and *admin* 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 *administrator* as the User Name and *admin* 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 12–33](#) displays.

Figure 12-33 Mobile Manager Login Page

12.6.2 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.

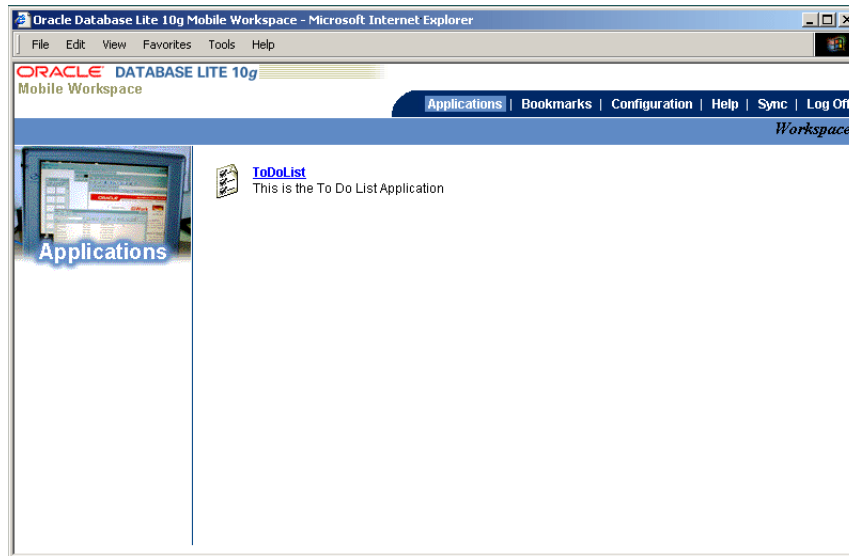
12.6.2.1 Required Action

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`

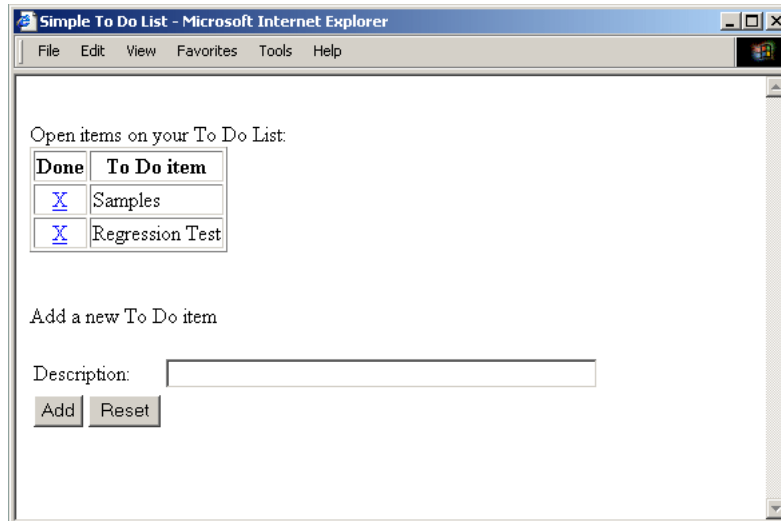
1. Log on to Web-to-Go using `tutorial` as the User Name and `tutorial` as the password.
2. As you are logging into the Mobile Client for Web-to-Go for the first time, you must complete the initial setup process. The client initialization page appears and displays a confirmation message. "The Web-to-Go 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`, as [Figure 12-34](#) displays.

Figure 12–34 Completion of the Synchronization Process



- Click the To Do List application icon. As [Figure 12–35](#) displays, Web-to-Go launches the To Do List application in your browser.

Figure 12–35 The To Do List Application



- Enter a new To Do item and save it in the database. Click **Add**.
- Exit the application by closing the browser window. This action returns you to the workspace.

12.6.3 Step 3: Synchronizing the Mobile Client for Web-to-Go

In this step, you synchronize the Mobile Client for Web-to-Go.

12.6.3.1 Required Action

To synchronize the Mobile Client for Web-to-Go with the Mobile Server, perform the following steps.

1. As [Figure 12–36](#) displays, click the Sync tab in the upper right corner of the workspace.

Figure 12–36 Sync Tab Location



The Mobile Client for Web-to-Go synchronizes the application and all of your data to the Oracle 10g Database. The workspace appears when the synchronization process has completed.

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:

- [Section 13.1, "Overview"](#)
- [Section 13.2, "Developing the Application"](#)
- [Section 13.3, "Packaging the JSP Application"](#)
- [Section 13.4, "Publishing and Configuring the JSP Application from the Mobile Manager"](#)
- [Section 13.5, "Testing the BC4J Application"](#)
- [Section 13.6, "Running the BC4J Application on the Mobile Client for Web-to-Go"](#)
- [Section 13.7, "Deploying the Sample Application"](#)

13.1 Overview

Oracle's BC4J (Business Components for Java) is a part of Oracle9i JDeveloper's IDE (Integrated Development Environment), and provides Java developers with the tools to create and manage reusable Java components.

BC4J offers a standards based, server side Java and XML framework for developers who 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, namely, Entity Objects, Associations, View Objects, View Links, and Application Modules. Each of these components is interrelated to the other components, thereby enabling 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, enabling Web-to-Go developers to focus on the business logic instead of spending their time on database related components during business application development.

The sample BC4J application which is used in this tutorial stores its items in a relational database. It maintains employee details.

13.1.1 Before You Start

Before you start developing business components in Java, you must ensure that the development computer meets the requirements specified below.

13.1.1.1 Development Computer Requirements

[Table 13–1](#) lists configuration and installation requirements for the development computer.

Table 13–1 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 1.3.1 or higher.
Installed Oracle Components	Mobile Server or Mobile Development Kit (Oracle Database Lite CD-ROM) Oracle 8.1.7 or higher Oracle9i JDeveloper, Release 9.0.3.

Note: The BC4J tutorial is shipped with the Mobile Development Kit as a JAR file named **9iLite_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 13.7, "Deploying the Sample Application"](#). If you want to develop the same application (as packaged in 9iLite_BC4J_Tutorial.jar), follow the steps from [Section 13.2, "Developing the Application"](#) to [Section 13.6, "Running the BC4J Application on the Mobile Client for Web-to-Go"](#), and then continue to [Section 13.7, "Deploying the Sample Application"](#).

13.2 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. Create a database connection.
2. Create the BC4J component.
3. Configure the BC4J component to use the WTGJdbc connection.
4. Build and deploy the BC4J component as a simple archive.
5. Write the JSP application to access the BC4J component.
6. Deploy the JSP application as a simple archive.
7. Deploy the BC4J component into the Mobile Server.

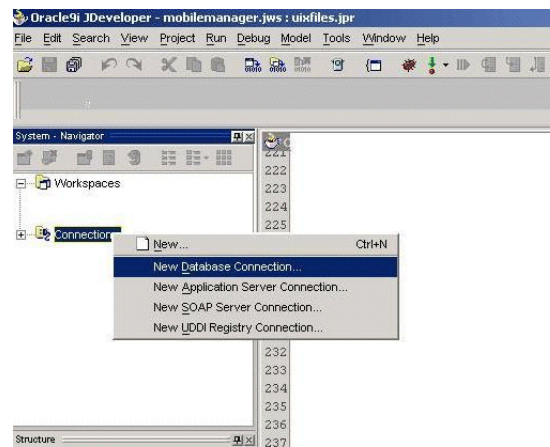
13.2.1 Creating the Database Connection

You must create two database connections: `tutorialConn` and `WTGJdbc` respectively. The `tutorialConn` connection connects to the primary Oracle database using the `oracle.jdbc.driver.OracleDriver` for developing and testing the application. The `WTGJdbc` connection connects to Oracle Database Lite using the `oracle.lite.web.WTGJdbcDriver`. You can change the connection from `tutorialConn` to `WTGJdbc` before deploying the application as a simple archive. The `WTGJdbc` connection is used during application deployment and uses a different driver.

To create the `tutorialConn` connection, perform the following steps.

1. In JDeveloper's **System Navigator** panel and as displayed in [Figure 13-1](#), right-click the **Connections** node and choose the **New Database Connection** option.

Figure 13-1 Choosing a New Database Connection

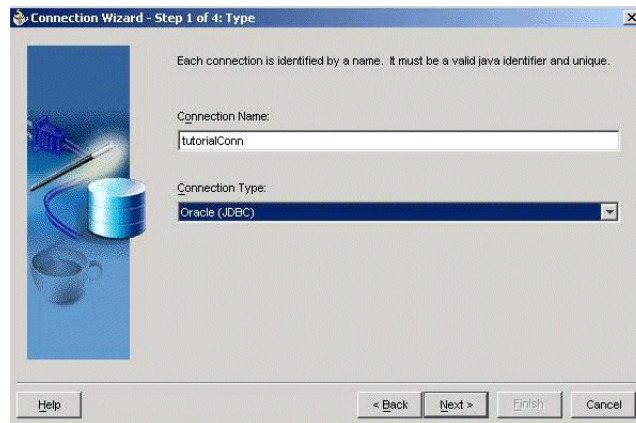


As displayed in [Figure 13-2](#), the Connection Wizard's Welcome panel appears.

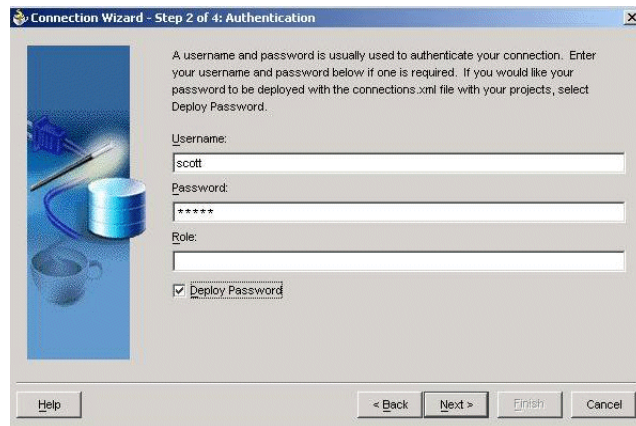
Figure 13-2 Welcome Panel - Connection Wizard



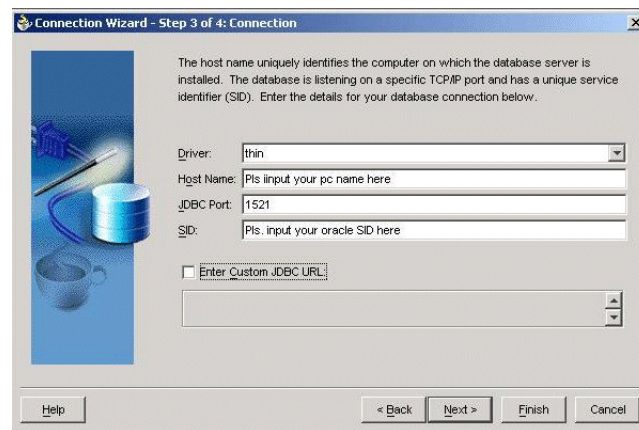
2. Click **Next**. As displayed in [Figure 13-3](#), the Connection Wizard - Step 1 of 4: Type panel appears. Create a connection named **tutorialConn** and choose **Oracle (JDBC)** from the Connection Type list.

Figure 13–3 Connection Wizard - Step 1 of 4: Type

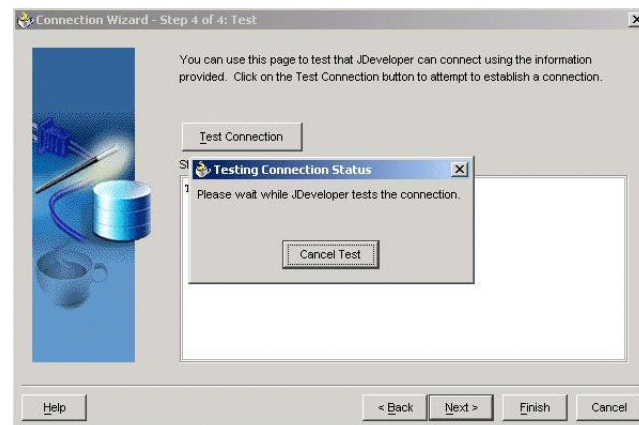
3. Click **Next**. As displayed in [Figure 13–4](#), the Connection Wizard - Step 2 of 4: Authentication panel appears. Enter **scott** as the user name and **tiger** as the password. Select the Deploy Password box.

Figure 13–4 Connection Wizard - Step 2 of 4: Authentication Panel

4. Click **Next**. As displayed in [Figure 13–5](#), the Connection Wizard - Step 3 of 4: Connection panel appears. 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.

Figure 13–5 Connection Wizard - Step 3 of 4: Connection Panel

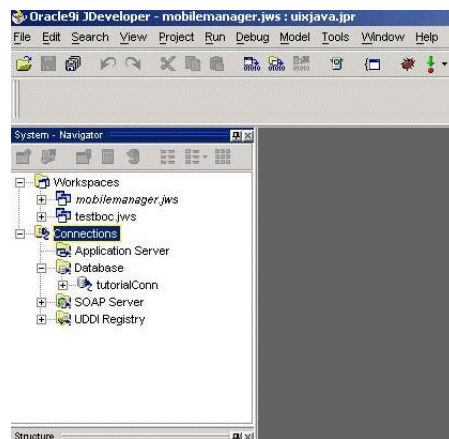
5. Click **Next**. As displayed in [Figure 13–6](#), the Connection Wizard - Step 4 of 4: Test panel appears. Click **Test Connection**. The Connection Wizard displays a connection status message.

Figure 13–6 Connection Wizard - Step 4 of 4: Test Panel

6. As displayed in [Figure 13–7](#), you have finished creating the tutorialConnection.

Figure 13–7 Connection Wizard - Finish Panel

As displayed in [Figure 13–8](#), The **tutorialConnection** icon appears in the System Navigator window under the Connections node.

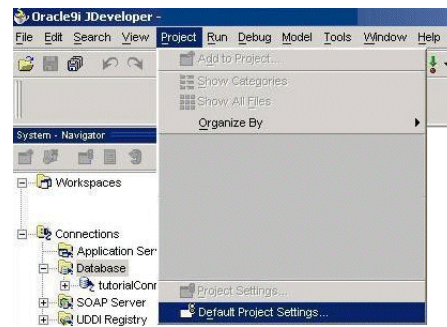
Figure 13–8 Tutorial Connection Icon in the System Navigator

[Table 13–2](#) summarizes values that you must enter or choose in the Connection Wizard.

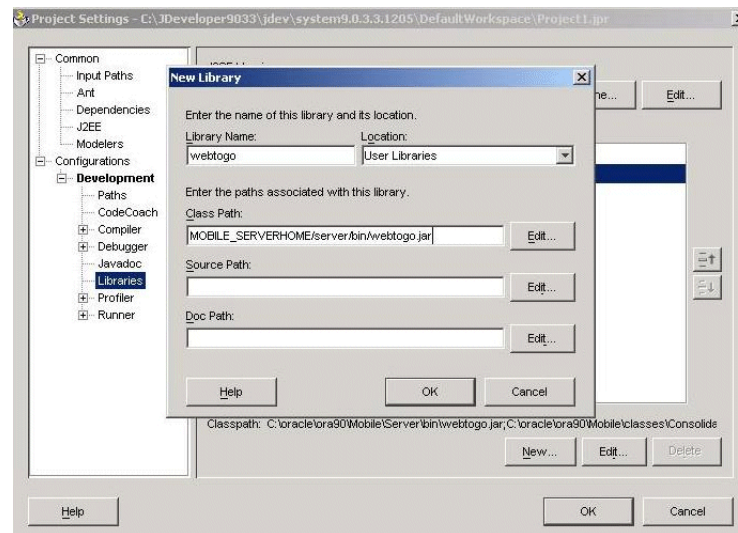
Table 13–2 TutorialConn - Connection Wizard Description

Field Name	Value
Connection Name	tutorialConn
User name	scott
Password	tiger
Select a JDBC Driver	Thin
SID	Your Oracle database SID

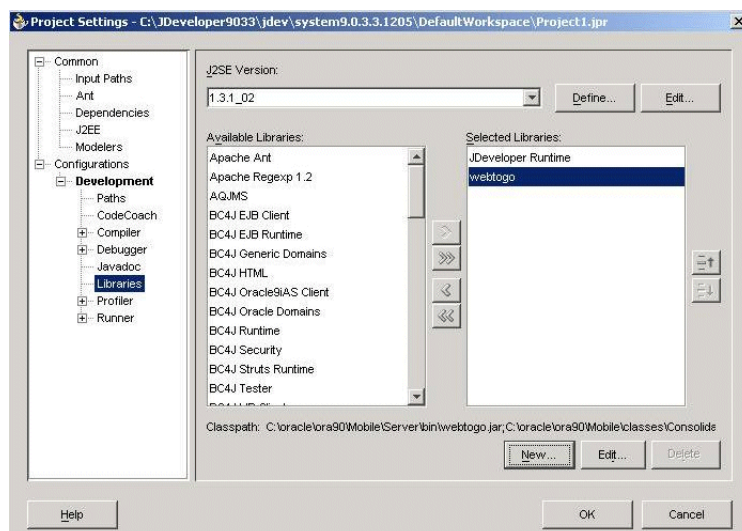
7. To create the WTGJdbc connection, you must configure the project settings and include the Oracle Database Lite user library named `webtogo.jar`. Start JDeveloper and click the **Project** menu. As displayed in [Figure 13–9](#), select the **Default Project Settings** option.

Figure 13–9 Choosing Default Project Settings

8. 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 13–10](#).
`mobile_serverhome/server/bin/webtogo.jar`

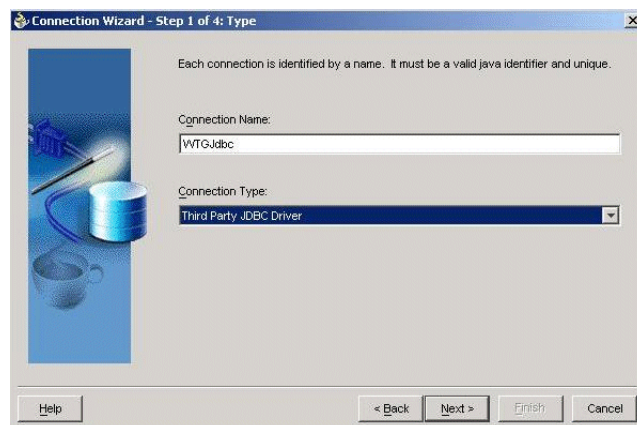
Figure 13–10 Adding a New Library and Classpath

9. After creating the new user library webtogo, move the library from the Available Libraries list to the Selected Libraries list as displayed in [Figure 13–11](#).

Figure 13–11 Moving the Webtogo Library to the Selected Libraries List

After configuring project settings as mentioned in this step, you can create the WTGJdbc connection using the same method that you used to create tutorialConnection.

10. To create the WTGJdbc connection, start JDeveloper and right-click the Connection object. As displayed in [Figure 13–12](#), 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.

Figure 13–12 Connection Wizard - Step 1 of 4: Type Panel

11. Click **Next**. As displayed in [Figure 13–13](#), the Connection Wizard - Step 2 of 4: Authentication panel appears. Do not enter any values in this panel.

Figure 13-13 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:

Password:

Role:

do not type any password or role in the given fields

☐ Deploy Password

Help < Back Next > Finish Cancel

12. Click **Next**. As displayed in [Figure 13-14](#), the Connection Wizard - Step 3 of 4: Connection panel appears.

Figure 13-14 Connection Wizard - Step 3 of 4: Connection Panel

Connection Wizard - Step 3 of 4: Connection

You need a driver name, and JDBC URL to connect using a third part JDBC driver. Enter the class name and URL below.

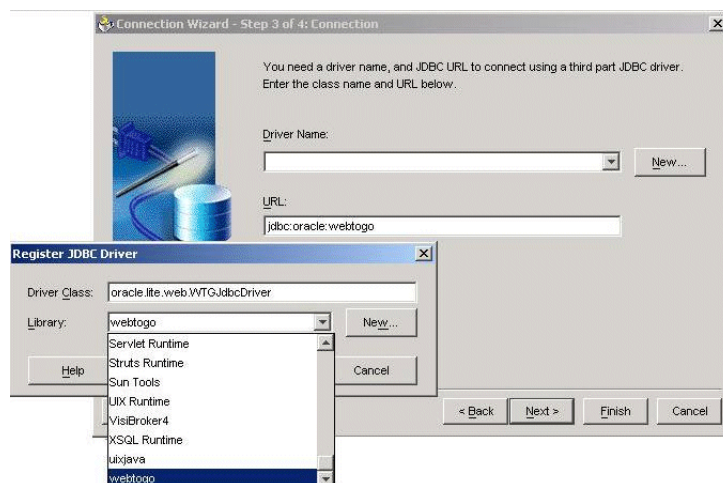
Driver Name: New...

URL:

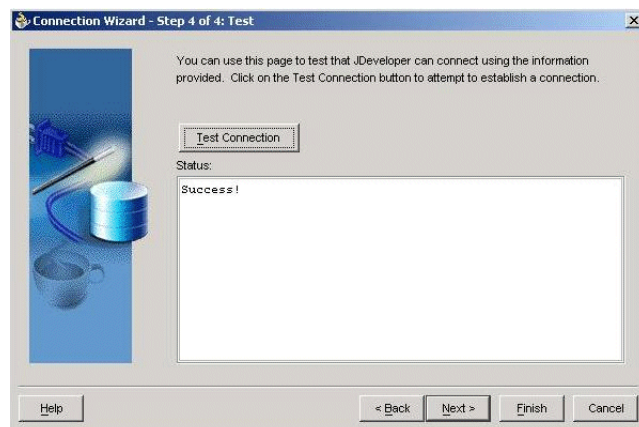
Help < Back Next > Finish Cancel

13. Click **New**. As [Figure 13-15](#) displays, the Register JDBC Driver dialog appears. Enter **oracle.lite.web.WTGJdbcDriver** as the Driver Class. Choose **webtogo** from the Library list. Enter the following URL.

```
jdbc:oracle:webtogo
```

Figure 13–15 JDBC Driver Dialog

14. Click **Next**. As [Figure 13–16](#) displays, the Connection Wizard - Step 4 of 4: Test panel appears. To test your WTGJdbc connection, click **Test Connection**. The Status box displays that the WTGJdbc connection has been created successfully.

Figure 13–16 Connection Wizard - Step 4 of 4: Test Panel

[Table 13–3](#) describes values that must be entered in the Connection Wizard to create the WTGJdbc connection.

Table 13–3 WTGJdbc Connection - Connection Wizard Description

Field Name	Values
Connection Name	WTGJdbc
Select a JDBC Driver	Third Party JDBC Driver
Class Name	oracle.jdbc.driver.OracleDriver
Datasource URL	jdbc:oracle:webtogo

Note: In the Connection Wizard, enter values as specified in [Table 13–2](#) and [Table 13–3](#) only. Retain all other values as default values.

13.2.2 Creating the BC4J Component

Using Oracle9i JDeveloper, you can create the BC4J component named "tutorialapp".

To create the BC4J component named "tutorialapp", perform the following steps.

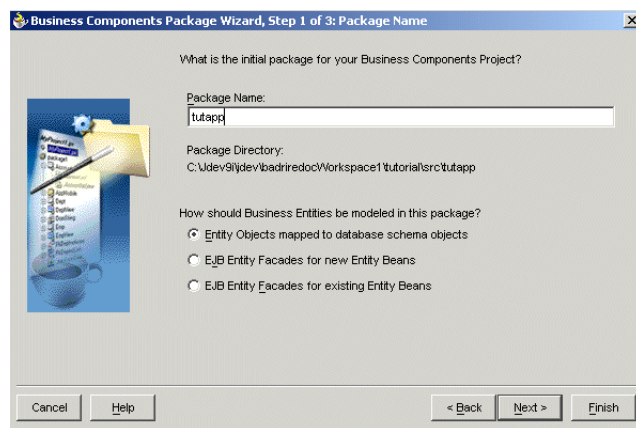
1. In Oracle9i JDeveloper, select **New** from the **File** menu. In the New dialog box that appears, the options named "Projects" in the left panel and "Empty Project" in the right panel are pre-selected as defaults. Click **OK**. Oracle9i JDeveloper creates a new empty project named "Project.jpr".
2. Rename "Project.jpr" to "tutorialapp.jpr", which creates a new project by that name.
3. Right click "tutorialapp.jpr" in the Oracle9i JDeveloper workspace. Select the "New Business Components Package..." option. The "Business Components Package Wizard, Welcome" dialog appears. Click **Next**.

[Figure 13–17](#) displays the Business Components Package Wizard, Welcome dialog.

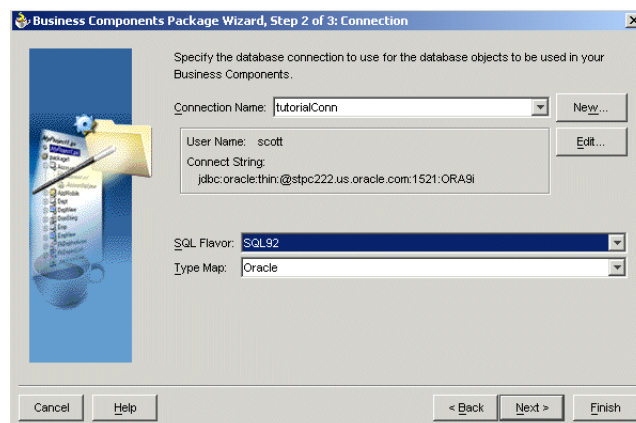
Figure 13–17 The Business Components Package Wizard, Welcome Dialog



4. The "Business Components Package Wizard, Step 1 of 3: Package Name" dialog appears, as illustrated in [Figure 13–18](#). In the "Package Name" field, enter tutapp. Click **Next**.

Figure 13–18 Business Components Package Wizard Step 1 of 3: Package Name

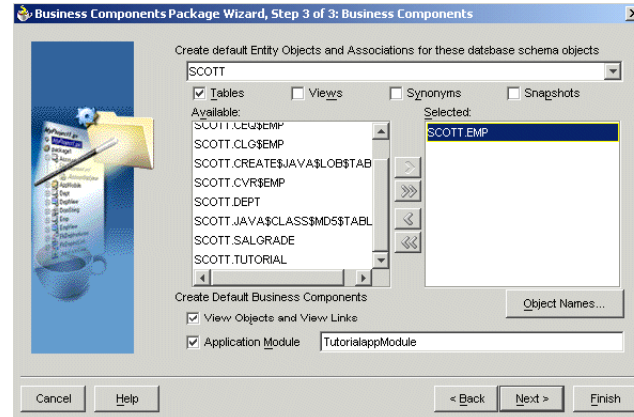
5. The "Business Components Package Wizard, Step 2 of 3: Connection" dialog appears, as depicted in [Figure 13–19](#). Select the values that are listed in [Table 13–4](#) and click **Next**.

Figure 13–19 Business Components Package Wizard Step 2 of 3: Connection**Table 13–4 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 13–20](#). Click **Finish**.

Figure 13–20 Business Components Package Wizard, Step 3 of 3: Business Components Dialog



7. At this stage, Oracle9i JDeveloper creates the BC4J component named "tutorialapp".

13.2.3 Configuring the BC4J Component to Use the WTGJdbc Connection

To configure the BC4J component to use the WTGJdbc connection, perform the following steps.

1. Right-click on the TutorialAppModule and double-click on 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.

13.2.4 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.
3. Click **Next**. 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**. The **tutorialapp.bcdeploy** file is created under **tutorial.jpr**.
5. Right-click the file **tutorialapp.bcdeploy** and select **Deploy**.
6. JDeveloper creates two jar files namely **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.

13.2.5 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 Oracle9i JDeveloper, select the **Empty Project** option under the 'File' menu. The system automatically creates a new empty project called `MyProject.jpr`.
2. Under the 'File' menu, select the **Rename...** option and rename `MyProject.jpr` to `tutorialclientapp.jpr`.
3. Click the **tutorialclientapp.jpr** file in the Oracle9i JDeveloper workspace. Click the File menu and select **New**. Click the Web Tier option and select **JSP for Business Components**. Click **OK**. The Business Components JSP Application Wizard appears. Click **Next**.
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.
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.
6. Click **Next**. The 'BC4J Client Data Model Definition Wizard: Step 1 of 2: Definition' appears, as displayed in [Figure 13–21](#), [Figure 13–22](#), [Figure 13–23](#), and [Figure 13–24](#).

Figure 13–21 Business Components JSP Application Wizard - Welcome Dialog

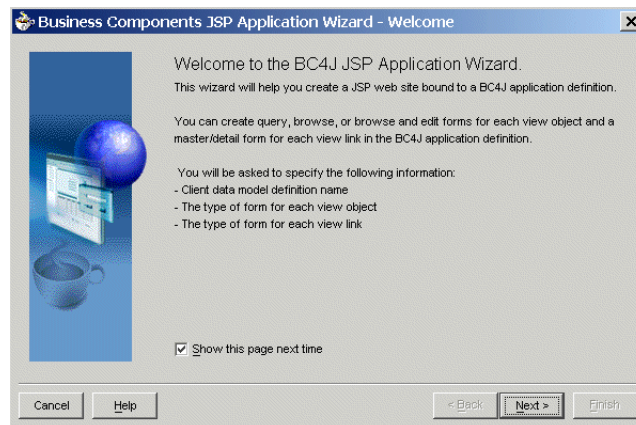


Figure 13–22 Business Components JSP Application Wizard - Step 1 of 3: Data Definition Dialog

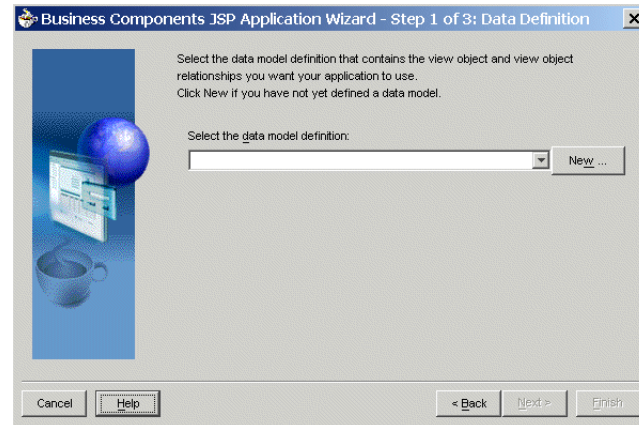


Figure 13–23 BC4J Client Data Model Definition Wizard - Welcome Dialog

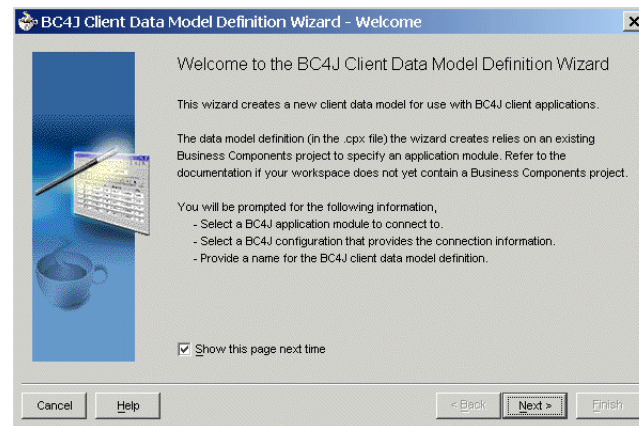
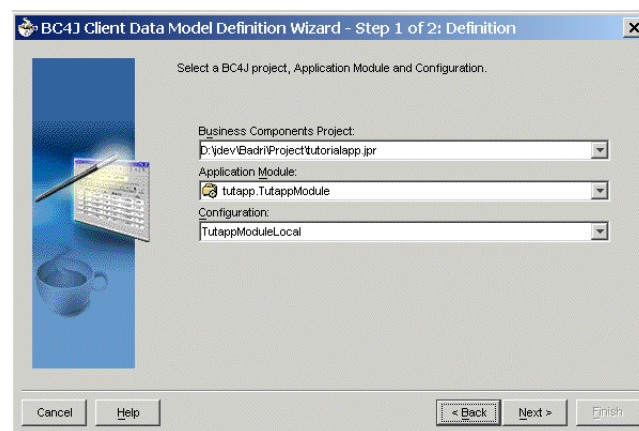


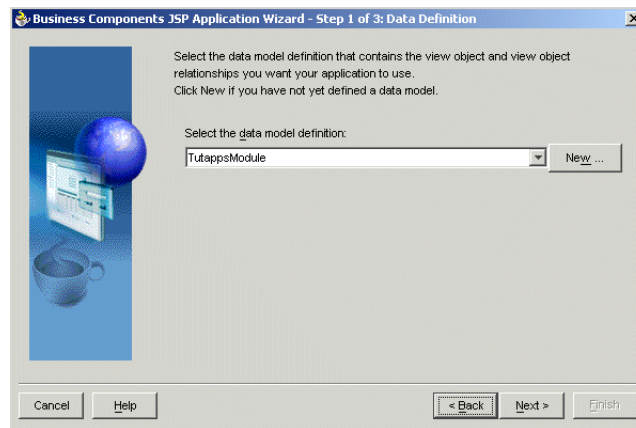
Figure 13–24 BC4J Client Data Model Definition Wizard, Step 1 of 2: Definition Dialog



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 13-25](#). Click **Next**.

Figure 13-25 Business Components JSP Application Wizard - Step 1 of 3: Data Definition Dialog



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**.

13.2.6 Deploying the JSP Application as a Simple Archive

To deploy the JSP application as a simple archive, perform the following steps.

1. In Oracle9i JDeveloper, click the `tutorialclientapp.jpr` file and select the file named `tutappclient_jpr_war.deploy`.
2. Select 'Deploy to WAR file'. The file `tutappclient_jpr_war.war` gets created. To track the deployment location, check the 'Deployment Log' text area in Oracle9i JDeveloper.

13.3 Packaging the JSP Application

To package the JSP application, perform the following steps.

1. Create a sub-directory called `bc4jtutapp` under the following location.
`<Oracle_Home>\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
from
<%@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 13–5](#).

Table 13–5 Packaging Wizard Input Details

Screen	Input	Details
Platform	Web-To-Go	NA
Application	Application Name	BC4J 9iLite Tutorial Application
Application	Virtual Path	/bc4jtutorial
Application	Description	BC4J 9iLite 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.	NA

[Table 13–6](#) lists the servlet names and their corresponding classes that are created in the Packaging Wizard by default.

Table 13–6 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 13–7](#) lists server side and client side database values that you must specify in the Packaging Wizard.

Table 13–7 Database Values

Screen	Input	Details
Database	Server side Database User Name	scott

Table 13–7 (Cont.) Database Values

Screen	Input	Details
Database	Number of Connections	0
Database	Share Connections	Do not select this check box
Database	Client side Database Name	Client DB

- 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 13–8](#) lists values that you must specify in the Connect to Database dialog.

Table 13–8 Connect to Database Dialog Description

Field	Description
User Name	scott
Password	tiger
Database URL	jdbc:oracle:thin:@DatabaseHostMachineName:port:SID

- After specifying the Database Connection values, select "Emp" from the list of tables.
- Click "Edit" and change the weight to "1" from "0".
- You must retain the default values for Roles, Sequences, DDLs, and Registry fields.
- Package the application into a JAR file.

13.4 Publishing and Configuring the JSP Application from the Mobile Manager

To configure the JSP application from the Mobile Manager, perform the following steps.

- Using the Command Prompt window, enter `java -jar oc4j.jar`, to start the Mobile Server.
- Using the following URL, browse the local host.
`http://localhost:portnumber`
 If the above port number is other than 80, you must specify the appropriate port number.
- Login into the Mobile Server using the Administrator's user name and password.
- Click **Mobile Manager**.
- Click the **Applications** link and publish the JAR file that you just created. In the **Repository Directory** field, enter `/bc4jtutorial`.

13.5 Testing the BC4J Application

To test the BC4J application, login to the Mobile Server as a 'tutorial' user. In the 'tutorial' workspace, double-click the 'BC4J 9iLite Tutorial Application'. The Business

Components JSP Application window appears. Click **Emp View** and browse through the employee table records.

13.6 Running the BC4J Application on the Mobile Client for Web-to-Go

To run the BC4J application on the Mobile Client for Web-to-Go, perform the following steps.

1. Using the following URL, check the Database Server's 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 the user name and password.
5. After the client machine synchronizes the application and data from the server, click the 'BC4J 9iLite Tutorial Application' link to test the application on the client machine.

13.7 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, run the `bc4j.sql` script.
2. Publish the **9iLite_BC4J_Tutorial.jar** file. It is found under the following directory.
`<Oracle_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 the **Applications** link.
4. Click the '9iLite 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 a user named 'tutorial' and assign 'user' as the privilege.
7. Click the **Applications** link and click the '9iLite BC4J Application' link. To provide access to the '9iLite BC4J Application', click the **Access** link and provide access 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 '9iLite BC4J Application' link.
13. Click the '9iLite BC4J Application' link to access the BC4J tutorial application.

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_NAME	LOC#	
EMP	SS# NAME JOB_TITLE WORKS_IN	SS#	WORKS_IN references DEPT (DEPT#)
DEPT	DEPT# NAME BUDGET LOC MGR	DEPT#	LOC references LOCATION (LOC#) 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';
```

If the selectivity (selecting more than 10% of the rows) of the indexing columns is poor, an index may ruin performance. 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);
```

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.1, "Overview"](#)
- [Section B.2, "Oracle Database Lite Load APIs"](#)
- [Section B.3, "File Format"](#)
- [Section B.4, "Limitations"](#)

B.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 the *Oracle Database Lite Tools and Utilities Guide*. You can use the API calls presented in this document to make your own customizations.

B.2 Oracle Database Lite Load APIs

The Oracle Database Lite Load APIs include:

- [Section B.2.1, "Connecting to the Database: olConnect"](#)
- [Section B.2.2, "Disconnecting from the Database: olDisconnect"](#)
- [Section B.2.3, "Deleting All Rows from a Table: olTruncate"](#)
- [Section B.2.4, "Setting Parameters for Load and Dump Operations: olSet"](#)
- [Section B.2.5, "Loading Data: olLoad"](#)
- [Section B.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.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-1](#):

Table B-1 *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.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-2](#):

Table B-2 *olDisconnect Arguments*

Argument	Description
dbh	The current application handle.

Return Value

(short) integer error code

B.2.3 Deleting All Rows from a Table: olTruncate

This API can be used to delete all rows from an existing table.

Syntax

```
olError olTruncate (DBHandle dbh, char* table );
```

The arguments for `olTruncate` are listed in [Table B-3](#):

Table B–3 *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.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–4](#):

Table B–4 *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.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.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–5](#):

Table B–5 *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.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–6](#):

Table B–6 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.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.2.8 Linking

Linking use the file **olloader40.dll** and the library file **olloader40.lib**. Include these files in your project settings.

B.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.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 `o1Load` is NULL, the table name and list of columns in the header prevails, otherwise the table argument of `o1Load` prevails over the header.

B.3.2 Parameters

Header file parameters listed in [Table B-7](#) are not case sensitive.

Table B-7 Parameters

Parameter	Description
FILEFORMAT	Input and output file format. The following formats are supported: <ul style="list-style-type: none"> 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 <code>o1Load</code> , <code>o1Dump</code> , and <code>o1Truncate</code> commit. The default value is -1, not to commit at all. Value 0 commits at the end of the operation, and values above 0 commit after the specified number of rows.
HEADER	FALSE (the default). TRUE to create a header in the beginning of the file during <code>o1Dump</code> .

Table B–7 (Cont.) Parameters

Parameter	Description
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 <code>o1Dump</code> provides an error trying to write nulls in binary.
NONULL	TRUE (the default) when trying to read or write nulls <code>o1Load</code> and <code>o1Dump</code> 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.3.3 Data Format

The data format can be comma separated value (CSV), fixed ASCII, or binary. The following cases apply:

B.3.3.1 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, `o1Load` gives an error otherwise.

B.3.3.2 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–8](#) 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 `o1Load` returns an error.

Table B–8 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-8 (Cont.) Datatypes

Datatype	Description
NUMERIC (p, s)	<p>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)</p> <p>12.3 -> ' 012.30'</p> <p>-12.3 -> '-012.30'</p> <p>1.23 -> ' 001.23'</p> <p>-1.23 -> '-001.23'</p> <p>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, <code>o1Dump</code> 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 '-'; <code>o1Load</code> gives an error otherwise.</p>
DECIMAL (p, s)	The same as NUMERIC(p,s).
INTEGER	<p>Length of the field is 11 characters. A negative sign or space followed by 10 digits.</p> <p>Leading digits are filled with zeros.</p>
SMALLINT	Field length is 6 characters. Minus sign or space followed by 5 digits.
FLOAT	<p>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:</p> <p>0 -> ' 0000000000000000000000'</p> <p>-12.34 -> '-000000000000000000012.34'</p> <p>In SQL92 mode the E (exponent) is always present and there is only 1 digit before the decimal point. For example:</p> <p>0 -> ' 0000000000000000000000E0'</p> <p>-12.34 -> '-000000000000000001.234E10'</p>
REAL	The same format as for double precision except that the total field length is only 16 characters instead of 23.

Table B–8 (Cont.) Datatypes

Datatype	Description
DOUBLE PRECISION	<p>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.</p> <p>For example, in SQL92 mode:</p> <p>0 -> '00000000000000000000E0'</p> <p>-1.79E10 -> '-0000000000000001.79E10'</p> <p>12 -> '0000000000000001.2E10'</p> <p>For example, in Oracle mode:</p> <p>1.2E75 -> '0000000000000001.2E75'</p> <p>-1.33333 -> '-0000000000000001.33333'</p> <p>-1.79E10 -> '-0000000000017900000000'</p>
DATE	<p>In SQL92 mode: YYYY-MM-DD, 10 characters long, for example:</p> <p>October 1, 1999 -> 1999-10-01</p> <p>In Oracle mode the date is dumped as timestamp.</p> <p>If it is not the default date format parameter, the date format corresponds to the specified date format string, for example:</p> <p>DATEFORMAT = "YYYYMMDD"</p> <p>October 1, 1999 -> 19991001</p>
TIME	<p>HH:MM:SS, 8 characters long, for example:</p> <p>5:01:58 p.m. is 17:01:58</p>
TIMESTAMP	<p>Date format, space, time format, dot, 6 digits after dot (precision of microseconds), total length of 26 characters:</p> <p>YYYY-MM-DD HH:MM:SS.SSSSSS</p> <p>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.</p>

B.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.

Web-to-Go Sample Applications

This appendix contains Web-to-Go sample applications. Topics include:

- [Section C.1, "Introduction"](#)
- [Section C.2, "Sample 1 - Hello World"](#)
- [Section C.3, "Sample 3 - Recording Tracker"](#)
- [Section C.4, "Sample 4 - Hello Applet"](#)
- [Section C.5, "Sample 6 - Image Gallery"](#)
- [Section C.6, "Sample 7 - Employee Data Applet"](#)

C.1 Introduction

Web-to-Go contains five sample programs that are installed with the Mobile Development Kit for Web-to-Go or the Mobile Server.

C.1.1 The Mobile Server

You can install the demos during Mobile Server installation or by running the batch file **instdemo.bat**. This batch file is located in the following directory.

```
<Oracle_home>\Mobile\Server\Samples
```

The command syntax is as follows.

```
instdemo.bat [SYSTEM_password] [repository_owner] [repository_password]
```

For example,

```
instdemo manager mobileadmin manager
```

C.1.2 The Mobile Development Kit for Web-to-Go

You can install the demos by running the batch file **sdkdemos.bat**. This batch file is located in the following directory.

```
<Oracle_home>\Mobile\sdk\wtg sdk\src\sdkdemos.bat
```

C.1.3 Accessing Sample Programs from the Mobile Development Kit for Web-to-Go

The Mobile Development Kit for Web-to-Go is bundled with sample programs that you can access from the following URL.

```
http://<mobile_server>:7070/
```

The browser displays icons for different sample programs. To launch a sample program, click the required icon for the corresponding program.

C.1.4 Accessing Sample Programs from the Mobile Server

As [Table C–1](#) describes, the Mobile Server automatically creates the following sample users when you install Web-to-Go demos.

Table C–1 Sample Users

User	Password
john	john
jack	jack
jane	jane

The above listed sample users can access the sample programs by logging on to the Mobile Server and clicking any of the sample application icons in the workspace.

C.2 Sample 1 - Hello World

Sample 1, Hello World is a servlet that returns a simple HTML page to the browser. It demonstrates the basic methods of `HttpServlet` and demonstrates the difference between the POST and GET methods.

C.2.1 Source Code Location

The source code location varies depending on whether you installed the Mobile Server or the Mobile Development Kit for Web-to-Go. [Table C–2](#) describes these locations.

Table C–2 Source Code Locations

Installation Type	Source Code Locations
Mobile Server	<Oracle_home>\Mobile\Server\samples\sample3\src
Mobile Development Kit for Web-to-Go	The Java source code can be found in the following location. <Oracle_home>\Mobile\Sdk\wtgSDK\src\sample3\servlets

C.2.2 Application Files

Sample 1 contains the **HelloWorld.java** file. It is the source code for the HelloWorld servlet.

C.3 Sample 3 - Recording Tracker

Sample 3, the Recording Tracker demonstrates how servlets can be used to maintain a database with recording information. The program allows users to search the database and enter recordings and tracks for a recording. Although the recordings are stored in the RECORDINGS table, users can only see their corresponding data when they access this table.

C.3.1 Using Sample 3

When a user goes offline, Web-to-Go automatically creates snapshots on the local client to hold a copy of the data. You can choose the rows that are replicated to the snapshot

by adding a subquery to the snapshot definition. This enables Web-to-Go to synchronize specific rows for the user to the local client. In Sample 3, both John and Jack have access to the same data. Jane can only see her own data and no other user has access to it. For more information on setting up a snapshot subquery, see the *Oracle Database Lite Administration and Deployment Guide*.

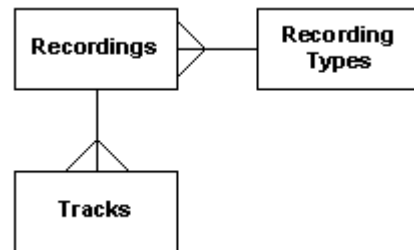
C.3.2 Sample 3 Database Tables

Sample 3 contains the following database tables.

- RECORDINGS
- RECORDING TYPES
- TRACKS

As [Figure C–1](#) displays, the database tables are illustrated in the following entity relationship diagram.

Figure C–1 Entity Relationship Diagram



C.3.3 Sample 3 Servlets

Sample 3 contains six different Java servlets. These servlets demonstrate two ways of generating HTML. The `DisplayRecord` servlet uses the `oracle.html` package to generate the entire HTML file. The `DisplayMasterDetail`, `ListSearchResults`, and `SimpleList` servlets generate HTML using the base class `oracle.lite.web.html.TemplateParser` and a static HTML template. In both cases, data is displayed in HTML using the `DBTable` class. Data changes are processed by the generic servlet `ProcessForm`, which is part of the `oracle.lite.web.html` package. `DeleteDetail` and `DeleteMasterDetail` extend the class `oracle.lite.web.html.DeleteRecords`. These servlets execute a request and then redirect the browser to another URL.

C.3.4 Sample 3 Resource Bundle

The Recording Tracker program also illustrates the use of the `ListResourceBundle` class to manage resources for locale specific strings. By isolating all text strings in a resource bundle, you can write programs that can easily be translated into other languages, or modified to add support for more languages. See `SampleResources.java` for more detailed information.

C.3.5 Source Code Location

The source code location varies depending on whether you installed the Mobile Server or the Mobile Development Kit for Web-to-Go. [Table C–3](#) describes these locations.

Table C-3 Source Code Locations

Installation Type	Source Code Locations
Mobile Server	<Oracle_home>\Mobile\Server\samples\sample1\src
Mobile Development Kit for Web-to-Go	The Java source code can be found in the following location. <Oracle_home>\Mobile\Sdk\wtgSDK\src\sample1\servlets

C.3.6 Application Files

Sample 3 contains the following application files.

File	Description
EnterSearchCriteria.html	The Static HTML file.
sample3.gif	The Sample 3 icon that appears in the Web-to-Go workspace.
sample3.html	The start page for the application.
sample3.sql	The SQL Script that installs the sample3 database objects.
table.sql	The SQL script that creates the sample3 database tables.
insert.sql	The SQL script that populates the sample3 database tables with data.
drop.sql	The SQL script that drops the sample3 database tables.
SampeProgram3.java	The source code that contains static definitions for the Sample 3 application.
SampleResources.java	The source code for the String Resources used by the servlets.
DisplayMasterDetail.java	The source code for the DisplayMasterDetail servlet.
DisplayRecord.java	The source code for the DisplayRecord servlet.
SimpleList.java	The source code for the SimpleList servlet.
ListSearchResults.java	The source code for the ListSearchResults servlet.
DeleteDetail.java	The source code for the DeleteDetail servlet.
DeleteMasterDetail.java	The source code for the DeleteMasterDetail servlet.
DisplayMasterDetail.html	The HTML template, used by the DisplayMasterDetail Servlet.
ListSearchResults.html	The HTML template, used by the ListSearchResults Servlet.
SimpleList.html	The HTML template, used by the SimpleList Servlet.

C.4 Sample 4 - Hello Applet

Sample 4, the Hello Applet illustrates how applets and servlets can communicate with each other. A Java applet calls a servlet running on the Mobile Server. The servlet responds by sending a string to the applet, which the applet then displays.

C.4.1 Sample 4 Servlets

Sample 4 contains two servlets. The `AppServlet` servlet generates HTML that instructs the browser to launch the applet. This HTML includes applet parameters that

contain the Mobile Server session information. The `HelloServlet` is called by the applet as part of the applet/servlet communication.

C.4.2 Source Code Location

The source code location varies depending on whether you installed the Mobile Server or the Mobile Development Kit for Web-to-Go. [Table C-4](#) describes these locations.

Table C-4 Source Code Locations

Installation Type	Source Code Locations
Mobile Server	<Oracle_home>\Mobile\Server\samples\sample1\src
Mobile Development Kit for Web-to-Go	The Java source code can be found in the following location. <Oracle_home>\Mobile\Sdk\wtgSDK\src\sample1\servlets

C.4.3 Application Files

Sample 4 contains the following application files.

File	Description
<code>Sample4.gif</code>	The Sample 4 icon that appears in the workspace.
<code>Sample4.html</code>	The start page for the application.
<code>HelloApplet.java</code>	The Java source code for the applet.
<code>AppServlet.java</code>	The Java source code for the <code>AppServlet</code> servlet.
<code>HelloServlet.java</code>	The Java source code for the <code>HelloServlet</code> servlet.

C.5 Sample 6 - Image Gallery

Sample 6, the Image Gallery demonstrates how to store binary data in the database without using the `LONG` datatype. When the sample program uploads images to the Mobile Server, it separates them into 255 byte chunks. As a result, you can store the images in a `RAW` datatype column.

C.5.1 Source Code Location

The source code location varies depending on whether you installed the Mobile Server or the Mobile Development Kit for Web-to-Go. [Table C-5](#) describes these locations.

Table C-5 Source Code Locations

Installation Type	Source Code Locations
Mobile Server	<Oracle_home>\Mobile\Server\samples\sample1\src
Mobile Development Kit for Web-to-Go	The Java source code can be found in the following location. <Oracle_home>\Mobile\Sdk\wtgSDK\src\sample1\servlets

C.5.2 Application Files

Sample 6 contains the following application files.

File	Description
<code>sample6.gif</code>	Icon for the application, used in the Web-to-Go workspace.
<code>loadImage.html</code>	HTML form to upload an image.
<code>DeleteImage.java</code>	Source code for the <code>DeleteImage</code> servlet.
<code>GetImage.java</code>	Source code for the <code>GetImage</code> servlet.
<code>Upload.java</code>	Source code for the <code>Upload</code> servlet.
<code>ImageList.java</code>	Source code for the <code>ImageList</code> servlet.
<code>ViewImage.java</code>	Source code for the <code>ViewImage</code> servlet.
<code>RawImage.java</code>	The Source code for the <code>RawImage</code> servlet.
<code>ImageList.html</code>	The HTML Template used by the <code>ImageList</code> servlet.
<code>ViewImage.html</code>	The HTML Template used by the <code>ViewList</code> servlet.
<code>sample6.sql</code>	The SQL script that installs the sample6 database objects.
<code>table.sql</code>	The SQL script that creates the sample6 database tables.
<code>drop.sql</code>	The SQL script that drops the sample6 database tables.

C.6 Sample 7 - Employee Data Applet

Sample 7, the Employee Data Applet, demonstrates how to use JDBC in an applet. The applet connects to the database using the `oracle.lite.web.applet.AppletProxy` class. This class automatically returns a database connection to the appropriate database depending on the user's connection mode. In online mode, the `oracle.lite.web.applet.AppletProxy` class returns a connection to the Oracle database. In offline mode, or when using the Mobile Development Kit for Web-to-Go, the class returns a connection to Oracle Database Lite.

Running Sample 7

To successfully run Sample 7, the client **webtogo.ora** file must be modified to support Applet JDBC connections. The **webtogo.ora** file is available at the following location.

```
<WebtoGo_Home>\bin\webtogo.ora
```

Uncomment the following line in the **webtogo.ora** file.

```
#APPLET_SUPPORT_ENABLE=YES
```

Ensure that the file **olite40.jar** is recognized by the chosen browser as being in the **CLASSPATH**.

Normally, the **System CLASSPATH** is the appropriate location to add the file. However, on some occasions the browser does not recognize changes to the **System CLASSPATH**, but does recognize changes to the **User CLASSPATH**. You must try the **System CLASSPATH** first. As a back up option, you can try the **User CLASSPATH**.

C.6.1 Source Code Location

The source code location varies depending on whether you installed the Mobile Server or the Mobile Development Kit for Web-to-Go. [Table C-6](#) describes these locations.

Table C-6 Source Code Locations

Installation Type	Source Code Locations
Mobile Server	<Oracle_home>\Mobile\Server\samples\sample1\src
Mobile Development Kit for Web-to-Go	The Java source code can be found in the following location. <Oracle_home>\Mobile\Sdk\wtgSDK\src\sample1\servlets

C.6.2 Application Files

Sample 7 contains the following application files.

File	Description
sample7.gif	The application icon that appears in the workspace.
sample7.html	The application start page.
AppApplet.java	The Java source code for the applet.
ErrorDialog.java	The Java source code for error dialog.
sample7.sql	The SQL Script to install the sample7 database objects.
table.sql	The SQL script that creates the sample7 database tables.
insert.sql	The SQL script that populates the sample7 database tables with data.
drop.sql	The SQL script that drops any old tables.

ODBC Support on Palm

This document describes the Open Database Connectivity (ODBC) support provided in Oracle Database Lite 10g for the Palm OS Platform. Topics include:

- [Section D.1, "ODBC Support"](#)

D.1 ODBC Support

For the Palm OS platform, Oracle Database Lite 10g 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 D–1](#).

Table D–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 D–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.

D.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 D–2](#):

Table D–2 SQLAllocConnect Arguments

Type	Name	Description
SQLHENV	<code>hEnv</code>	Environment handle. If set to NULL, creates a new environment.
SQLHDBC*	<code>hDbc</code>	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.

D.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 D-3](#):

Table D-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.

D.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 D-4](#):

Table D–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 D–5](#):

Table D–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.

D.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 D–6](#):

Table D–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.

D.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 D–7](#):

Table D–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.

D.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 D–8](#):

Table D–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.

D.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 D–9](#):

Table D–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.

D.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 D–10](#):

Table D-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.

D.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 `SQLAllocHandle` function.

Syntax

```
RETCODE SQLConnect( hConn, dbName, dbNameLen, userName, userNameLen, auth, authLen
)
```

Arguments

The arguments for `SQLConnect` are listed in [Table D-11](#):

Table D-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.

D.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 D-12](#):

Table D-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.

D.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 D-13](#):

Table D-13 SQLBindParameter 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 <code>SQL_PARAM_INPUT</code> is supported.

Table D–13 (Cont.) SQLBindParameter Arguments

Type	Name	Description
SQLSMALLINT	cType	The C datatype of the parameter.
SQLSMALLINT	sqlType	The SQL datatype of the parameter.
SQLINTEGER	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.

D.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 D–14](#):

Table D–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.

D.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 D-15](#):

Table D-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.

D.1.14 SQLExecute

`SQLExecute` executes the prepared SQL using `SQLPrepare`.

Syntax

```
RETCODE SQLExecute( hStmt )
```

Arguments

The arguments for `SQLExecute` are listed in [Table D-16](#):

Table D-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.

D.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 D-17](#):

Table D–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.

D.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 D–18](#):

Table D–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.

D.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 D–19](#):

Table D–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.
SQLINTEGER*	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.

D.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 D–20](#):

Table D–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 D–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.

D.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 D–21](#):

Table D–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.

D.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 D-22](#):

Table D-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.

D.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 D-23](#):

Table D-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.

D.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 D-24](#):

Table D-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 database server is the third tier of the Web-to-go three-tier 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.

Java Servlet Development Kit

The Java Servlet Development Kit is a tool provided by JavaSoft for developing Java servlets.

Java Web Server Development Kit

The Java Web Server Development Kit 1.0.1 is a JavaSoft tool for developing both JavaServer Pages (JSP) and Java servlets.

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 three-tier Web model. It contains the Mobile Server and Oracle Database Lite. Web-to-go replicates the user's applications and data to Oracle Database Lite when the user switches to offline mode. When the user switches back to online mode, 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 database server. The Mobile Server can be configured to run with the Oracle HTTP Server, the Apache server, and the 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. When the Mobile Client for Web-to-go is in online mode, it stores applications and data on the Oracle database.

Oracle Database Lite

Oracle Database Lite is the database component of the Mobile Client for Web-to-go. When the client is in offline mode, it stores applications and data on Oracle Database Lite.

Offline Mode

Offline mode is the condition of the Mobile Client for Web-to-go when it is disconnected from the Mobile Server. In offline mode, the client applications are executed locally and data is accessed and stored in Oracle Database Lite. See also ["Online Mode"](#).

Online Mode

Online mode is the condition of the Mobile Client for Web-to-go when it is connected to the Mobile Server. See also "[Offline Mode](#)".

Packaging Wizard

The Packaging Wizard enables administrators to publish Web-to-go applications to the Mobile Server repository. Administrators can use the Packaging Wizard to create a new Web-to-go 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.

Registry

The registry contains unique Web-to-go name/value pairs. All registry names must be unique.

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. Web-to-go avoids replication conflicts by using sequence values for disconnected clients.

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. See also "[Window Sequence](#)".

Sites

Web-to-go creates a database for each user on the Mobile Client for Web-to-go. This database is called a site. A client can contain multiple sites, but only one site per user. Users can have multiple sites on different clients.

Snapshots

Snapshots are copies of application data that Web-to-go captures in real-time 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 goes offline, Web-to-go automatically creates the snapshots on the client machine. Each subsequent time that a user goes online or offline, Web-to-go 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.

Switching Modes

Switching modes is the process the Mobile Client for Web-to-go uses to go offline or to go back online. When the client switches to offline mode, it downloads all of the applications and data required to work offline on Oracle Database Lite. When the client switches back to online mode synchronizes data changes on Oracle Database Lite with the Oracle database.

Synchronization

Synchronization is the process Web-to-go uses to replicate data between the Mobile Client for Web-to-go and the Oracle database. Web-to-go replicates the user's applications and data to Oracle Database Lite when the user switches to offline mode. When the user switches back to online mode, Web-to-go replicates any data changes 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 a 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.

Window Sequence

The window sequence is one of two sequences Web-to-go uses in order to provide unique primary key values to the Mobile Client for Web-to-go when it is in offline mode. The window sequence 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, Web-to-go recreates the sequence with a new, unique range of values.

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

A

- administration, 11-14, 12-27
 - defining snapshot values, 12-31
 - granting user access, 12-30
 - setting properties, 12-29
- ADO.NET, 2-6
- ADO.NET Provider
 - Classes, 6-1
 - Limitations, 6-6
 - Thread Safety, 6-6
 - Running the Demo, 6-5
- Advanced Functions for Customizing
 - Consolidator, 3-40
- AlterPublicationItem, 3-40
- application
 - administration, 11-14, 12-27
 - Web-to-Go, 12-33
- application development, 4-1
- Application Model and Architecture
 - Data Source Name, 1-9
 - Java Support, 1-9
 - Message Generator and Processor (MGP), 1-6
 - Mobile Development Kit, 1-7
 - Mobile Server, 1-5
 - Mobile Server Repository, 1-6
 - Mobile Sync, 1-4
 - Oracle Database Lite RDBMS, 1-4
 - Supported Platforms, 1-9
- applications
 - building Web applications, 12-1
 - packaging, 12-12
 - publishing, 12-22
 - uploading, 12-25

B

- BC4J, 13-2
 - access by JSP, 13-14
 - creating the database connection, 13-3
 - creating the BC4J component, 13-11
 - deploying, 13-19
 - deploying JSPs, 13-16
 - developing applications, 13-2
 - overview, 13-1
 - packaging JSPs, 13-16

- publishing, 13-18
 - testing, 13-18
 - Web-to-Go, 13-19
- BLOB, 2-18
- building applications
 - using BC4J, 13-2
 - Win32, 10-1, 10-2
 - Windows CE, 11-1
- building Mobile applications
 - before you start, 12-1
 - developing, 12-2
- building Mobile Web applications, 12-1

C

- Caching Publication Item Queries, 3-44
- Callback Customization for DML Operations, 3-52
- C/C++ Interface, 5-8
- Classes
 - OracleBlob and Large Object Support, 6-2
 - OracleCommand, 6-2
 - OracleConnection, 6-1
 - OracleParameter and PreparedStatements, 6-2
 - OracleSync and Data Synchronization, 6-3
 - Transaction Management, 6-1
- clients
 - subscribing to publications, 3-21
- COM Interface, 5-3
- Complete Refresh for Views, 3-42
- conflicts, 3-56
- Consolidator API, 3-8
- Creating a Dependency Hint, 3-41

D

- data source
 - creating name, 2-7
- database
 - accessing, 2-4
 - backing up, 2-11
 - building demo tables, 2-10
 - connecting, 2-8
 - creating, 2-7
 - encrypting and decrypting, 2-11
 - granting privileges, 2-10
 - granting roles, 2-10

- populate, 2-10
 - revoking roles, 2-10
 - starter, 2-6
- database connection
 - verification using msql, 2-4
- database objects
 - creating, 11-2
- datatypes
 - mapping, 3-58
 - Oracle Database Lite, 3-58
- decryption
 - database, 2-11
- Developing and Testing the Application, 4-1
- developing applications
 - execution, 12-10
- Developing Java Server Pages, 4-3
- Developing Mobile Applications for PALM OS
 - Devices
 - Building a SODA Application, 7-3
 - Building a SODA Forms Application, 7-3
 - Building an ODBC Application, 7-3
 - Customizing Oracle Database Lite Runtime, 7-4
 - Installing Oracle Database Lite Runtime on the Device, 7-1
 - Packaging your Application with Oracle Database Lite Runtime, 7-3
 - Palm Shared Library Manager (PSLM), 7-4
 - Running Oracle Database Lite on Palm OS Emulator, 7-2
 - Running Oracle Database Lite on Palm OS Simulator, 7-2
 - Uninstalling or Replacing Oracle Database Lite Runtime, 7-2
 - Using Oracle Database Lite Base Libraries, 7-2
- Developing Mobile Web Applications, 4-1
- development
 - compiling, 12-4
 - create database object, 12-3
 - registration, 12-5
- Development and Testing
 - Building Web-to-Go Applications, 4-2
 - Customizing the Workplace Application, 4-20
 - Debugging Web-to-Go Applications, 4-16
 - Developing Applet JDBC Communication, 4-13
 - Developing Applet Servlet Communication, 4-14
 - Developing Java servlets for Web-to-Go, 4-4
 - Specifying Application Roles, 4-3
 - Using the Mobile Server Admin API, 4-21
 - Using Web-to-Go Applets, 4-11
- Development Architecture, 4-1
- development interfaces, 2-2
 - for object database development, 2-2
 - for relational database development, 2-2
- JDBC, 2-2
- ODBC, 2-5
- SODA, 2-5
- doCompose Method, 3-28
- DSN
 - creating, 2-4

E

- Embedded Visual Tools, 11-1
- encryption
 - database, 2-11
- errors, 3-56

F

- fast refresh and update, 3-41
- Fast Refresh for Views, 3-42
- foreign key constraint, 3-50
- foreign key constraints, 3-49
 - violations, 3-50

H

- hints, 3-41

I

- INSTEAD OF Triggers, 3-41
- INSTEAD OF triggers, 3-41
- interfaces, 2-2
- isolation level
 - changing, 2-13

J

- JDBC driver, 2-2
 - description, 2-2
- JSP
 - access BC4J component, 13-14
 - deploying, 13-16

L

- linguistic sort, 2-14
- Load APIs, 2-6, B-1
- Load Utility (OLLOAD), 2-6

M

- Message Generator and Processor (MGP)
 - Applying Changes to the Server Database, 1-6
 - starting, 11-17
 - The Apply Phase, 1-6
 - The Compose Phase, 1-6
- Mobile client
 - synchronizing, 12-36
- Mobile Development Kit
 - Mobile SQL (MSQL), 1-8
 - Using the Packaging Wizard, 1-8
- Mobile Manager
 - application properties, 12-29
 - starting, 12-27
- Mobile Server
 - logon, 12-23
 - overview, 1-5
- msql
 - verifying database connection, 2-4

- MyCompose, 3-27
 - doCompose, 3-28
 - needCompse Method, 3-27

N

- Native Application Development
 - Data Source Name, 5-2
 - Java Support, 5-1
 - Supported Platforms, 5-1
 - Using the Packaging Wizard, 5-17
- needCompose Method, 3-27
- Null Sync Callout, 3-49

O

- OCBC Administrator, 2-7
- ODBC
 - development interfaces, 2-5
- ODBC driver, 2-5
 - description, 2-5
- ODBC support
 - Palm, D-1
- openConnection, 3-12
- Optimizing SQL Queries
 - Optimizing Single-Table Queries, A-1
- Oracle Database Lite
 - Application Model and Architecture, 1-3
 - Introduction, 1-1
- Oracle Database Lite Datatypes, 3-58

P

- packaging applications, 12-12, 12-21
 - application connection, 12-14
 - defining, 12-12
 - sequences, 12-19
 - snapshot, 12-15
 - SQL files, 12-21
- Packaging Wizard, 11-7
- packaging wizard, 12-12
- Palm Shared Library Manager (PSLM)
 - Building a Shared Library Project, 8-3
 - Building an Application Using PSLM, 8-6
 - Calling a PSLM Library from Your Application, 8-5
 - Cloaked Shared Libraries, 8-6
 - Exceptions Across Modules, 8-6
 - Overview, 8-1
 - Patching the CodeWarrior Runtime, 8-7
 - Trying out PSLM, 8-1
 - Writing a PSLM Library, 8-2
- parent tables
 - hints, 3-41
 - INSTEAD OF triggers, 3-41
 - updatable, 3-41
- password, 2-11
- PL/SQL, 3-45
- Pocket PC
 - installing, 11-18
 - synchronizing, 11-19

- primary key index, 3-23
- Programming interfaces
 - C / C++, 5-8
 - COM, 5-3
- properties
 - setting, 12-29
- publication, 3-8
- publication item, 3-8
- publication items
 - adding to publications, 3-18
- publications
 - subscribing clients to, 3-21
- publish applications, 12-22
 - create database account, 12-22
- publishing application
 - create database objects, 12-22
- publishing applications
 - logon to the Mobile Server, 12-23
 - start the Mobile Server, 12-23
 - uploading, 12-25
- publish/subscribe model, 3-8
- purging transactions, 3-58

Q

- query optimizer, A-2
- Queue Interface, 3-45

R

- RDBMS
 - changing passwords, 2-9
 - creating multiple users, 2-8
 - development interfaces, 2-1, 2-2
 - dropping users, 2-9
 - introduction, 2-1
 - linguistic sort, 2-14
- Refresh
 - complete, 3-42
 - fast, 3-42
- Restricting Predicate, 3-54

S

- Servlets
 - registering, 12-5
- setPassword, 3-23
- snapshot, 3-8
- snapshot definitions
 - creating, 2-16
 - creating publications, 2-17
 - declarative, 2-16
 - programmatic, 2-17
- snapshots
 - defining, 11-10
- SQLAllocConnect, D-2
- SQLAllocEnv, D-3
- SQLAllocHandle, D-3
- SQLAllocStmnt, D-4
- SQLBindCol, D-11
- SQLBindParameter, D-8

- SQLConnect, D-7
- SQLDescribeCol, D-11
- SQLDisconnect, D-8
- SQLException, D-12
- SQLExecDirect, D-9
- SQLExecute, D-10
- SQLFetch, D-10
- SQLFreeConnect, D-5
- SQLFreeEnv, D-5
- SQLFreeHandle, D-6
- SQLFreeStmt, D-6
- SQLGetData, D-13
- SQLNumResultCols, D-13
- SQLPrepare, D-9
- SQLRowCount, D-14
- SQLTransact, D-14
- subscription, 3-8
- subscription parameters, 3-8
 - defining, 3-16
- Synchronization
 - Add Map Table Partitions, 3-38
 - Adding Publication Items to the Publication, 3-18
 - Advanced Features for Customizing
 - Consolidator, 3-26
 - Change Password, 3-23
 - Client Device Database DDL Operations, 3-23
 - Complete Refresh Synchronization, 3-7
 - Compose Phase Customization, 3-27
 - Create a Map Table Partition, 3-38
 - CreateSubscription, 3-21
 - Creating Publication Item Indexes, 3-17
 - Creating Publication Items, 3-14
 - Creating Users, 3-20
 - Defining Client Subscription Parameters, 3-16
 - Defining Conflict Rules, 3-19
 - Defining Publication Items, 3-15
 - DownloadInfo Class Access Methods, 3-34
 - Drop a Map Table Partition, 3-39
 - Drop all Map Table Partitions, 3-39
 - Drop User, 3-21
 - Extending MyCompose, 3-30
 - Extending MyCompose as a User Defined
 - Sub-Class, 3-27, 3-32
 - Fast Refresh Synchronization, 3-5
 - getDownloadInfo Method, 3-34
 - InstantiateSubscription, 3-22
 - Instantiating the Subscription, 3-22
 - Map Table Partition APIs, 3-37
 - Merge Map Table Partitions, 3-39
 - Mobile Sync APIs, 3-7
 - Overview, 3-1
 - Process, 3-5
 - PublicationSize Class, 3-35
 - Publish and Subscribe Model, 3-8, 3-9
 - Publishing Synonyms, 3-24
 - Remote Database Link Support, 3-23, 3-26
 - Sequence Support, 3-15
 - SetSubscription Parameters, 3-16
 - Subscribing Users, 3-21
 - Sync Discovery API, 3-33

- Synchronizing an Encrypted Database, 3-7
- synchronization
 - conflicts, 3-56
 - errors, 3-56
- synchronizing
 - Mobile client, 12-36
 - Pocket PC, 11-19

T

- tables
 - building, 2-10
- tracing, 2-21
- transactions, 2-11
 - changing the default isolation level, 2-13
 - executing, 3-57
 - locking, 2-13
 - purging, 3-58
 - tuning, 2-14

U

- user
 - granting access, 12-30
- users
 - defining snapshot values, 12-31
- Using Mobile Sync for Palm
 - Configuring mSync, 9-1
 - Using HotSync to Synchronize Data with the
 - Mobile Server, 9-2
 - Configuring HotSync for a PalmOS
 - Device, 9-3
 - Configuring PalmOS Emulator for
 - HotSync, 9-3
 - HotSync Timeout Errors, 9-3
- Using Network Sync, 9-3
 - Network Sync With PalmOS Emulator, 9-4
 - Synchronizing Using a Cradle and Windows
 - Desktop, 9-4

V

- versioning, 3-57
- views
 - fast refresh and update, 3-41
- Virtual Primary Key, 3-43

W

- Web-to-Go
 - install the client, 12-33
 - introduction, C-1
 - sample applications, C-1
- Web-to-go
 - client installation, 12-33
- Win32
 - command sequence, 10-2
- Windows CE
 - compilation, 11-6
 - creating database object, 11-2
 - developing applications, 11-2

- packaging, 11-7
- publishing, 11-7
 - using the Packaging Wizard, 11-7
 - writing application code, 11-4
- winning rules, 3-57

