

Oracle Fusion Middleware 11g: Build Applications with Oracle Forms

Volume I • Student Guide

D61530GC10

Edition 1.0

December 2009

DXXXX

ORACLE®

Author	Copyright © 2009, Oracle. All rights reserved.
Pam Gamer	Disclaimer
Technical Contributors and Reviewers	This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.
Glenn Maslen Duncan Mills Grant Ronald	The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.
Editors	Restricted Rights Notice
Raj Kumar Amitha Narayan	If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:
Publishers	U.S. GOVERNMENT RIGHTS The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.
Jayanthy Keshavamurthy Veena Narasimhan Giri Venugopal	Trademark Notice Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

I

Introduction

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Identify the course objectives
- Identify the course content and structure



Lesson Aim

This lesson introduces you to the *Oracle Fusion Middleware 11g: Build Internet Applications with Oracle Forms* course:

- The objectives that the course intends to meet
- The topics that it covers
- How the topics are structured over the duration of the course

Course Objectives

After completing this course, you should be able to do the following:

- Create Forms modules including components for database interaction and GUI controls
- Display Forms modules in multiple windows and a variety of layout styles
- Test Forms modules in a Web browser
- Debug Forms modules in a three-tier environment



Course Objectives

In this course, you learn to build, test, and deploy interactive Internet applications. Working in a graphical user interface (GUI) environment, you learn how to create and customize forms with user input items such as check boxes, list items, and radio groups. You also learn how to modify data access by creating event-related triggers, and you display Forms elements and data in multiple canvases and windows.

Course Objectives

- Implement triggers to:
 - Enhance functionality
 - Communicate with users
 - Supplement validation
 - Control navigation
 - Modify default transaction processing
 - Control user interaction
- Reuse objects and code
- Link one Forms module to another

ORACLE®

Course Content

Day 1

- Lesson 1: Introduction to Oracle Forms Builder and Oracle Forms Services
- Lesson 2: Running a Forms Application
- Lesson 3: Working in the Forms Environment
- Lesson 4: Creating a Basic Forms module
- Lesson 5: Creating a Master-Detail Form
- Lesson 6: Working with Data Blocks and Frames

ORACLE®

I - 5

Copyright © 2009, Oracle. All rights reserved.

Course Content

The lesson titles show the topics that are covered in this course, and the usual sequence of lessons. However, the daily schedule is an estimate, and may vary for each class.

Course Content

Day 2

- Lesson 7: Working with Text Items
- Lesson 8: Creating LOVs and Editors
- Lesson 9: Creating Additional Input Items
- Lesson 10: Creating Noninput Items

ORACLE®

Course Content

Day 3

- Lesson 11: Creating Windows and Content Canvases
- Lesson 12: Working with Other Canvas Types
- Lesson 13: Introduction to Triggers
- Lesson 14: Producing Triggers
- Lesson 15: Debugging Triggers

ORACLE®

I - 7

Copyright © 2009, Oracle. All rights reserved.

Course Content

Day 4

- Lesson 16: Adding Functionality to Items
- Lesson 17: Run-Time Messages and Alerts
- Lesson 18: Query Triggers
- Lesson 19: Validation
- Lesson 20: Navigation

ORACLE®

Course Content

Day 5

- Lesson 21: Transaction Processing
- Lesson 22: Writing Flexible Code
- Lesson 23: Sharing Objects and Code
- Lesson 24: Using WebUtil to Interact with the Client
- Lesson 25: Introducing Multiple Form Applications

ORACLE®

Summary

In this lesson, you should have learned to:

- Identify the course objectives
- Identify the course content and structure



Summary

In this lesson you learned about the objectives of the *Oracle Fusion Middleware 11g: Build Applications with Oracle Forms*. You learned the topics that it covers and how those topics are structured over the duration of the course.

Introduction to Oracle Forms Builder and Oracle Forms Services

1

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to describe the following:

- The components of Oracle Fusion Middleware 11g
- The features and benefits of Oracle Forms Services and Oracle Forms Builder
- The architecture of Oracle Forms Services
- The course application



Lesson Aim

This course teaches you how to build effective and professional applications by using Oracle Forms Builder.

This lesson identifies the key features of Oracle Fusion Middleware 11g, Oracle Forms Services, Oracle Forms Builder, and the course application model and contents.

There are many terms used in this course that may be unfamiliar to you. For a glossary containing definitions of many of these terms, see <http://www.oracle.com/glossary>.

Web Computing Solutions

Application Type and Audience	Product Approach	Oracle Products
<i>Enterprise applications, business developers</i>	Declarative	<i>Oracle JDeveloper and ADF, Oracle Forms</i>
Java components, component developers	Two-way coding, Java and JavaBeans	Oracle JDeveloper
Self-service applications & content management, Web site developers	Browser-based, dynamic HTML	Oracle WebCenter
Reporting and analytical applications, MIS and business users	Dynamic Web reporting, Drill, analyzing, forecasting	Oracle Reports and Oracle Discoverer

ORACLE®

1 - 3

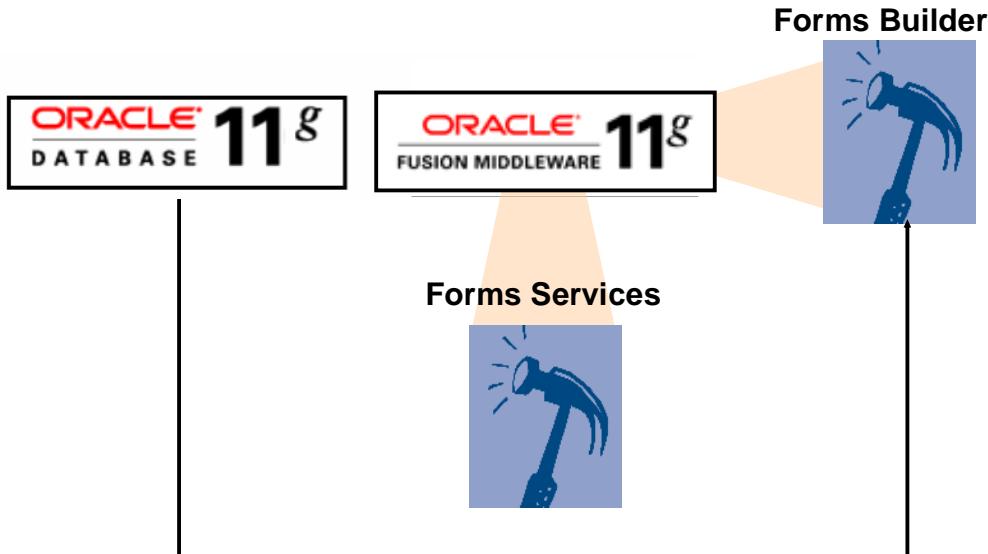
Copyright © 2009, Oracle. All rights reserved.

Web Computing Solutions

Oracle offers a range of tools and deployment options for Web computing, some of which are shown in the slide and described briefly below. Different types of developers and applications require different toolsets.

- Enterprise application developers need a declarative approach. Oracle JDeveloper's Application Development Framework and Oracle Forms both can provide this solution. This course focuses on how you can use Oracle Forms Builder to rapidly build scalable, high-performance applications for the Web and then deploy the applications with Oracle Forms Services.
- Component developers need different tools and methods. For these developers, Java is the language of choice. Oracle's solution is JDeveloper.
- For Web site developers and content publishers who want to build self-service applications for Web sites, Oracle WebCenter supports the creation of all types of portals, Web sites, and composite applications, and is designed to enable business users to evolve these applications as their business requirements change.
- For management information systems (MIS) developers and end users, there is the Oracle Business Intelligence toolset. Oracle Reports Developer, Oracle Reports Services, and Oracle Discoverer provide the whole range for reporting and analysis facilities.

Oracle 11g Products and Forms Development



Oracle 11g Products and Forms Development

The graphic in the slide depicts the three major components used in developing and deploying Forms applications.

Oracle Database: Manages all your information, such as Word documents, Excel spreadsheets, Extensible Markup Language (XML), and images. Oracle Forms Builder is designed for the Oracle database. It delivers the following services for you natively—services you would otherwise have to code by hand:

- Connects to and maintains a connection to the Oracle database
- Queries and handles a large number of records on demand
- Locks database records on demand
- Generates code that automatically supports multi-user locking scenarios
- Manages inserts, updates, and deletes automatically
- Allows programmatic manipulation of sets of records for a developer
- Communicates transactions efficiently to the database in an atomic fashion, meaning that a transaction is either fully committed or not at all
- Automatically handles communication with database Advanced Queuing queues
- Automatically handles logins when database proxy users are used for deployment

Oracle 11g Products and Forms Development (continued)

The other two components in the slide are part of Oracle Fusion Middleware:

- **Oracle Forms Services:** A comprehensive application framework optimized to deploy Forms applications in a multitier environment. Oracle Forms Services is integrated with Oracle WebLogic Server for running Forms applications.
- **Oracle Forms Builder:** Leverages the infrastructure offered by Oracle Fusion Middleware and Oracle Database, enabling developers to quickly and easily build scalable, secure, and reliable e-business applications. Oracle Forms Builder is included in Oracle Developer Suite, which provides a complete and highly productive development environment for building applications.

Oracle Forms Builder and Oracle Forms Services provide a complete application framework for optimal deployment of Oracle Forms applications on the Web as well as corporate networks. This application framework infrastructure is provided for you, yet you still have the flexibility to leverage the latest technologies within your applications. This allows you to focus on the real value and spend your time thinking about the application business logic and functionality rather than worrying about the application infrastructure.

Oracle Fusion Middleware 11g Architecture



Oracle Fusion Middleware 11g Architecture

Oracle Fusion Middleware is a standards-based family of products that are often deployed and used in conjunction with one another to develop Java EE applications, providing the benefits of common security, management, deployment architecture, and development tools.

The slide shows an overview of Oracle Fusion Middleware, displaying some of its many features and components, including the following broad categories:

- **Development Tools:** Integrated set of tools for developing SOA applications; includes tools such as JDeveloper and Application Development Framework (ADF), WebCenter Suite, SQL Developer, Forms, Designer, and Reports
- **User Interaction and Enterprise 2.0:** A dynamic, personalized UI layer using Web 2.0 technologies; includes products such as Oracle WebCenter Services and Oracle WebLogic Portal
- **Enterprise Performance Management:** A modular suite of applications supporting strategic and financial management processes; integrates with Oracle Business Intelligence to integrate data from multiple sources and provide dashboards, reporting, and analysis
- **Business Intelligence:** A portfolio of technologies and applications providing an integrated array of query, reporting, analysis, alerting, analytics, desktop integration, and data warehousing tools
- **Content Management:** Robust, scalable solutions for managing all types of content

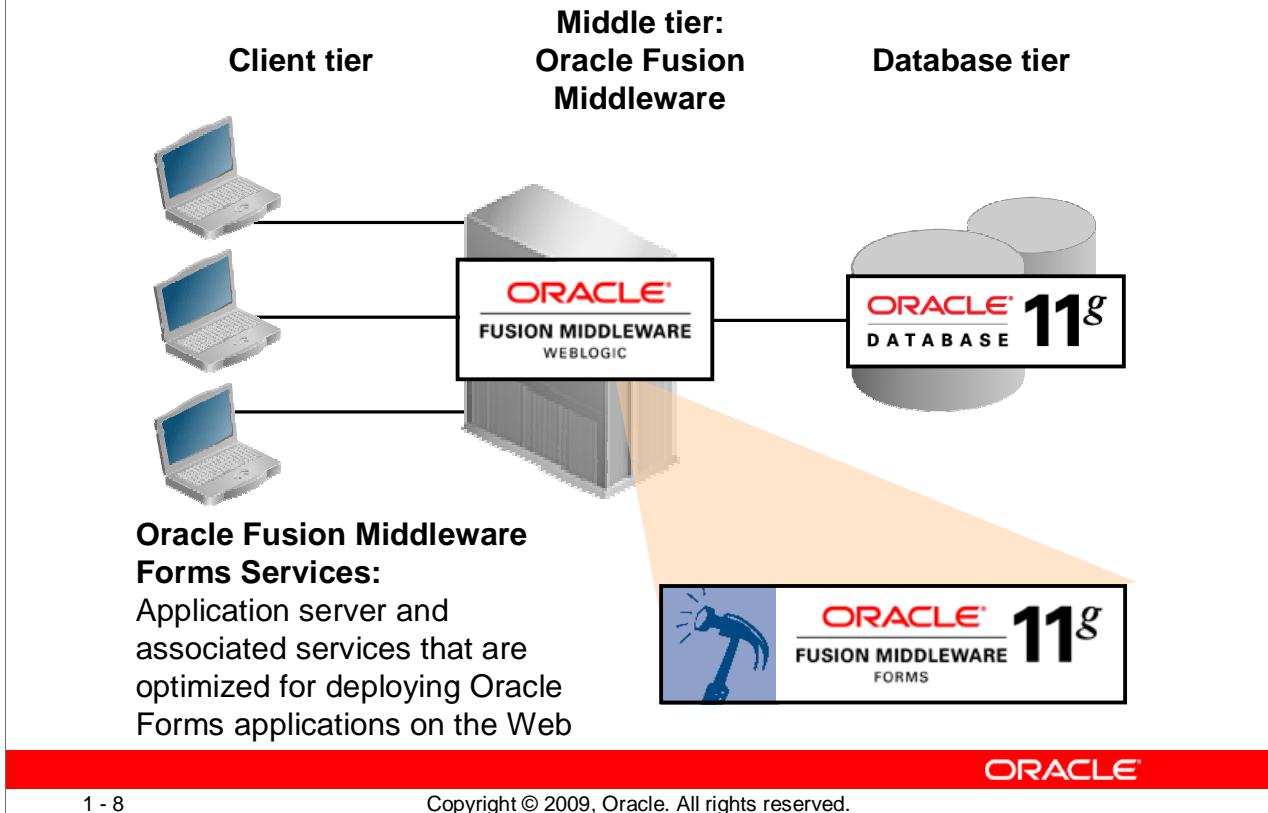
Oracle Fusion Middleware 11g Architecture (continued)

- **SOA and Process Management:** An architecture and tools enabling development of enterprise applications as modular business services that can be easily integrated and reused
- **Application Server:** Oracle WebLogic Server, a complete Java EE 5 and Java SE 6 implementation enabling development and deployment of enterprise applications and services
- **Grid Infrastructure:** An architecture where pools of resources are established, shared, and configured to support varying demands of an enterprise; includes grid control, clustering, and storage management
- **Enterprise Management:** A tool that enables top-down management of applications, middleware, and databases
- **Identity Management:** A set of services providing management of user identities across all enterprise resources, both within and outside the firewall

Oracle Fusion Middleware includes the following major products:

- **Oracle WebLogic Server:** Discussed in the subsequent slides
- **Oracle Metadata Repository:** Contains metadata for system components as well as for configuration of middleware and applications
- **Oracle Identity and Access Management:** An enterprise identity management system
- **Oracle Fusion Middleware Application Components:** Includes Oracle HTTP Server, Oracle Web Cache, Oracle Portal, Oracle Web Services, Oracle Forms Services, and developer tools such as Oracle JDeveloper and Oracle Forms Builder
- **Oracle SOA Suite:** A set of service infrastructure components for creating, managing, and orchestrating services into composite applications and business processes
- **Oracle WebCenter Framework:** An integrated set of components with which you can create social applications, enterprise portals, collaborative communities, and composite applications, built on a standards-based, service-oriented architecture
- **Oracle Business Intelligence:** An integrated solution that addresses all business intelligence requirements

Oracle Forms Services: Overview



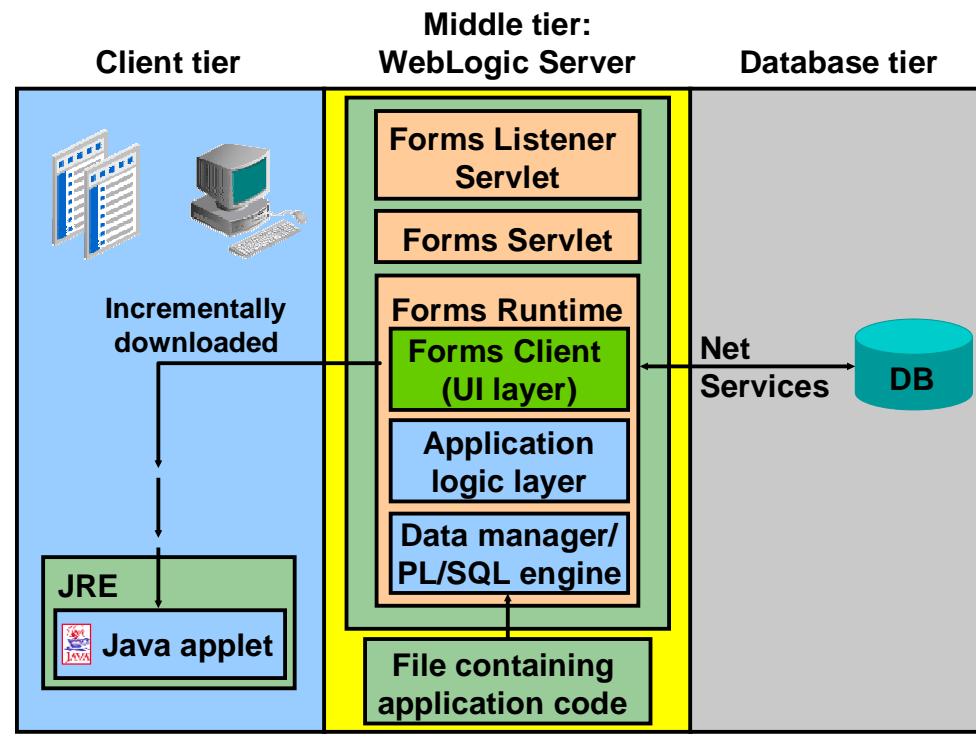
Oracle Forms Services: Overview

Oracle Forms Services is a component of Oracle Fusion Middleware for delivering Oracle Forms applications to the Web. Oracle Forms Services automatically provides the infrastructure that is needed to successfully deliver applications on the Web through built-in services and optimizations.

Oracle Forms Services uses a three-tier architecture to deploy database applications:

- The client tier contains the Web browser, where the application is displayed and used.
- The middle tier is the application server, where the application logic and server software reside.
- The database tier is the database server, where enterprise data is stored.

Forms Services Architecture



1 - 9

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Forms Services Architecture

Forms Services consists of four major components: the Java client (Forms Client), the Forms Listener Servlet, the Forms Servlet, and the Forms Runtime Engine.

When a user runs a Forms session over the Web, a thin, Java-based Forms applet is dynamically downloaded from the application server and automatically cached on the client computer. The same Java applet code can be used for any form, regardless of size and complexity.

Although Forms Services uses a Java applet for displaying the form on the client browser, the developer does not need to know Java in order to develop and deploy a Forms application.

You learn more about the Forms Services components and the process of starting a Forms session in the lesson titled “Running a Forms Application.”

Oracle Forms Builder: Overview

Oracle Forms Builder:

- Is a productive development environment for Web-deployed business applications
- Provides for:
 - Data entry
 - Queries



ORACLE®

1 - 10

Copyright © 2009, Oracle. All rights reserved.

What Is Oracle Forms Builder?

Oracle Forms Builder is a productive development environment for building enterprise-class, scalable database applications for the Web. Oracle Forms Builder provides a set of tools that enable business developers to easily and quickly construct sophisticated database forms and business logic with a minimum effort.

Oracle Forms Builder uses powerful declarative capabilities to rapidly create applications from database definitions that leverage the tight integration with the Oracle database. The toolset leverages Java technology, promotes reuse, and is designed to allow developers to declaratively build rich user interfaces. Developer productivity is further increased through a single integrated development environment that enables distributed debugging across all tiers, utilizing the same PL/SQL language for both server and client.

Oracle Forms 11g: Key Features

- Tools for rapid application development
- Use of PL/SQL scripting language
- Application partitioning
- Extended scalability
- Object reuse
- Integration

ORACLE®

1 - 11

Copyright © 2009, Oracle. All rights reserved.

Oracle Forms 11g: Key Features

Tools for rapid application development: You can create and modify applications with little or no code using wizard-based rapid application development and built-in commands.

Use of PL/SQL scripting language: You can write and debug PL/SQL code from within Forms Builder.

Application partitioning: You can place individual PL/SQL program units on the database server or in the application, whichever is most suitable. You can drag objects between modules and the database server.

Extended scalability: The multitiered architecture enables you to scale applications from a single user to tens of thousands of users, with no changes to the application. You can use server functionality, such as array data manipulation language (DML), database cursors, or bind variables, to improve scalability.

Object reuse: Oracle Forms Builder offers an inheritance model that facilitates the inheritance of attributes and code from one object to another and from one application to another, through subclassing and object libraries.

Integration: You can integrate existing Oracle Forms applications to take advantage of Web technologies and Service-Oriented Architecture (SOA).

Oracle Forms 11g: New Features

- For development:
 - Integration with external events
 - JavaScript integration
 - New built-ins to support database proxy users
 - Events for Pluggable Java Components
- For configuration and deployment:
 - Enterprise Manager integration improvements
 - Tracing improvements
 - Support for Oracle Diagnostic Logging
 - Using Java Controller with Reports called from Forms
 - Use of database proxy users

ORACLE®

1 - 12

Copyright © 2009, Oracle. All rights reserved.

Oracle Forms 11g: New Features

Oracle Forms 11g includes several new features for both development and deployment:

- **Development features:**
 - Integration with external events by using Oracle database Advanced Queuing
 - Integration with JavaScript, giving the ability to call into Forms from JavaScript and also to invoke JavaScript code from Forms
 - New built-ins for support of using database proxy users
 - Ability to dispatch events from Pluggable Java Components
- **Configuration and deployment features:**
 - New Enterprise Manager user interface and functionality
 - Tracing improvements that enable logging of called PL/SQL functions and procedures with their parameter names, types, and values
 - Support for Oracle Diagnostic Logging (ODL), which is Oracle's standardized logging architecture
 - Ability to use the Java Virtual Machine (JVM) controller when integrating with Oracle Reports
 - Ability to use database proxy users with very few limited privileges for applications with high security requirement or many users

Although these features are not covered in this basic Forms course, you may obtain more information from the sources mentioned in the next slide.

Obtaining More Information

You can obtain further information from the following sources:

- Oracle Fusion Middleware Administrator's Guide 11g:
http://download.oracle.com/docs/cd/E12839_01/core.1111/e10105/toc.htm
- Oracle Fusion Middleware Forms Services Deployment Guide 11g:
http://download.oracle.com/docs/cd/E12839_01/web.1111/e10240/toc.htm
- Forms page on OTN:
<http://www.oracle.com/technology/products/forms>
- Forms forum:
<http://forums.oracle.com/forums/forum.jspa?forumID=82>
- Forms online help

ORACLE®

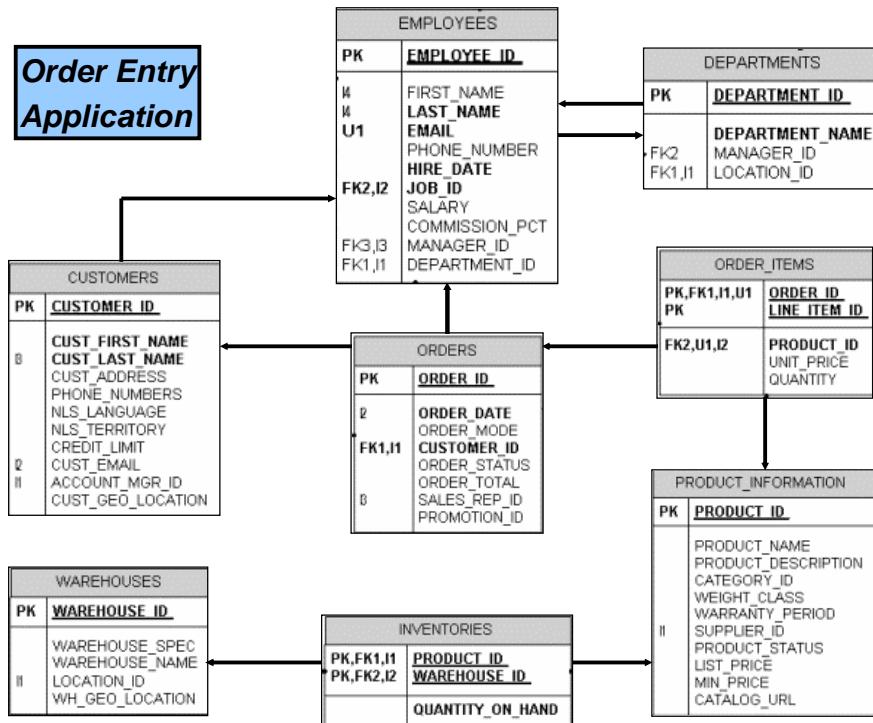
1 - 13

Copyright © 2009, Oracle. All rights reserved.

Obtaining More Information

This course focuses on development. You can learn more about administering Forms Services from the sources mentioned in the slide.

Summit Office Supply Schema



ORACLE®

Introducing the Course Application

Summit Office Supply Schema

The simplified table diagram shows the tables that are used throughout the course to build the Forms application. These tables are used in other Oracle University courses as well.

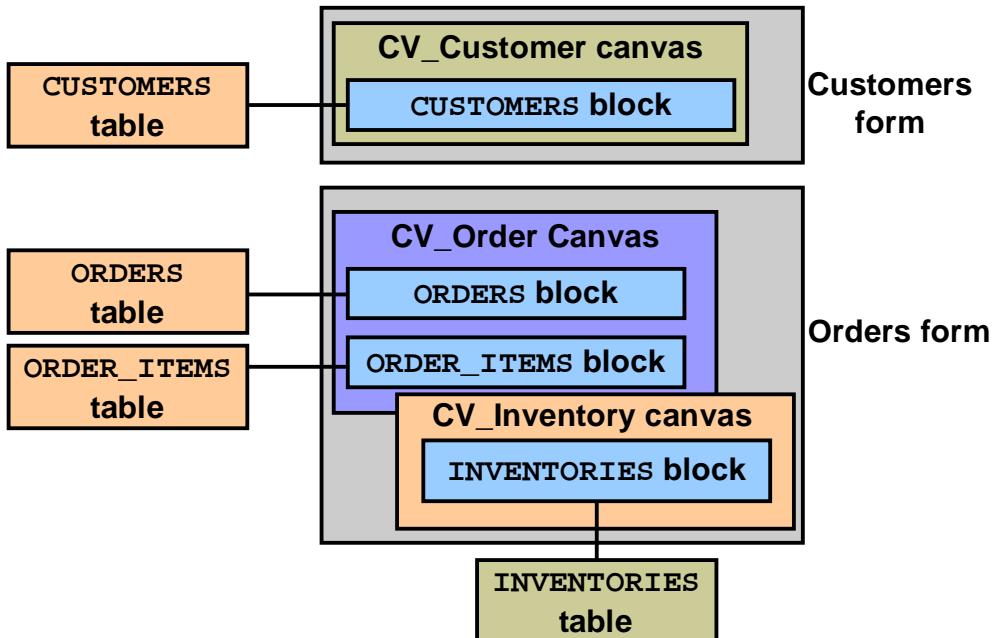
Summit Office Supply is a company that sells office products to customers. Summit has a number of employees in several departments. Some employees are sales representatives who have a relationship with specific customers.

Customers place orders. Each order consists of one or more line items. Each line item represents a product.

Many products have an associated image, in the form of an image file.

The company products are stored in a number of warehouses. The contents of the warehouses are managed in the inventory.

Summit Office Supply Application



Summit Office Supply Application

The following example of a Forms Builder application will familiarize you with the main run-time facilities of the product. You will also build your own version of this application during the practices in this course.

The Summit company produces a range of office supplies that they sell to businesses and individuals (their customers). The Summit application is an order-entry system that maintains customer details, their orders, and the available stock (inventory).

The application consists of two main forms:

- **Customers form:** The Customers form facilitates queries on existing customers and the insertion, update, or deletion of customer records. When a customer is selected, the user can open the Orders form to enter or view orders for that customer. The form consists of a single block, the CUSTOMERS block, which is a single record block whose base table is CUSTOMERS.

Summit Office Supply Application (continued)

- **Orders form:** Opened from the Customers form, the Orders form displays orders for a customer and the line items that belong to each order. Orders may also be created, modified, or deleted in this form. You can also display the stock available on the ordered products. The form consists of three blocks:
 - ORDERS block: The ORDERS block is a single-record master block for the form. The base table is ORDERS, but the block also displays associated information from other tables, such as the name of the customer.
 - ORDER_ITEMS block: The block is the related detail block for an order, showing its line items and the products ordered. This is a multirecord block whose items are on the same canvas as those in the ORDERS block. The base table of the Order_Items block is ORDER_ITEMS, but the block displays information from other tables, such as the product description.
 - INVENTORIES block: The INVENTORIES block is a multirecord block showing warehouse stock for a product. Its base table is the INVENTORIES table. Its items are on a separate canvas, which is assigned to its own window. This block is linked to the current product in the ORDER_ITEMS block, but the two blocks can operate independently.

Summary

In this lesson, you should have learned that:

- Oracle Fusion Middleware 11g provides services for building and deploying Web applications
- Oracle Forms Services provides:
 - Optimized Web deployment of Forms applications
 - Rich Java UI without Java coding
 - Generic Java applet to deploy any Forms application
- Oracle Forms Services consists of:
 - Forms Client
 - Forms Servlet
 - Forms Listener Servlet
 - Forms Runtime Engine



Summary

Oracle Fusion Middleware provides a variety of services for building and deploying Web applications, including Oracle HTTP Server (OHS), Reports Services, and Forms Services.

Oracle Forms Services, a component of Oracle Fusion Middleware 11g, provides for the Web deployment of Forms applications with a rich Java user interface. It uses the same generic applet for any form.

The components of Oracle Forms Services all play a role in running an application. These components are the Forms Client (Java applet), the Forms Servlet, the Forms Listener Servlet, and the Forms Runtime Engine.

Summary

- Benefits of Oracle Forms Builder include rapid application development, application partitioning, flexible source control, extended scalability, and object reuse
- The course application is a customer and order entry application for Summit Office Supply



Summary (continued)

Oracle Forms Builder enables you to develop Forms applications. Benefits of Oracle Forms Builder include:

- **Rapid application development:** Creating and modifying applications with little or no code
- **Application partitioning:** Dragging objects between modules and the database server
- **Flexible source control:** Integration with Software Configuration Manager (SCM)
- **Extended scalability:** Use of server functionality such as array DML, database cursors, or bind variables
- **Object reuse:** Subclassing, object libraries

Running a Forms Application



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Start WebLogic Server
- Describe the run-time environment
- Describe the elements in a running form
- Navigate a Forms application
- Describe the two main modes of operation
- Run a form in a Web browser:
 - Retrieve both restricted and unrestricted data
 - Insert, update, and delete records
 - Display database errors



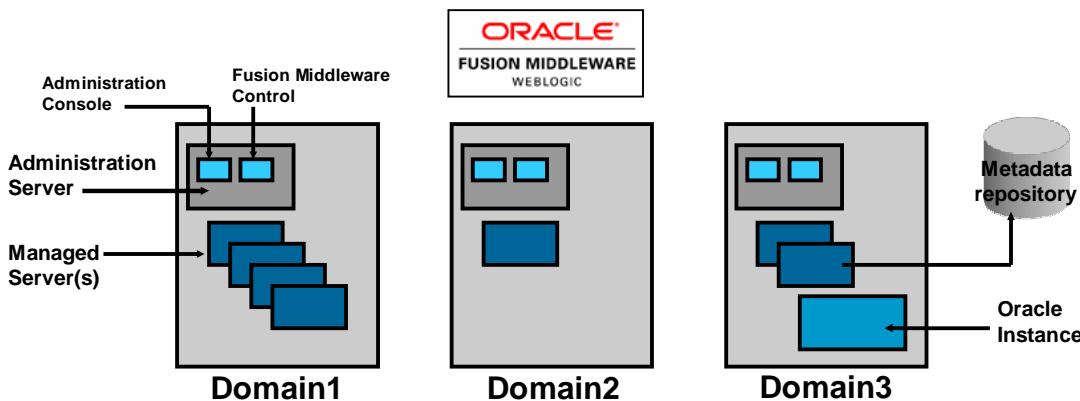
Lesson Aim

It is helpful to understand the environment of the form operator before designing and building your own applications. In this lesson, you run an existing application on the Web in order to familiarize yourself with the run-time interface of Oracle Forms Builder.

What Is Oracle WebLogic Server?

Oracle WebLogic Server:

- Is Oracle's application server for running and administering Forms (and other) applications
- Defines manageable environments called domains
- Enables testing of Forms applications within Forms Builder



What Is Oracle WebLogic Server?

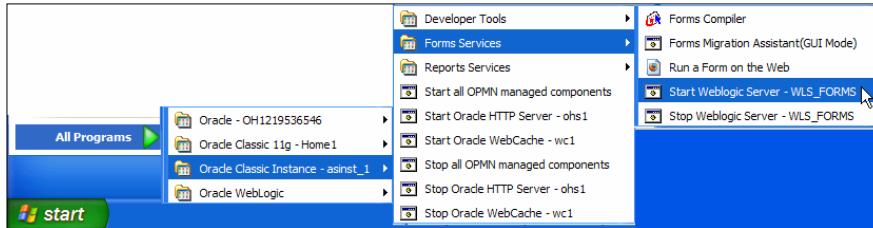
Oracle WebLogic Server is a scalable, enterprise-ready Java Platform, Enterprise Edition (Java EE) application server. The WebLogic Server infrastructure supports the deployment of many types of distributed applications and is an ideal foundation for building applications based on Service-Oriented Architecture (SOA). SOA is a design methodology aimed at maximizing the reuse of application services.

The graphics in the slide show that the WebLogic Server can define multiple environments called domains. Each domain consists of one Administration Server and one or more Managed Servers. Depending on the components that are installed, you may also have an Oracle Instance, and possibly a metadata repository if the installed components require one.

WebLogic Server provides a servlet container that is ideally suited to run Forms applications. The servlet container calls the servlet's methods and provides services that the servlet needs when running. When a request for a Forms application is routed to WebLogic Server, WebLogic Server performs the following actions:

1. If an instance of the servlet does not exist, it loads the servlet class, then instantiates and initializes an instance of the servlet class.
2. It invokes the servlet, passing request and response objects. The request object contains information about the client, request parameters, and HTTP headers. The response object returns the servlet's output to the client.

Starting and Stopping Oracle WebLogic Managed Servers



- Run `startManagedWebLogic.cmd` or `stopManagedWebLogic.cmd` (can run from the Start menu).
- You can minimize, but do not close, the command window.

```
Start Weblogic Server - WLS_FORMS
May 26, 2009 1:19:16 PM MDT <Notice> <Server> <BEA-002613> <Channel "Default" is now listening on 192.168.1.100:7001
May 26, 2009 1:19:16 PM MDT <Notice> <WebLogicServer> <BEA-000358> <Started WebLogic Independent Manager on port 7001
May 26, 2009 1:19:16 PM MDT <Warning> <JMX> <BEA-149510> <Unable to establish JMX Connectivity with the JMX Service URL: null>
May 26, 2009 1:19:16 PM MDT <Warning> <Server> <BEA-002611> <Hostname "pgamer-us.us.oracle.com". maps port 239 to 192.168.1.100
May 26, 2009 1:19:16 PM MDT <Notice> <WebLogicServer> <BEA-000365> <Server state changed to RUNNING>
May 26, 2009 1:19:16 PM MDT <Notice> <WebLogicServer> <BEA-000360> <Server started in RUNNING mode>
```

ORACLE

Starting and Stopping Oracle WebLogic Managed Servers

A managed server is a WebLogic Server instance that runs deployed applications. When you install and configure Oracle Forms, a managed server is created named `WLS_FORMS`. In order to run a form, you first must start this managed server. You can start a managed server even if the Administration Server (needed for configuring Forms Services) is not running.

To start or stop the `WLS_FORMS` managed server on Windows, execute the appropriate batch file, either `startManagedWebLogic.cmd` or `stopManagedWebLogic.cmd`, located in the `user_projects\domains\ClassicDomain\bin` subdirectory of the Fusion Middleware home directory. You must pass `SERVER_NAME` and `ADMIN_URL` command-line arguments to these scripts; for example:

```
startManagedWebLogic.cmd WLS_FORMS t3://myhost.com:7001
```

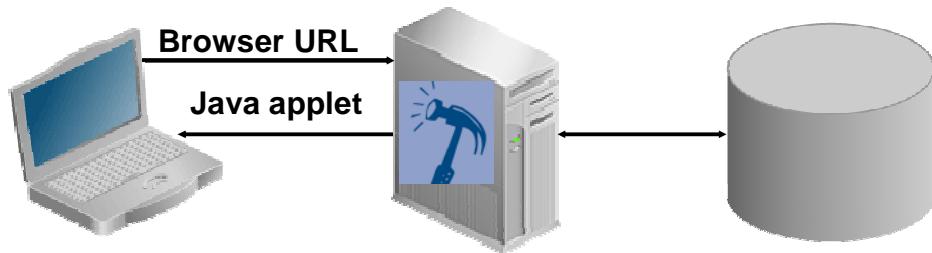
Note: The batch files on Linux systems end with the `.sh` extension rather than `.cmd`.

Alternatively, you can call these batch files from the Windows Start menu as shown in the top screenshot in the slide: All Programs > Oracle Classic Instance > Forms Services > Start [Stop] WebLogic Server – `WLS_FORMS`. These menu items already have the command-line arguments. You will need to input the username and password for starting the Oracle WebLogic Server, which were specified at the time of installation.

You can minimize, but do not close, the command window, as shown in the bottom screenshot.

Running a Form

Oracle Forms Services deployment:



ORACLE®

2 - 5

Copyright © 2009, Oracle. All rights reserved.

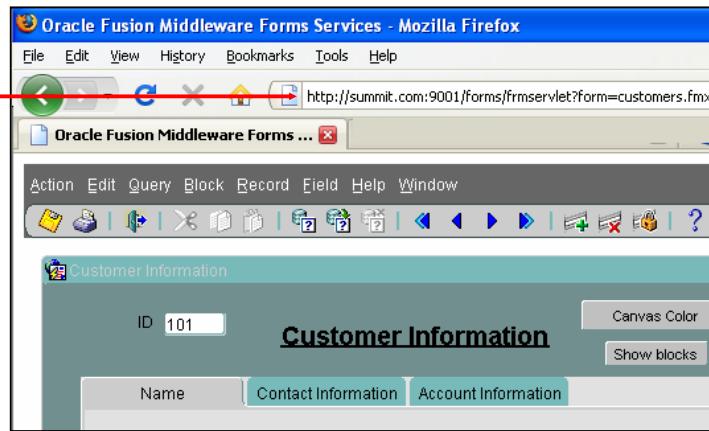
Running a Form

Deploying Forms applications to the Web is implemented by the three-tier architecture of Oracle Fusion Middleware. Application logic and the Forms Runtime Engine reside on the middle tier application server. All trigger processing occurs on database and application servers, whereas user interface processing occurs on the Forms Client. End users can run Forms applications in a Web browser.

Users request an application in their Web browsers by entering a URL that points to the application. Forms Services then generates an HTML file that downloads a Java applet to the client machine. This small applet is capable of displaying the user interface of any form, while the application logic is executed on the middle tier.

The graphics in the slide depict a client computer at the left and a middle tier computer in the middle, with Forms Services running. The middle tier computer communicates with the database shown at the right of the slide. The client sends an HTTP request in the form of a URL to the middle tier, which downloads a Java applet to the client.

Running a Form: Browser



**[http://summit.com:9001/forms/frm servlet
?form=<formname>.fmx
&userid=<username>/<password>@<database>
&buffer_records=NO&debug_messages=NO&array=YES
&query_only=NO](http://summit.com:9001/forms/frm servlet?form=<formname>.fmx&userid=<username>/<password>@<database>&buffer_records=NO&debug_messages=NO&array=YES&query_only=NO)**

ORACLE®

2 - 6

Copyright © 2009, Oracle. All rights reserved.

Running a Form: Browser

The graphic in the slide shows a user accessing the application. The URL to invoke an application must have the following format:

`http://<host>.<domain>:<port>/forms/frm servlet[?<parameter list-value pairs separated by "&"]. Optional portions of the URL are enclosed in brackets.`

Summit's URL consists of the following components:

Protocol	http
Host and domain	summit.com
Port for WebLogic Server or Oracle HTTP Server	9001 default for WebLogic Server (used for testing from Forms Builder) 8888 default for Oracle HTTP Server (more commonly used for production)
Forms Servlet Alias or static HTML file	/forms/frm servlet
Parameters: This section begins with "?"; parameters separated by "&" (can be specified in the URL or taken from configuration file).	form=customers.fmx userid=<username>/<password>@<database> buffer_records=NO debug_messages=NO

Java Runtime Environment

- The Forms applet runs in a Java Runtime Environment (JRE) on the client machine.
- Forms uses the JRE provided by the Sun Java Plug-in (JPI).



ORACLE®

2 - 7

Copyright © 2009, Oracle. All rights reserved.

Java Runtime Environment

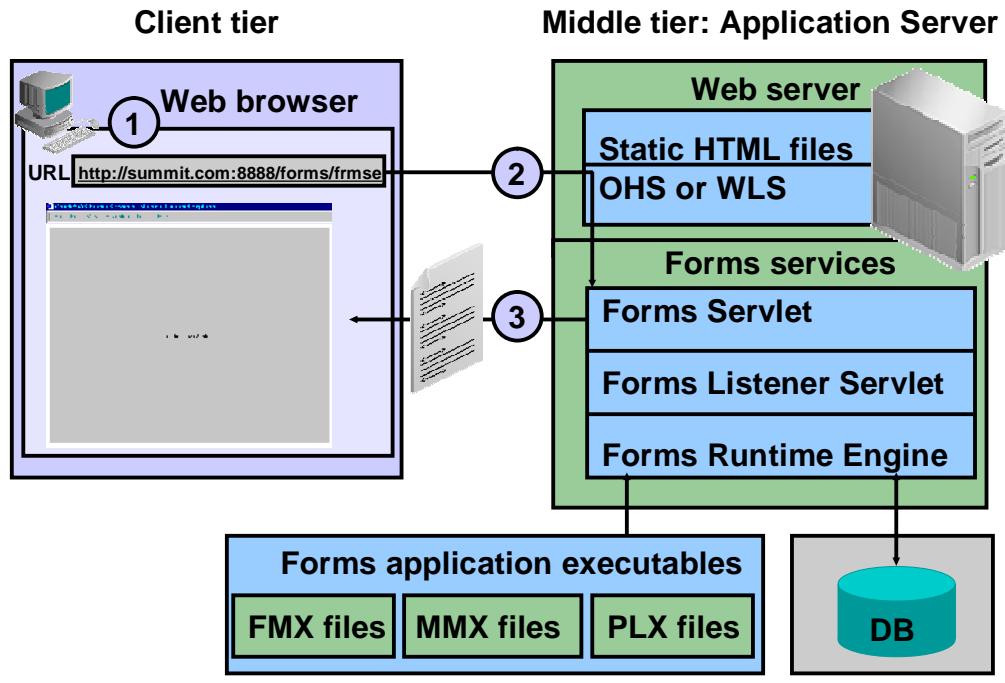
The Web browser can run a Java applet because it provides a JRE. However, the advanced features of the Forms Client require the Sun JavaSoft 1.6.x plug-in (other JREs may be supported in later patch releases) with one of the following browsers:

- **IE 7.x:** When a user attempts to run a Forms application with the Internet Explorer browser on a computer that does not have Java 1.6 installed, the user can download the plug-in, and it is automatically installed.
- **Firefox 3.x browser:** When a user attempts to run a Forms application with the Mozilla Firefox browser on a computer that does not have Java 1.6 installed, Firefox does not download the JRE 1.6 plug-in automatically. Instead, Firefox displays the following message: “Additional plugins are required to display this page...”. The workaround is to download the JRE 1.6 plug-in by clicking the Install Missing Plugin link and install it manually, then restart Firefox.

The slide shows a symbol of the Sun JavaSoft Plug-in.

For more information about using JPI, see the following white paper:
Oracle 10gR2 Forms Services – Using Sun’s Java Plug-in

Starting a Run-Time Session



2 - 8

Copyright © 2009, Oracle. All rights reserved.

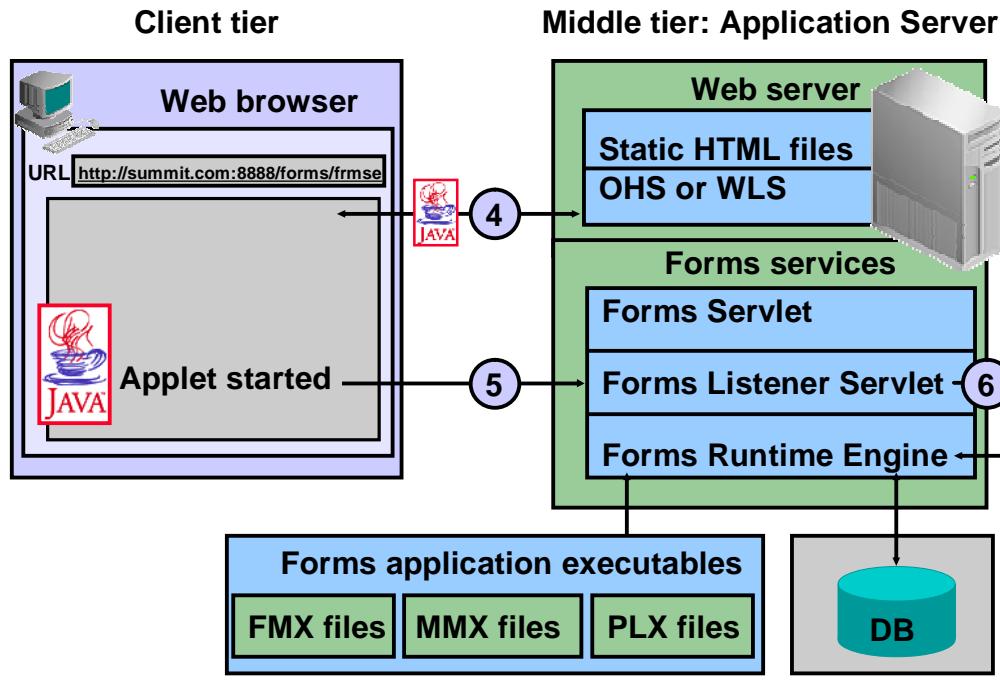
ORACLE®

Starting a Run-Time Session

Starting a run-time session involves the following steps, which are depicted as numbered in this and the following slides:

1. The user accesses the URL that indicates that a Forms application should be run.
2. The Oracle HTTP Server (OHS) or the Oracle WebLogic Server (WLS) receives an HTTP request from the browser client and contacts the Forms Servlet.
3. The Forms Servlet dynamically creates an HTML page containing all the information to start the Forms session.

Starting a Run-Time Session



2 - 9

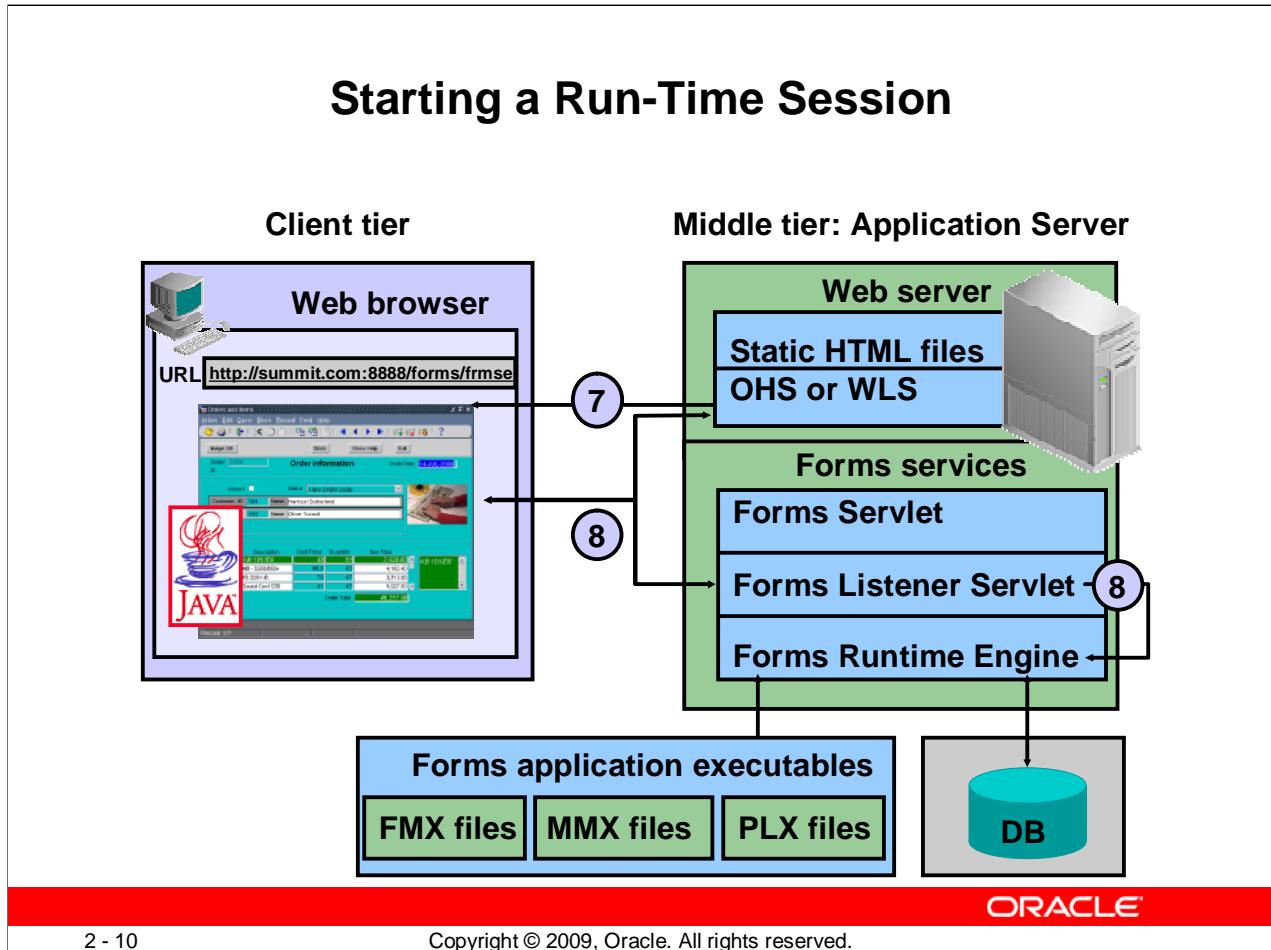
Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Starting a Run-Time Session (continued)

4. The Oracle HTTP Server or the Oracle WebLogic Server downloads a generic applet to the client after checking that it has not already been downloaded. The client caches the applet so that it can run future Forms applications without downloading it again.
5. The client applet contacts the Forms Listener Servlet to start the session. The Forms Listener Servlet starts an instance of the Forms Runtime Engine on the Forms Server (middle tier). If included in the HTML file, Forms Runtime command-line parameters (such as form name, user ID and password, database SID, and so on) and any user-defined Forms Builder parameters are passed to the process by the Forms Listener Servlet.
6. The Forms Listener Servlet establishes a connection with the Forms Runtime Engine, which connects to the database if needed and loads application executable files.

Starting a Run-Time Session



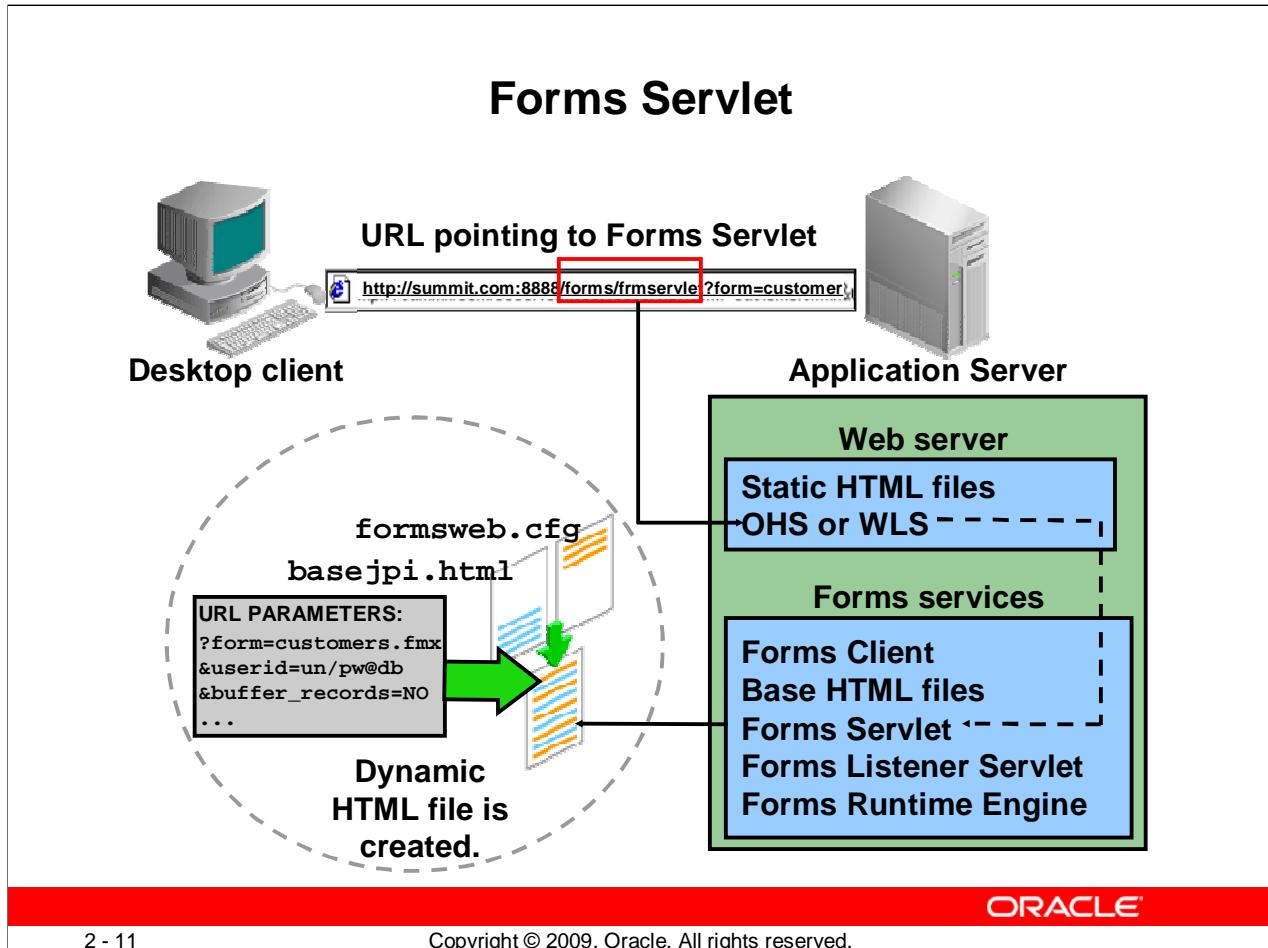
Starting a Run-Time Session (continued)

7. The Forms applet displays the user interface of the application in the main window of the user's Web browser.
8. The Forms Listener Servlet, working through HTTP Server or WebLogic Server, manages communication between the Forms applet and the Forms Runtime Engine.

Technical Note

For more information about Oracle Forms Listener Servlet and the connection process, refer to the *Oracle Forms Services & Oracle Forms Developer 11g Technical Overview* white paper, available on OTN.

Forms Servlet



2 - 11

Copyright © 2009, Oracle. All rights reserved.

Forms Servlet

The Forms Servlet is a Java servlet that creates a dynamic HTML file by merging information from the following sources:

- The Forms Web configuration file
- The Forms base HTML file
- The application's URL parameters

The graphics in the slide show:

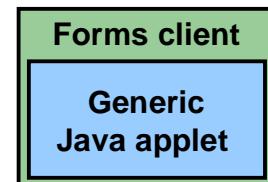
- A desktop client sending a URL to the application server; the URL contains “forms / frm servlet” to invoke the Forms Servlet
- The Web server receiving the request and passing it to the Forms Servlet component of Forms Services
- The Forms Servlet constructing the HTML file by combining parameters from the formsweb.cfg file, the basejpi.html file, and the parameters that are passed on the URL

Forms Client

- Generic Java applet
- Responsibilities:
 - Displays the form's user interface
 - Processes user interaction back to Forms Services
 - Processes incoming messages from Forms Services



Desktop client



Forms Client

The Forms Client is a generic Java applet. Forms Services dynamically downloads this applet and automatically caches it on the client machine. The Forms Client consists of a set of Java classes. At startup, only those Java classes that are necessary to initialize the application are downloaded. Additional class files are downloaded dynamically, as needed, to support additional user interface activity.

You do not have to deploy a separate Java applet for each application. The same generic applet is used to run any Forms Services application, regardless of its size and complexity.

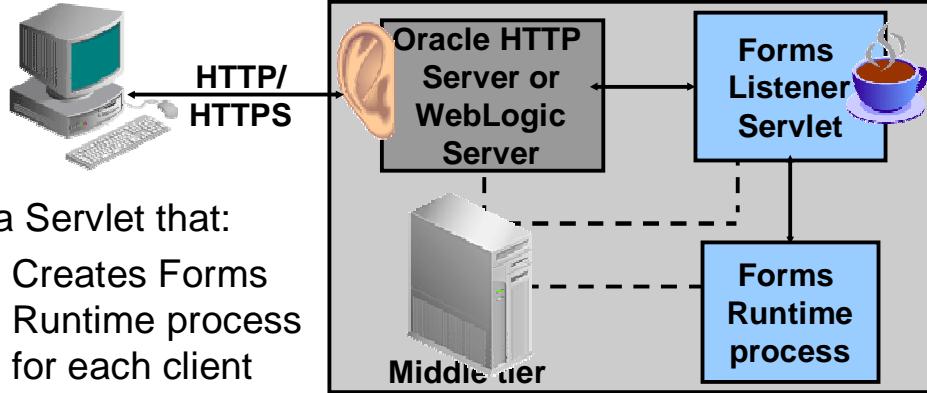
Responsibilities of the Forms Client

The Forms Client represents the user interface layer and has three primary functions:

- To render the Forms Services application display for the user
- To efficiently process user interaction back to Forms Services
- To process incoming messages from Forms Services and translate them into interface objects for the end user efficiently

The slide graphics depict the Forms Client that is downloaded to the desktop client computer in the form of a generic Java applet.

Forms Listener Servlet



Java Servlet that:

- Creates Forms Runtime process for each client
- Stops the Runtime process at the end of a session
- Manages network communications between the client and the Forms Runtime process
- Communicates through a Web server process

ORACLE®

2 - 13

Copyright © 2009, Oracle. All rights reserved.

Forms Listener Servlet

The Forms Listener Servlet is a Java servlet that runs in a Web server equipped with a servlet engine, such as WebLogic Server. The Web server directs HTTP requests for the Forms Listener Servlet directly to the servlet instances.

The Forms Listener Servlet is in charge of:

- Managing the creation of the Forms Runtime process for each client
- Managing the network communications that occur between the client and its associated Forms Runtime process, through the Web server

This means that the client sends HTTP requests and receives HTTP responses from the Web server process itself. Because the Web server acts as the network endpoint for the client, there is no need to expose additional server machines and ports at the firewall.

The graphics in the slide show that the Forms Listener Servlet creates the Forms Runtime process and manages communication between that process and the desktop client.

Forms Runtime Engine

The Forms Runtime Engine:

- Is a process (`frmweb`) that runs on the Application Server
- Manages application logic and processing
- Communicates with the client browser and the database

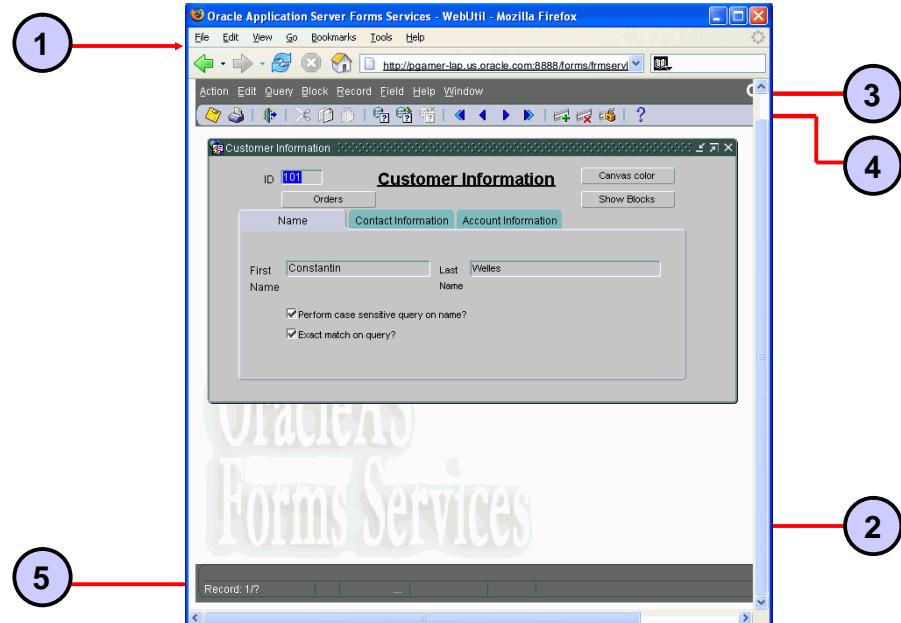


Forms Runtime Engine

The Forms Runtime Engine is a process on the Application Server that is started by the Forms Listener Servlet. You cannot start the runtime engine directly.

The Forms Runtime Engine handles all the application logic and Forms functionality and executes the code written into the application. It manages requests from the Forms Client and sends metadata to the client to describe the user interface. It connects to and communicates with the Oracle database via Oracle Net Services, the replacement for Net8 and SQL*Net.

What You See at Run Time



What You See at Run Time

At run time, you will see the following components that are pictured and numbered as follows in the screenshot in the slide:

1. Browser window
2. Java applet (contained within the browser window)
3. Default menu (contained within the applet)
4. Menu toolbar (contained within the applet)
5. Console (contained within the applet)

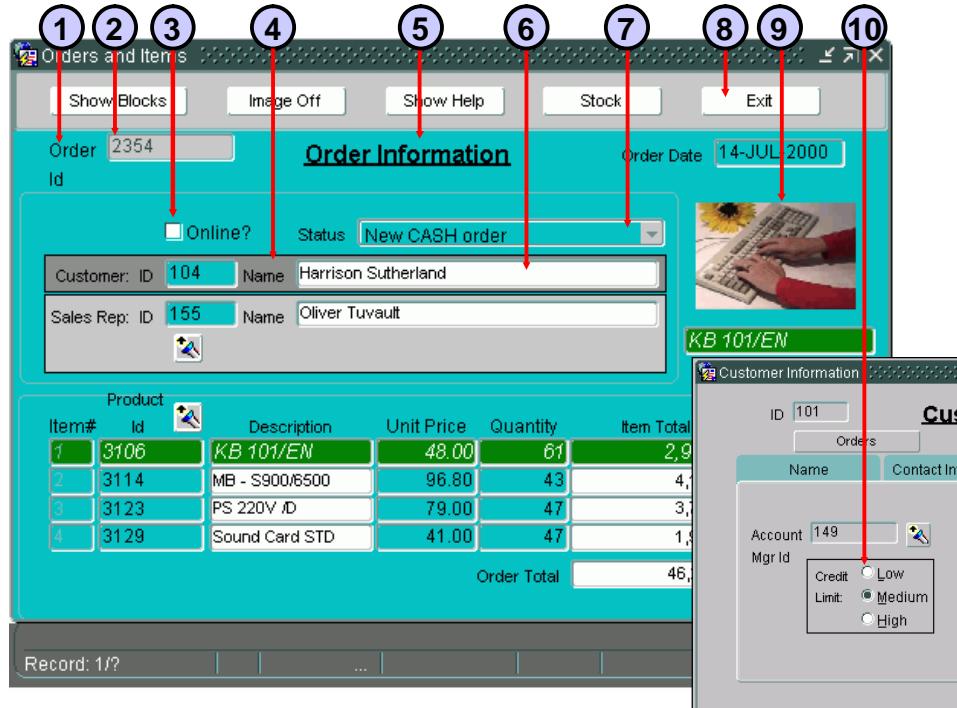
What Is the Default Menu?

The *default menu*, which is part of all Oracle Forms applications, is an alternative to keystroke operations. You can replace or customize the Default menu to introduce your own functionality into a form module.

What Is the Menu Toolbar?

The *menu toolbar* contains buttons corresponding to menu items. At run time, it appears above any user-defined toolbars. It executes the same code as menu items, and it is a shortcut to menu commands that does not duplicate code or effort.

Identifying the Data Elements



ORACLE®

Identifying the Data Elements

A Forms application may contain different kinds of data elements that are pictured and numbered in the screenshot in the slide:

1. Prompts
2. Text items
3. Check boxes
4. Boilerplate graphics
5. Boilerplate text
6. Display items
7. List items
8. Push buttons
9. Image items
10. Radio groups

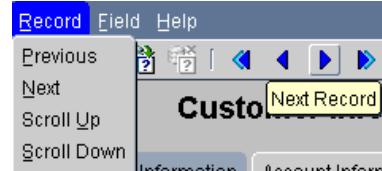
Not pictured:

- Hierarchical tree items
- Chart items
- Custom items

Navigating a Forms Application

Methods of navigation:

- Default menu
- Menu toolbar
- Mouse
- Buttons
- Function keys



ORACLE®

2 - 17

Copyright © 2009, Oracle. All rights reserved.

Navigating a Forms Application

Default Menu

The default menu, shown in the top screenshot in the slide, is automatically available in a form, unless it is disabled or replaced with a customized menu. Select options from the menu by using the mouse or function keys. At run time, use the menu to perform the following tasks:

- Move the cursor and navigate between data blocks, records, and items.
- Save or clear all changes.
- Execute queries.
- Insert new records or delete existing records.
- Invoke Help.

Menu Toolbar

You can use the default menu toolbar buttons, also shown in the top screenshot, to perform the following operations, which are also available through the default menu:

- Save all the changes.
- Exit the form.
- Execute queries.
- Navigate between data blocks or records.

Navigating a Forms Application (continued)

- Insert new records or delete existing records.
- Invoke Help to see the properties of an item.

Mouse

You can use the mouse to navigate and to perform many user operations in a bitmapped environment without needing to learn the function keys. Use the mouse to perform the following actions:

- Move the cursor.
- Select from a menu.
- Select from a list of values (LOV).
- Select or deselect a check box.
- Select a button, including an option button.
- Switch to an open window.
- Respond to an alert.
- Scroll records or lines by using a data block or item scroll bar.
- Manipulate a custom item.

Buttons

Web applications often use buttons as a means of navigation. You can click buttons with the mouse. Use buttons to perform the following tasks:

- Move input focus.
- Display a list of values.
- Invoke an editor.
- Invoke another window.
- Commit data.
- Issue a query.
- Perform calculations.
- Exit the form.

Function Keys

In addition to navigating with the mouse, you can move from item to item in sequence with function keys. Use function keys to perform the following tasks:

- Navigate between data blocks, records, and items.
- Execute queries.
- Insert new records or delete existing ones.
- Invoke Help.

To view a list of keys and the functions they perform, select Help > Keys, or press Ctrl + K. The Keys window, shown in the bottom screenshot in the slide, displays the list of keys and functions.

Modes of Operation: Enter-Query Mode

Allows:

- Unrestricted and restricted queries
- Query/Where dialog box
- Record count by using Query > Count Hits

Does not allow:

- Navigation out of the current data block
- Exiting the run-time session
- Certain functions
- Insert, update, delete

ORACLE®

2 - 19

Copyright © 2009, Oracle. All rights reserved.

Modes of Operation: Enter-Query Mode

Forms Builder has two main modes of operation: Enter-Query mode and Normal mode. The third mode of operation, Query mode, occurs while Forms is processing a query; the user cannot interact with the form while this query processing is taking place.

Enter-Query Mode

Use Enter-Query mode to enter search criteria for a database query. In Enter-Query mode, your keystrokes are interpreted as search criteria for retrieving restricted data.

You can do the following in Enter-Query mode:

- Retrieve all records.
- Retrieve records by using selection criteria.
- Retrieve records by using the Query/Where dialog box.
- Obtain the number of records that will be retrieved before fetching them from the database by using Query > Count Hits.

You cannot do the following in Enter-Query mode:

- Navigate out of the current block.
- Exit from the run-time session.
- Use certain functions, such as Next Record.
- Insert new records.
- Update existing records.
- Delete records.

Modes of Operation: Normal Mode

Allows:

- Unrestricted queries
- Insert, update, delete
- Commit (Save)
- Navigation out of the current data block
- Exiting the run-time session

Does not allow:

- Restricted queries
- Query/Where dialog box

ORACLE®

2 - 20

Copyright © 2009, Oracle. All rights reserved.

Modes of Operation: Normal Mode

Use Normal mode to insert and alter records in the database. In Normal mode, your keystrokes are interpreted as either the entering of new records or the altering of existing ones.

You can do the following in Normal mode:

- Retrieve all records.
- Insert new records.
- Update records.
- Delete records.
- Commit (Save) records.
- Roll back (Clear) records.
- Navigate out of the current data block.
- Exit the run-time session.

You cannot do the following in Normal mode:

- Retrieve a restricted set of records.
- Invoke the Query/Where dialog box.

Retrieving Data

Unrestricted query

A	B	C	D
1			
2			
3			
4			

A	B	C	D
1			
2			
3			
4			

Restricted query

A	B	C	D
1			
2			

A	B	C	D
1			
2			
3			
4			

ORACLE®

2 - 21

Copyright © 2009, Oracle. All rights reserved.

Retrieving Data

You can use a form module to retrieve information from the database without knowing any SQL syntax. However, if you are an experienced SQL user, you may want to supplement Forms default processing with your own SQL predicates. There are two general types of queries:

Query Type	Description
Unrestricted (global query)	The equivalent of selecting all the rows for all the represented columns from the base table for the queried data block; depicted by the graphics at the left of the slide
Restricted	The equivalent of selecting a restricted set of rows for all the represented columns from the base table for the queried data block; depicted by the graphics at the right of the slide

Performing an Unrestricted Query

You can retrieve unrestricted data by performing one of the following actions:

- Select Query > Execute.
- Press the appropriate function key.
- Click the Execute Query button.

Note: You cannot perform a query while you have unsaved updates, inserts, or deletes. Either save or undo the changes before you continue with the query.

Performing a Restricted Query

- Do not use quotation marks with character and date items.
- The `LIKE` operator is implied with `%` or `_`.
- Use hash (#) in front of SQL operators.
- Use Query/Where for complex query conditions.
- Use default date format (DD-MON-RR) in Query/Where.
- Use double quotation marks around literals in Query/Where.

ORACLE®

2 - 22

Copyright © 2009, Oracle. All rights reserved.

Performing a Restricted Query

You can use any one of the following methods to perform a restricted query:

- Matching values
- Matching patterns (wildcards)
- A Query/Where dialog box for user entry of SQL predicates

Valid Search Criteria

Item	Criterion	Uses
Order_Id	2356	Exact match
Customer_Id	%04	Implied LIKE operator
Order_Id	#BETWEEN 2350 AND 2360	BETWEEN operator
Order_Status	“CREDIT order paid” selected from pop-up list	Exact match on item value (10)
Sales_Rep_Id	:S	Query/Where dialog box

Performing a Restricted Query (continued)

How to Perform a Restricted Query

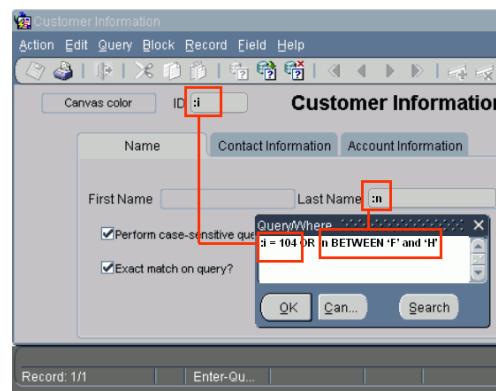
You can perform a restricted query by using the following steps:

1. Perform one of the following:
 - Select Query > Enter.
 - Click the Enter Query button.
 - Press the appropriate function key.
2. Enter-Query displays on the status line.
3. Enter search criteria into appropriate items.
4. Perform one of the following:
 - Select Query > Execute.
 - Click the Execute Query button.
 - Press the appropriate function key.

Note: Forms Builder constructs a select statement by using the AND operator for all the specified conditions.

Using the Query/Where Dialog Box

- Invoke by:
 - Entering
 - :<variable_name>
 - &<variable_name>
 - :
 - &
 - Executing query
- Used to write:
 - Complex search conditions
 - Queries with OR predicates
 - The ORDER BY clause



Using the Query/Where Dialog Box

In the Query/Where dialog box, you can enter complex search criteria by using raw SQL. To use the Query/Where dialog box effectively, you should have knowledge of SQL. Use Query/Where to perform the following tasks:

- Write complex search conditions.
- Write queries with OR predicates.
- Order the result of a query.

Forms Builder logically uses the AND operator to append the Query/Where conditions to any other search criteria (including those imposed by the form designer) and constructs a SELECT statement. The Query/Where dialog box does not work in an item whose name differs from the name of its corresponding database column.

To improve security, Forms 10.1.2.0.2 introduced a new run-time environment variable, FORMS_RESTRICT_ENTER_QUERY, that is initially set to TRUE, which makes it impossible to invoke the Query/Where dialog box. You must set this environment variable to FALSE (or comment out the line that sets it to TRUE) to enable the use of the Query/Where dialog box. The lesson titled “Working in the Forms Environment” discusses environment variables and how to set them.

Using the Query/Where Dialog Box (continued)

To use the Query/Where dialog box to enter query criteria:

1. Perform one of the following:
 - Select Query > Enter.
 - Click Enter Query.
 - Press the appropriate function key.
2. Enter a colon (:) or an ampersand (&) followed by a unique character variable name in one or more items.
3. Perform one of the following: Select Query > Execute, click Execute Query, or press the appropriate function key.
Note: You can select Query > Count Hits if you simply want to know how many records will match your criteria.
4. The Query/Where dialog box is displayed.
5. Enter the search criteria by using variables, SQL, and logical operators. Click OK.
Note: To perform a query without any variables, enter only the colon (:) or ampersand (&) and execute the query. Doing so also displays the Query/Where dialog box.

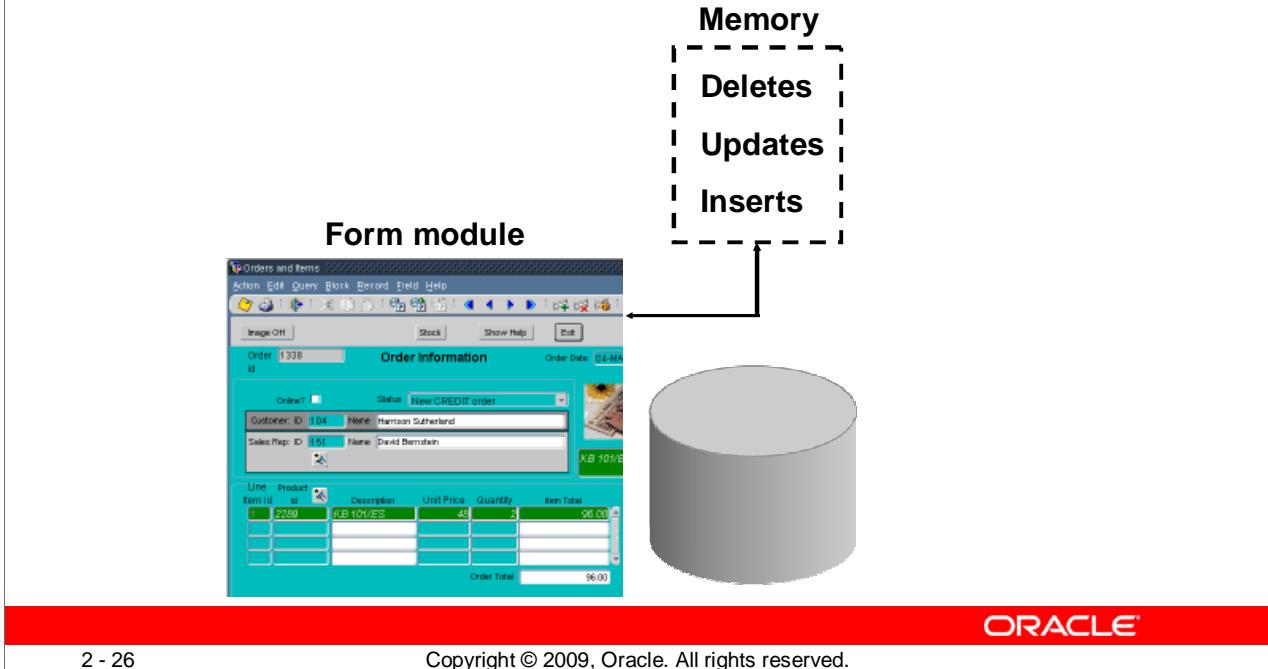
If you enter an ORDER BY at run time, it overrides any ordering defined by the designer.

The screenshot in the slide shows a restricted query being performed. In Enter-Query mode, the user enters :i in the ID field and :n in the Last Name field. When the user clicks Execute Query, the Query/Where dialog box appears, where the user enters the WHERE clause :i=104 OR n BETWEEN 'F' and 'H'. This query returns customer 104 along with all customers whose last name starts with F through G (although in an Oracle database the SQL operator BETWEEN is inclusive, any last name beginning with "H" would be greater than the single letter "H" and so would not be included.)

Example

- To restrict the query to orders with a Sales_Rep_Id (:S) of 11 or an Order_Id (:O) between 100 and 200, enter the following in the Query/Where dialog box:
 :S = 11 OR :O between 100 and 200
- To sort the data by Sales_Rep_Id (:S), enter the following in the Query/Where dialog box:
 ORDER BY :S

Inserting, Updating, and Deleting Records



Inserting, Updating, and Deleting Records

Upon entering a typical form module, you are in Normal mode. This means that Forms Builder regards anything that you enter into a blank record as an insert and anything that you enter over an existing record as an update. The graphics in the slide depict the fact that all deletes, updates, and inserts that are performed in Forms are retained in memory until they are committed to the database.

How to Insert a Record

To insert a record, perform the following steps:

1. Ensure that you have the cursor positioned on a blank record by performing one of the following steps:
 - Scroll down until you find a blank record (always the last in the block).
 - Select Record > Insert.
 - Click Insert Record (green +).
 - Press the appropriate function key.
2. Enter the data into the relevant items.

Inserting, Updating, and Deleting Records (continued)

How to Update a Record

To update a record, perform the following steps:

1. Select Query > Enter.
2. Enter the search criteria to retrieve the appropriate record.
3. Select Query > Execute to retrieve all the records that satisfy your specific search criteria.
4. Scroll through the records, and stop at the record that is to be updated.
5. Update the record.

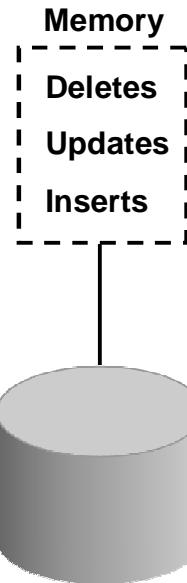
How to Delete a Record

To delete a record, perform the following steps:

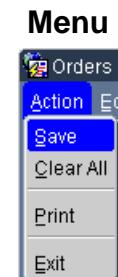
1. Select Query > Enter.
2. Enter the search criteria to retrieve the appropriate record.
3. Select Query > Execute to retrieve all records that satisfy your specific search criteria.
4. Scroll through the records, and stop at the record that is to be deleted. Delete the record by performing one of the following actions:
 - Select Record > Remove to clear the record and mark it for deletion.
 - Click Remove Record (red X) to clear the record and mark it for deletion.
 - Press the appropriate function key.

Making Changes Permanent

- Select Action > Save to make changes permanent.
- Select Action > Clear All to discard changes.



To commit or
roll back:



or
Toolbar



ORACLE®

Making Changes Permanent

As previously stated and as depicted by the slide graphics, changes are retained in memory. To make any inserts, updates, or deletes permanent, you must save (commit) them to the database. To do this, perform one of the following actions:

- Select Action > Save (shown in the top right screenshot).
- Click Save on the Menu toolbar (shown in the bottom right screenshot).

Discarding Inserts, Updates, and Deletes

To discard any inserts, updates, or deletes, you must clear the records (roll back) instead of saving. Perform a rollback by selecting Action > Clear All.

Exiting a Run-Time Session

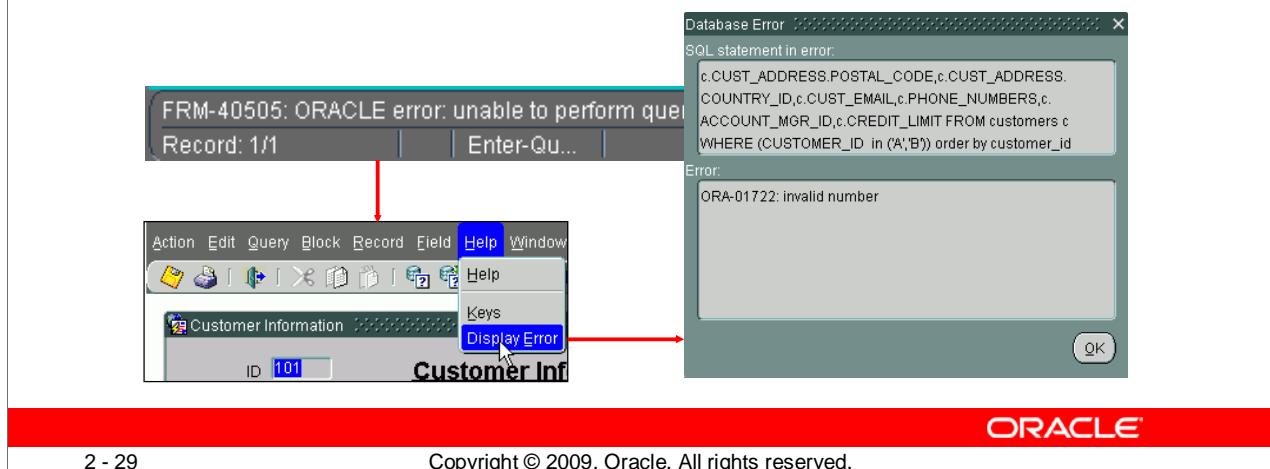
You exit the run-time session by performing one of the following actions:

- Select Action > Exit.
- Click Exit.
- Press the appropriate function key.

Note: By default, you cannot exit the form while you have unsaved updates, inserts, or deletes. You need to either save or undo the changes before you can exit.

Displaying Errors

- Use to view Oracle errors.
- Select Help > Display Error.
- Shows the Database Error window:
 - SQL statement
 - Error information



Displaying Errors

If an Oracle error is displayed on the message line while you are operating a Forms application, you can view the underlying SQL code by selecting Help > Display Error.

The screenshots in the slide show, at the top left, a form's status line showing error FRM-40404: ORACLE error: unable to perform query. The bottom left screenshot shows selecting Help > Display Error from the menu, and the screenshot at the right shows the Database Error window that shows the SQL query statement at the top and the database error at the bottom: ORA-01722: invalid number.

Example

The following is the SQL statement in error and its corresponding error:

```
SELECT order_id, order_date, order_mode, order_status,
       customer_id, sales_rep_id
  FROM orders
 WHERE (order_id in ('a','b'))
ORA-01722: invalid number
```

Note: Selecting Help > Display Error displays only those errors where the error on the message line is preceded by ORACLE error.

Summary

In this lesson, you should have learned that:

- You can use WebLogic Server on the development machine to run a Forms application in a Web browser
- At run time:
 - The Forms Client is downloaded
 - The Forms Servlet creates a start HTML file
 - The Forms Listener Servlet starts a run-time session and maintains communication between it and the Forms Client
 - The Forms Runtime Engine carries out application logic and maintains a database connection on behalf of the Forms Client

ORACLE®

2 - 30

Copyright © 2009, Oracle. All rights reserved.

Summary

This lesson introduced the operator interface of Forms Builder. The following concepts were covered in this lesson:

- Starting WebLogic Server on the development machine
- The run-time environment for Forms:
 - Running a form in a browser
 - Starting a run-time session: The Forms Servlet, the Forms Client, and the Forms Listener Servlet
- The data elements of a form

Summary

- When you run a form, you see a Java applet running in a browser and displaying a menu, menu toolbar, console, and several kinds of data elements
- Users navigate a Forms application using the menu, toolbar, the mouse, buttons, or function keys
- The two main modes of operation are Normal mode and Enter-Query mode
- Executing a query returns all records, unless the query is restricted by search criteria

ORACLE®

2 - 31

Copyright © 2009, Oracle. All rights reserved.

Summary (continued)

- Elements of a running form
- Navigation methods
- Modes of operation:
 - Normal mode
 - Enter-Query mode
(There is also a Query mode that occurs when the form is fetching records; the operator cannot interact with the form in Query mode.)
- Retrieving data by performing:
 - Restricted queries: You supply the search criteria.
 - Unrestricted queries: You supply no search criteria.

Summary

- In Normal mode, you can insert, update, and delete records and commit changes to the database
- You display database errors from the menu (Help > Display Error)

ORACLE®

2 - 32

Copyright © 2009, Oracle. All rights reserved.

Summary (continued)

- Inserting, updating, and deleting records
- Saving or clearing changes
- Displaying database errors

Practice 2: Overview

This practice covers the following topics:

- Starting the Forms WebLogic Managed Server
- Running the course application:
 - Querying records
 - Inserting a record
 - Updating a record
 - Deleting a record



Practice 2: Overview

In this practice session, you start an instance of the Forms managed WebLogic Server to function as a Web server on your local machine. You run the course application in a browser and execute both unrestricted and restricted queries. You navigate through the application and use it to insert, update, and delete records.

Working in the Forms Environment



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Identify the main Forms executables
- Describe the Forms module types
- Describe Forms Builder components
- Navigate the Forms Builder interface
- Identify the main objects in a form module
- Customize the Forms Builder session
- Use the online Help
- Set environment variables for design and run time
- Run a form from within Forms Builder

ORACLE®

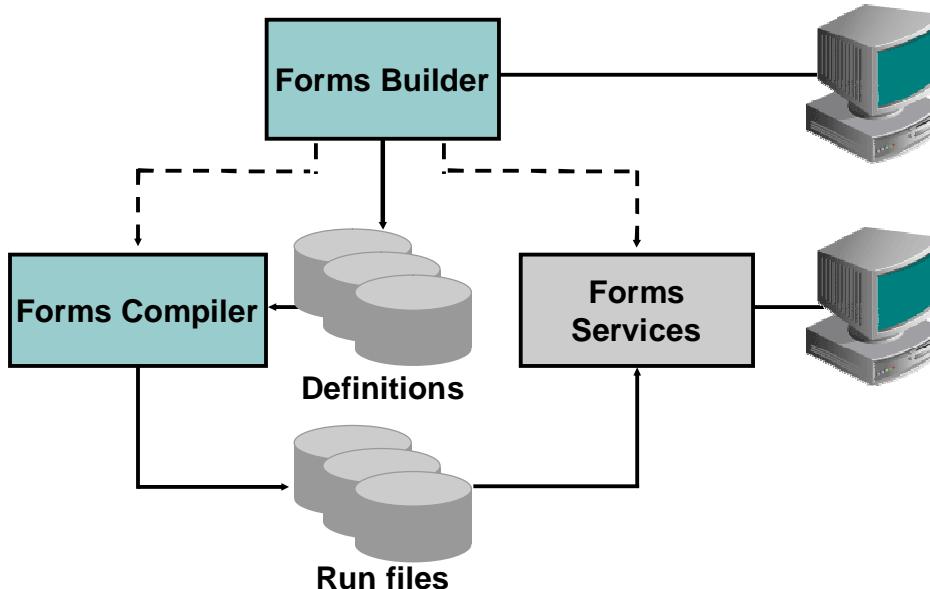
3 - 2

Copyright © 2009, Oracle. All rights reserved.

Lesson Aim

This lesson provides you with an overview of Forms Builder, including a high-level description of its components and object hierarchy. Using this knowledge, you can plan and implement the structure of your Forms applications.

Forms Developer Executables



ORACLE®

3 - 3

Copyright © 2009, Oracle. All rights reserved.

Forms Developer Executables

Forms Builder includes the following two executables (components) that you can access as the designer of applications.

- **Forms Builder:** This is the application-building component of Oracle Forms Developer. You can use Forms Builder to design and store the definitions of form, menu, and library documents. While in the Forms Builder, you can invoke the other component, Forms Compiler. You must run the Forms Builder component in a graphical user interface (GUI) environment in order to use its graphical design facilities.
- **Forms Compiler:** After your form is built, use the Forms Compiler. This reads the definition of your module and creates an executable run file.

Invoking Forms Builder Executables

In a GUI environment, you usually store commands to invoke Forms Builder components in menus and icons for convenient access. You can also enter these commands on the command line. For example:

```
FRMBLD [my_form] [scott/tiger@my_database]
```

Note: Commands for invoking the product components vary according to platform.

Forms Developer Executables (continued)

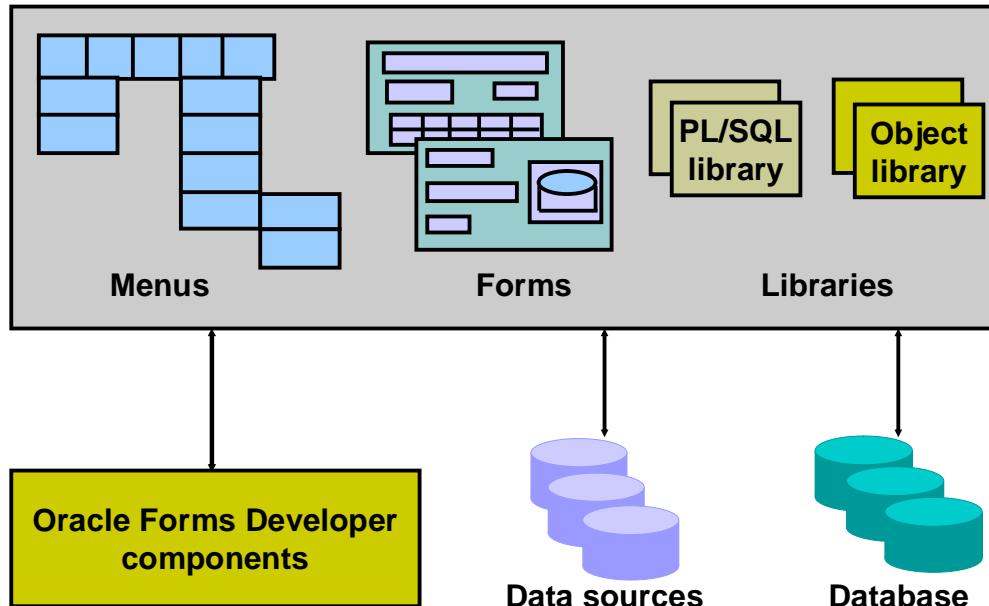
Forms Services

Because Forms applications are Web based, it is not possible to run them directly from the command line. Instead, they are invoked by entering a URL, directed to Forms Services, in a browser.

The files used at run time must already have been compiled by the Forms Compiler component. These files must reside on the middle-tier machine in a directory accessible to the Forms Runtime Engine (in FORMS_PATH).

To test your applications, you can also access Forms Services directly from Forms Builder by setting certain preferences, as described later in this lesson.

Forms Developer Module Types



3 - 5

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Forms Developer Module Types

A Forms application can consist of many modules—that is, files. A module is a major component of your application and is the basis for storage and ownership. A module owns the objects that it contains.

A Forms Developer module can be of the following types, as pictured in the graphics in the slide:

- **Form:** As the main component of an application, the form module presents the objects and data that users can see or interact with. Data items in a form are arranged into records.
- **Menu:** A menu module can consist of a hierarchy of menus, each with selectable items. Forms Builder provides the default menu for every form. The default menu includes commands for all basic database operations, such as insert, delete, query, and so on. If your application has specific requirements that are not met by the default menu, you can create a custom menu module. You can use a menu module with multiple forms.

Forms Developer Module Types (continued)

- **PL/SQL Library:** A PL/SQL library is a collection of PL/SQL program units whose code can be referenced and called from other modules. PL/SQL library documents can contain program units that can be used by other form and menu modules.
- **Object Library:** An object library is a collection of form objects that you can use in other modules. You can create it to store, maintain, and distribute standard objects that can be reused across the entire development organization.

You can build an application from multiple form modules, menu modules, and library documents as needed. A form module can be run independently, but menu modules, PL/SQL libraries, and object libraries are functional only when attached to or included in a form module.

Forms Builder: Key Features

With Forms Builder you can:

- Provide an interface for users to insert, update, delete, and query data
- Present data as text, image, and custom controls
- Control forms across several windows and database transactions
- Use integrated menus
- Send data to Oracle Reports



Forms Builder: Key Features

Forms Builder is a major component of Oracle Forms Developer. You can use Forms Builder to quickly develop form-based applications for presenting and manipulating data in a variety of ways.

Users of Forms Builder applications can:

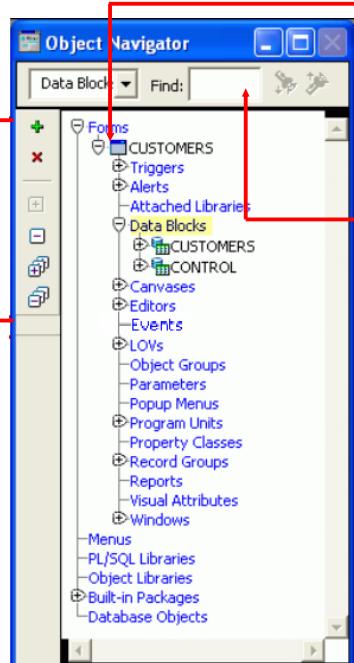
- Insert, update, delete, and query data by using a variety of interface items
- Present data by using text, image, and custom controls, including JavaBeans and Pluggable Java Components
- Control forms across several windows and database transactions
- Access comprehensive facilities by using integrated menus
- Send data directly to Oracle Reports

As the designer of Forms Builder applications, you can:

- Design forms that use a number of data sources, including Oracle databases
- Build applications quickly and easily by using powerful GUI development tools
- Design applications for Internet deployment
- Copy and move objects and their properties easily between applications
- Use design features such as wizards, the Layout Editor, Object Navigator, and PL/SQL Editor

Forms Builder Components: Object Navigator

- Objects displayed hierarchically
- Toolbar to create, delete or unload, and expand or contract



- Icons to represent objects
- Fast search feature

Forms Builder Components: Object Navigator

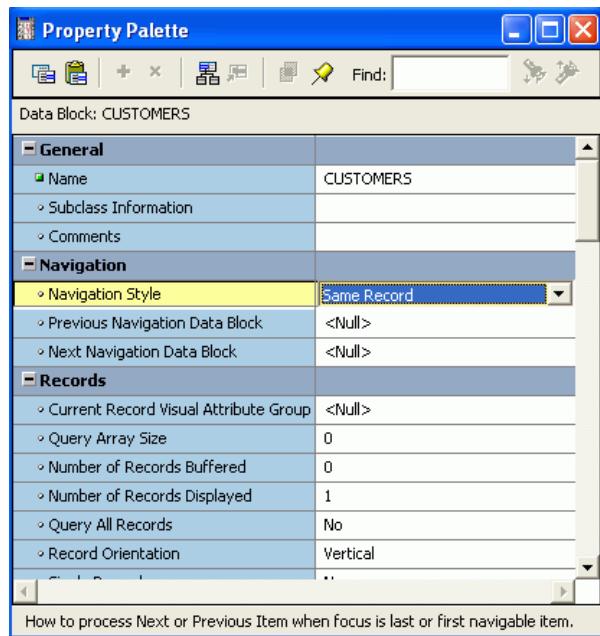
The interface components of the Forms Builder tool help to provide the flexibility and productivity of the Oracle Forms development environment.

The Object Navigator, pictured in the screenshot in the slide, is a hierarchical browsing and editing interface. You can use the Object Navigator to locate and manipulate application objects quickly and easily. Features include:

- A hierarchy represented by indentation and expandable nodes
(Top-level nodes show application objects [forms, menus, and libraries], database objects, and built-in packages. All other nodes and the objects they contain are indented to indicate that they belong to these higher-level nodes.)
- Find field and icons, enabling forward and backward searches for any level of node or for an individual item in a node
- Icons in the vertical toolbar to create, delete or unload, and expand or contract
- An icon next to each object to indicate the object type

Forms Builder Components: Property Palette

- Copy and paste properties
- Fast search feature



Forms Builder Components: Property Palette

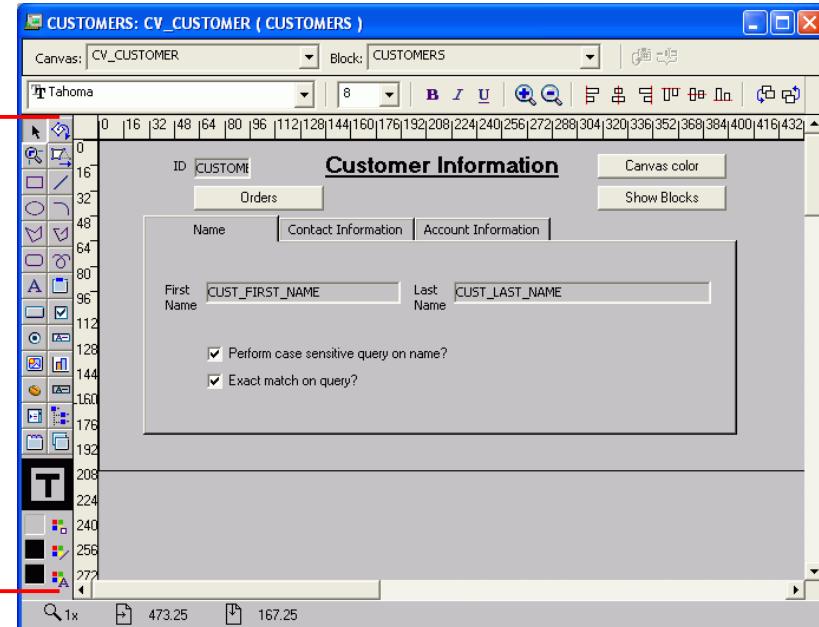
All objects in a module, including the module itself, have properties that you can see and modify in the Property Palette, shown in the screenshot in the slide. Features include:

- Copy and reuse properties from another object
- Find field and icons, similar to the Object Navigator

Forms Builder Components: Layout Editor

Toolbar

Tool palette



ORACLE®

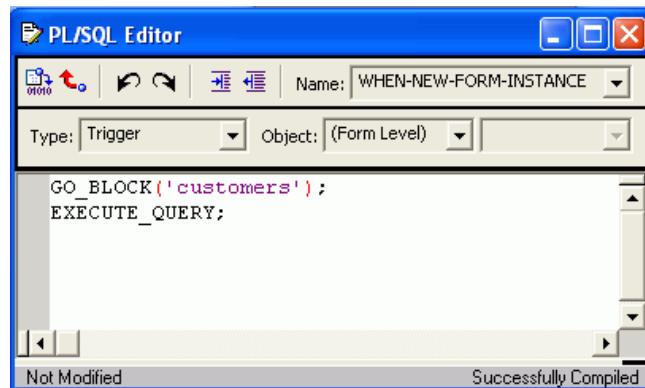
Forms Builder Components: Layout Editor

The Layout Editor, shown in the screenshot, is a graphical design facility for creating and arranging interface items and graphical objects in your application. You can use the tool palette and the toolbar available in the Layout Editor to design the style, color, size, and arrangement of visual objects in the application. The layout can include graphical objects and images.

Forms Builder Components: PL/SQL Editor

With the PL/SQL Editor, you can:

- Use PL/SQL in Forms
- Enter and compile code



ORACLE®

3 - 11

Copyright © 2009, Oracle. All rights reserved.

Forms Builder Components: PL/SQL Editor

The PL/SQL Editor, shown in the screenshot, enables you to incorporate PL/SQL code objects into your form. Code objects in Forms Developer include event triggers, subprograms (functions and procedures), menu item commands, menu startup code, and packages. You enter and compile code in the PL/SQL Editor. You will learn more about the PL/SQL Editor in later lessons when you use it to code triggers in Forms Builder.

Getting Started in the Forms Builder Interface



- Start Forms Builder.
- Connect to the database:
 - Menu:
Select File > Connect.
 - Or
 - Toolbar:
Click Connect.

ORACLE®

3 - 12

Copyright © 2009, Oracle. All rights reserved.

Getting Started in the Forms Builder Interface

Starting Forms Builder

To start Forms Builder on Windows, invoke it from the Start menu: Programs > Oracle Classic Instance > Developer Tools > Forms Builder.

When you invoke Forms Builder, a Welcome dialog box appears. If you click Cancel to dismiss the dialog box, you see the Object Navigator and an empty new module.

If you build applications that access database objects, you must connect to a database account from the Forms Builder. Connect to a database if you need to:

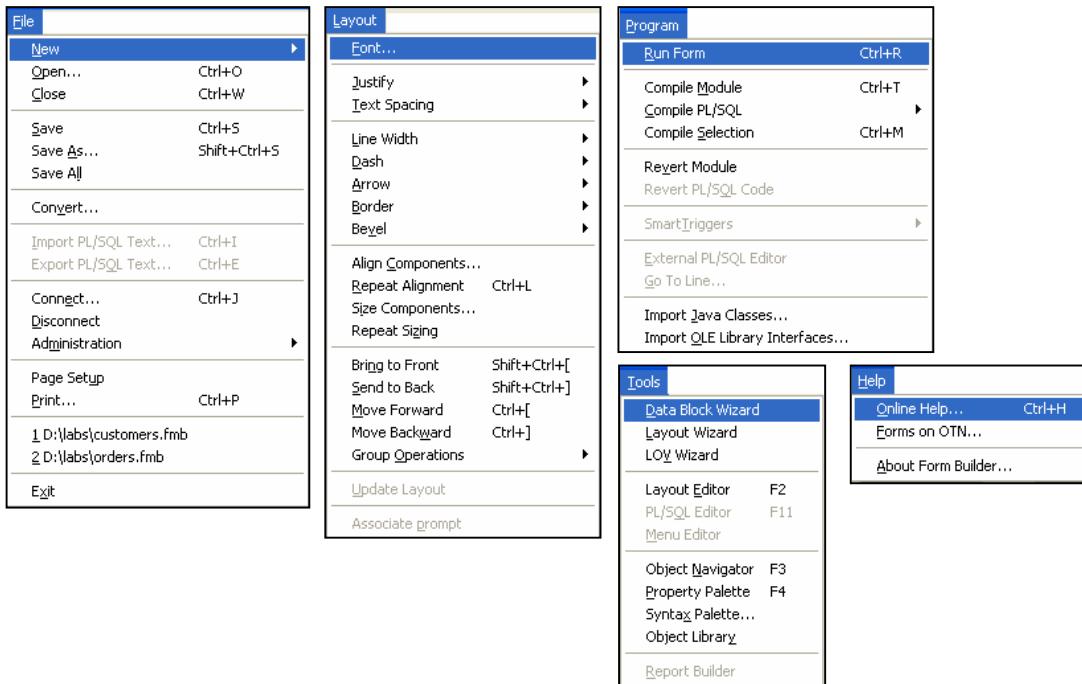
- Compile code that contains SQL
- Access database objects in the Object Navigator
- Create Oracle Forms Developer objects that are based on database objects

Connecting to a Database

1. Select File > Connect from the menu, or click the Connect icon on the toolbar as shown in the slide.
2. Enter the database user and password in the Connect dialog box. If not connecting to the default database, provide the necessary connect string or database alias.

Note: Oracle Forms Builder automatically displays the Connect dialog box if you try to perform a task that requires connection.

Navigating the Forms Builder Main Menu



ORACLE®

3 - 13

Copyright © 2009, Oracle. All rights reserved.

Navigating the Forms Builder Main Menu

The Forms Builder main menu contains options that enable you to create, modify, and manage your form modules.

Common Menu Features

The following table describes some common features in GUI menus:

Feature	Description
Underline	Shortcut key: Alt + letter
Ellipsis (...)	Additional input, usually by using a dialog box
▶	Menu option has a submenu
Windows menu	List of open windows; select any window to make it active
Help	List of help facilities and an About box

Native GUI Interface

The menus shown in the slide are screenshots of the Windows environment. However, menus appear with the same look and feel of your native GUI interface.

For example, in Motif, the Windows Print Dialog options appear as submenus of the Font menu.

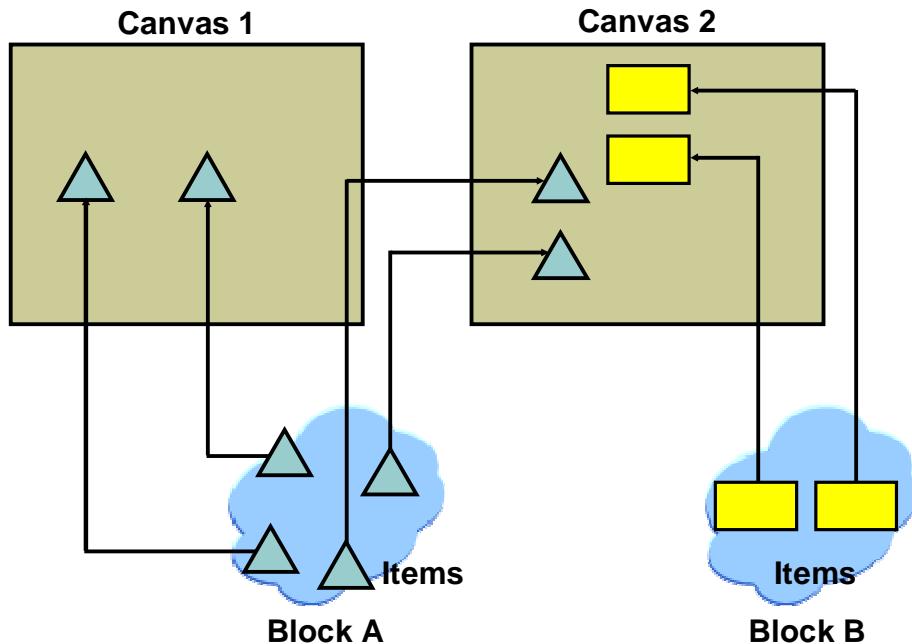
Navigating the Forms Builder Main Menu (continued)

Forms Builder Main Menu

Menu Item	Description
File *	Common file utilities, such as open, save, connect, and administration
Edit	Cut, copy, paste, create, preferences, and so on
View	Switch view in current window; options vary depending on context
Layout *	Common commands for use in the Layout Editor
Program *	Includes compilation and commands related to code
Debug	Invokes debugger functionality
Tools *	Access to wizards and other Forms Builder components
Window	Access to windows displayed in Forms Builder
Help *	Access to built-in and online help systems

* Pictured in the slide

Items, Blocks, and Canvases: Overview



Items, Blocks, and Canvases: Overview

Forms modules make up the main “body” of an Oracle Forms Developer application. They can consist of many types of objects, some of which are visible to the user at run time.

The three major objects in a form, as shown in the graphics in the slide, are:

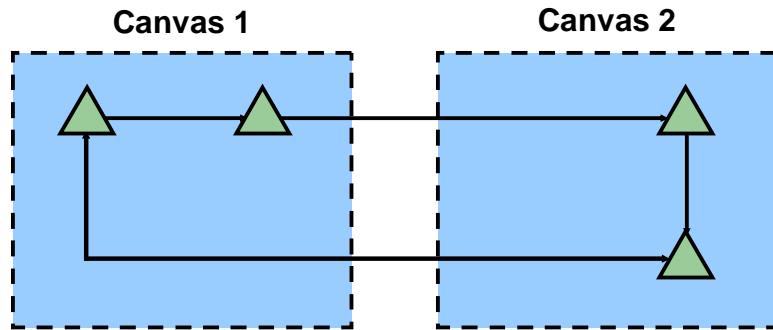
- **Items:** These are interface objects that present data values to the user or enable the user to interact with the form, depending upon the item type. There are several types of items. Items are logically grouped into blocks and visibly arranged on canvases.
- **Blocks:** A block is the intermediate building unit for forms. Each form consists of one or more blocks. A block is the logical owner of items, and each item in a form belongs to a block. Items in one block are logically related (for example, they may correspond to columns in the same database table or may need to be part of the same navigation cycle). Blocks, therefore, provide a mechanism for grouping related items into a functional unit for storing, displaying, and manipulating records.

Items, Blocks, and Canvases: Overview (continued)

- **Canvases:** A canvas is a “surface” where visual objects, such as graphics and items, are arranged. A form module can have several canvases (such as the pages of a paper form). A canvas can display items from one or more blocks. To see a canvas and its items, you must display the canvas in a window. A window can have more than one canvas. By default, all canvases in a form appear in the same window (which could mean you see only one canvas at a time), but you can assign separate windows for each canvas so that several canvases can be viewed at once. You learn more about windows and canvases in the lessons titled “Creating Windows and Content Canvases” and “Working with Other Canvas Types.”

Note: Items in one block do not need to be physically grouped. They can span many canvases (and windows). A canvas is similar to a picture portrait, and a window is similar to a picture frame. Just as you need a picture frame to display a picture portrait, you need a window to display a canvas and its contents.

Navigating in a Forms Module



ORACLE®

3 - 17

Copyright © 2009, Oracle. All rights reserved.

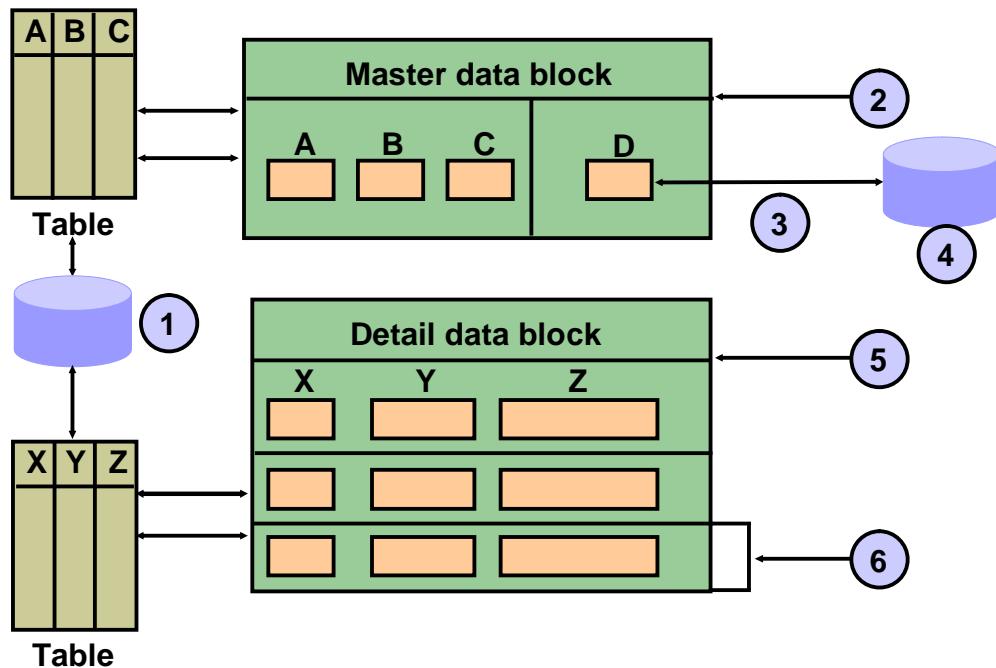
Navigating in a Forms Module

When you run a form, you principally navigate by way of items and blocks, not by canvases. Each item has a sequenced position within its block, and each block has a sequenced position in the form.

When a user requests to move to the next item in a block, focus will be set on the next item in sequence, wherever that may be. Default navigation follows the order of items in a block as listed in the Object Navigator, but you can change this. If the next item is on a different canvas, as shown in the slide, Forms displays that canvas automatically. Similarly, users can request to move to the next block (or previous block). If the first item in this block resides on another canvas, that canvas is displayed automatically.

If you can already see the item that you want to move to, you may click it. You can also program mechanisms into the application to enable navigation in other ways. You learn more about this in the lesson titled “Navigation.”

Using Blocks



Using Blocks

In Forms Builder, there are two main types of blocks: data blocks and control blocks.

Data Blocks

When you build database applications with Forms Builder, many of the blocks will be data blocks. A data block is associated with a specific database table (or view), a stored procedure, a FROM clause query, or transactional triggers.

If it is based on a table (or view), the data block can be based on only one base table, even though the data block can be programmed to access data from more than one table and data sources. By default, the association between a data block and the database enables the user to automatically access and manipulate data in the database. However, to access data from other tables (nonbase tables), you need to write triggers.

The graphics in the slide depict the following:

1. Base table source
2. Single-record data block
3. Trigger access
4. Nonbase table source
5. Multirecord data block
6. Record

Using Blocks (continued)

For a base table, Forms Builder can automatically perform the following actions:

- Create items in the data block to correspond to columns in the table. (These items are data items or base-table items.)
- Produce code in the form to employ the rules of the table's constraints.
- Generate SQL at run time (implicit SQL) to insert, update, delete, and query rows in the base table, based on the user's actions.

At run time, you can use standard function keys, buttons, menu options, or standard toolbar options to initiate query, insert, update, or delete operations on base tables, and the subsequent commit of the transaction.

Control Blocks

A control block is not associated with a database, and its items do not relate to any columns within any database table. Its items are called control items. For example, you can create many buttons in your module to initiate certain actions, and you can logically group these buttons in a control block.

Master Versus Detail Blocks

To support the relationship between data blocks and their underlying base tables, you can define one data block as the detail (child) of a master (parent) data block. This links primary key and foreign key values across data blocks, and synchronizes the data that these data blocks display.

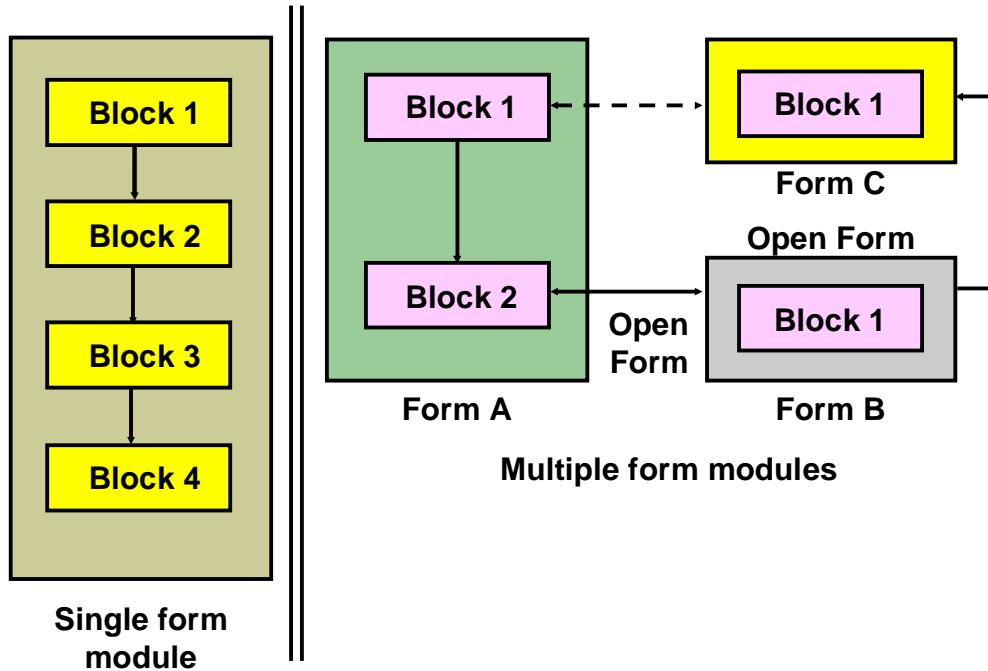
Forms Builder automatically generates the objects and code needed to support master-detail relationships. As the designer, you need only request it.

Note: If your application requires it, you can also create independent data blocks in which there is no relationship between the two data blocks.

Single-Record Versus Multirecord Blocks

You can design a data block to show one record at a time (single-record block) or several records at once (multirecord block). Usually, you create a single-record data block to show master block data and a multirecord data block to show detail block data. In either case, records in a data block that are currently not visible on the screen are stored in a block buffer.

Designing Multiblock and Multiform Applications



Designing Multiblock and Multiform Applications

Typically, a Forms Builder application consists of more than one data block. With more than one data block, you can do the following:

- Separate the navigation cycle of one group of items from another.
- Map each data block to a different database table. (You can have one base table per data block.)
- Produce a master-detail form, with a master data block and corresponding detail data blocks that are related to the master.

You can create a large form module with many data blocks, similar to the one shown in the left side of the slide. Alternatively, you can create several smaller form modules with fewer data blocks in each, as shown by the graphic in the right side of the slide.

Generally, a modular application with several smaller form modules has the following characteristics:

- Modules are loaded only when their components are required, thus conserving memory.
- Maintenance can occur on one module without regenerating or loading the others.
- Forms can call upon one another, as required.

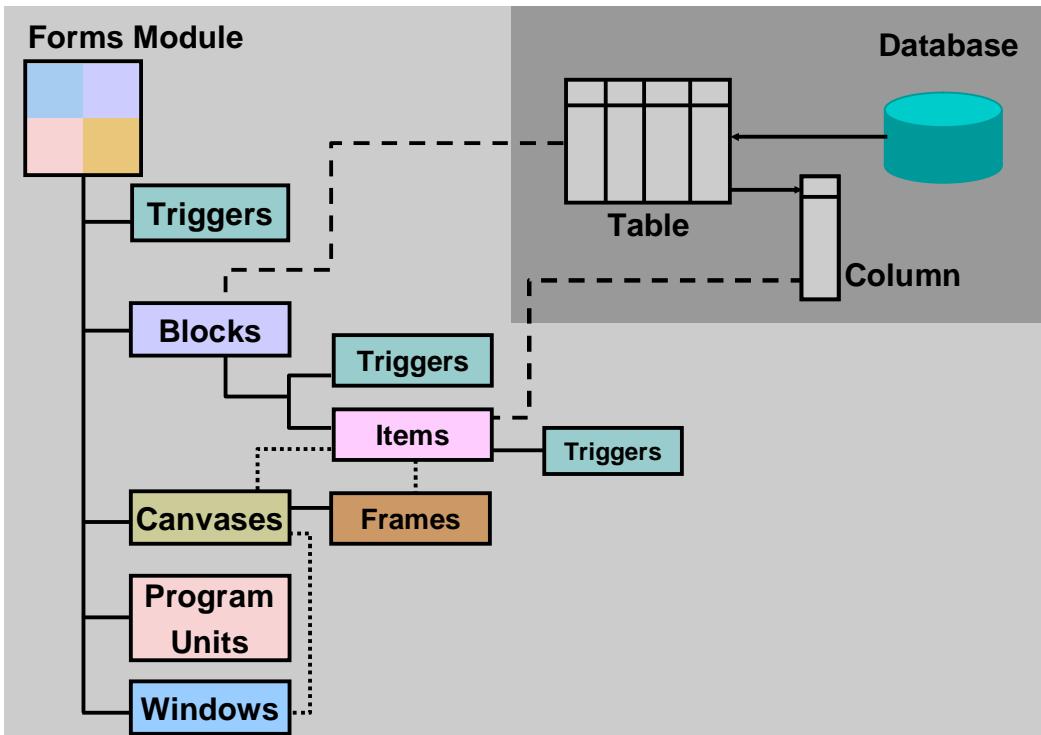
Multiblock and Multiform Applications (continued)

- Parallel development can be carried out by different team members on different components.

Here are some points to consider when grouping data blocks in the application:

Data Blocks in the Same Form Module	Data Blocks in Different Form Modules
The data blocks can be directly linked in master-detail relationships.	The data blocks cannot be linked by the standard interblock relations.
Navigation between data blocks is handled by default functionality.	Navigation between data blocks of different forms is programmed by the designer (although mouse navigation to visible items can be automatic).

Form Module Hierarchy



ORACLE®

Form Module Hierarchy

The hierarchy of a form module is depicted by the graphics in the slide that show some of the objects of a module.

Triggers: Using triggers, you can write PL/SQL code to add functionality to your form. Triggers can be written at form, block, or item level in a Forms module.

Blocks: A form module is made up of one or more blocks. A data block is based on a database object, such as a table or a view. A data block can contain both data items, which represent columns in the base table, and control items. The dashed lines in the slide represent the relationships between database objects (table and column) and Forms objects (block and item.)

Canvases: To be visible to the end user, each item in a block must appear on a canvas. A frame can be created to arrange data block items on the canvas.

Program Units: User-named program units enable you to write additional PL/SQL code through procedures, functions, and packages.

Windows: To be visible to the end user, each canvas must appear in a window. A form module can have one or more canvases and windows.

Form Module Hierarchy (continued)

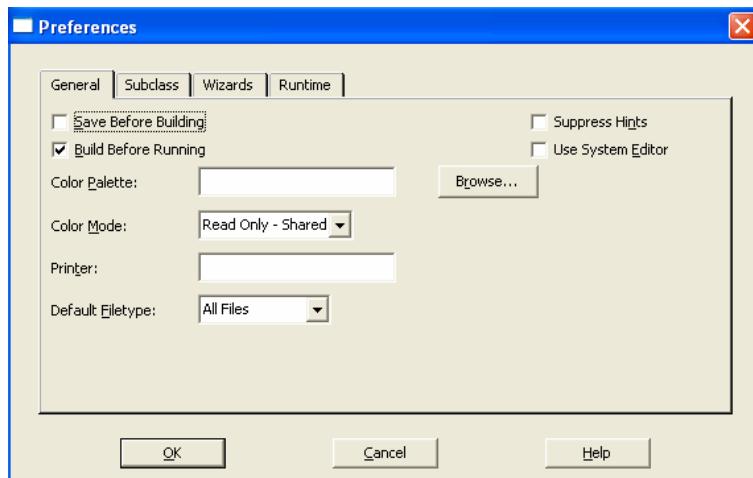
Object Hierarchy

You can create many types of objects in a form module. They are discussed in more detail in later lessons.

In the following table, note that some objects are associated, even though one might not be “owned” by the other. These associations are shown in the slide by the dotted lines connecting items with canvases and frames and also connecting canvases with windows.

Object	Description
Block	Logical section of a form; owned by the Forms module
Item	Member of a data block (items are functionally grouped into records that can represent a row in a table, but records are not Forms objects)
Canvas	The surface where visual objects are arranged; owned by the form module; can contain text and graphics—static information that the user cannot interact with—as well as items
Window	Produced to contain the views of canvases; owned by the form module
Frame	A graphic object that appears on a canvas, is owned by that canvas, and is used to arrange the items within a data block, although items can appear on a canvas without a frame object
User-named program unit	Named procedure, function, or package owned by the form module
Trigger	PL/SQL block executed on an event; depending on scope, can be owned by the form module, a data block, or an item
Other objects	Mainly owned by the form module itself; include alerts, attached libraries, editors, events, object groups, parameters, property classes, record groups, report objects, and visual attributes

Customizing Your Forms Builder Session



ORACLE®

3 - 24

Copyright © 2009, Oracle. All rights reserved.

Customizing Your Forms Builder Session

You can use preferences to customize some aspects of your Forms Builder session. There are four tabs in the Preferences dialog box. The screenshot shows the General tab and the preferences that you can set there.

To see a description of each preference, click Help in the Preferences dialog box or press the Help key (F1 for Windows).

In addition to session preferences, you can also set run-time preferences that apply to running your form from within the builder.

To modify preferences, perform the following steps:

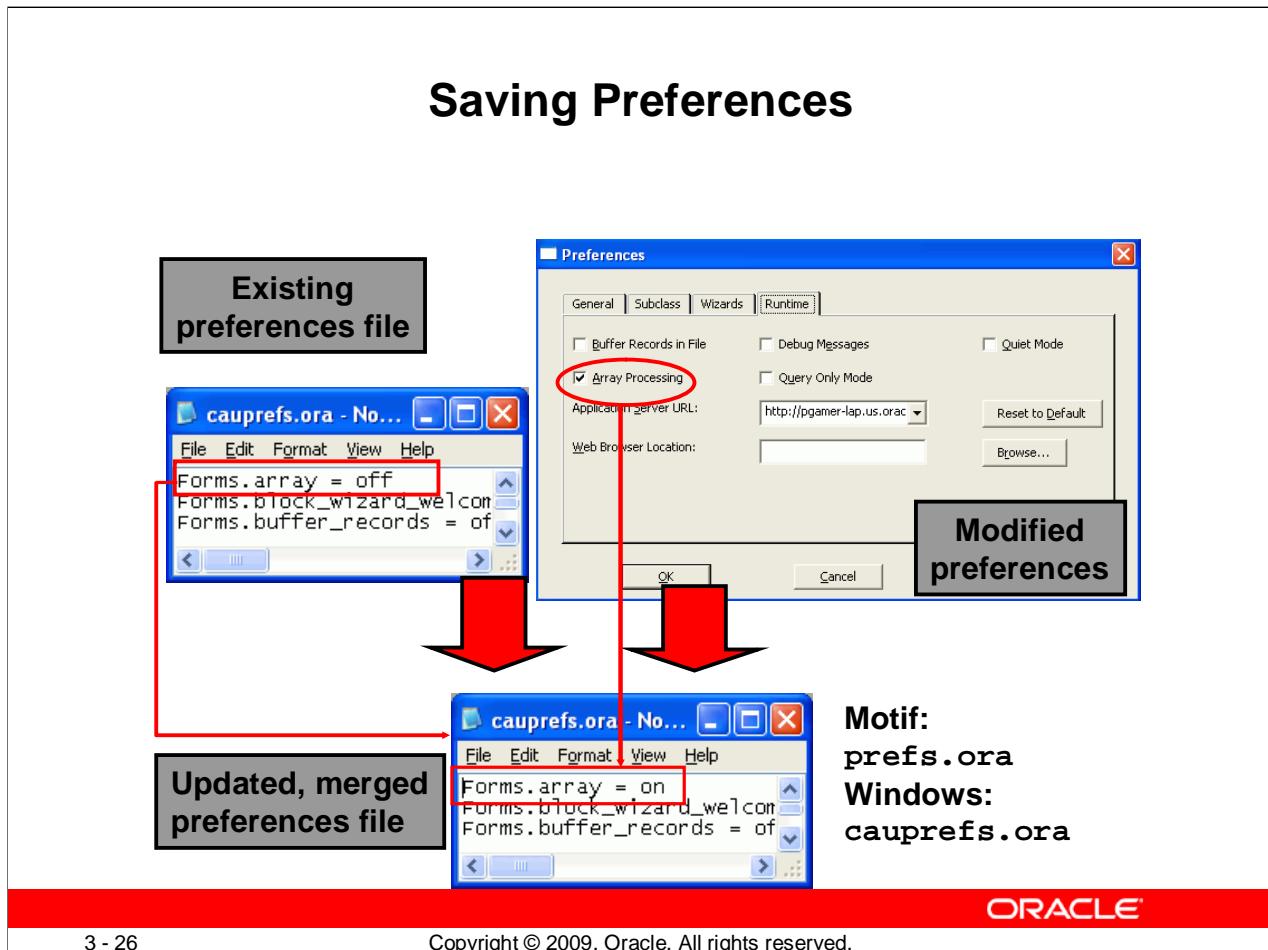
1. Select Edit > Preferences.
2. Specify any options that you require.
3. Click OK to save the changes or Cancel to cancel the changes.

Customizing Your Forms Builder Session (continued)

You can specify several related preferences on each of the Preference tabs. The following table describes one preference on each of the tabs:

Tab	Preference Name	Description
General	Build Before Running	Determines whether Forms Builder automatically compiles the active module when you run a form. This option enables you to avoid issuing separate Compile and Run commands each time you modify and run a form.
Subclass	Subclassing Path	Options for keeping or removing the subclassing path
Wizards	Welcome Dialog	Check box to suppress or display the first Welcome dialog box. There are several similar check boxes.
Runtime	Array Processing	Determines whether Forms Builder processes groups of records at a time, reducing network traffic and increasing performance

Saving Preferences



Saving Preferences

When you click OK in the Preferences dialog box, Oracle Forms Builder updates your current session with the changes.

When you exit the builder (File > Exit), Oracle Forms Builder writes the changes to a preference file for future sessions. The name of the preference file varies on different platforms.

Oracle Forms Developer and Oracle Reports Developer share the same preference file. If the preference file already exists, Oracle Forms Developer merges its changes with the existing file. This does not affect preferences for Reports.

Each option in the preference file is prefixed by the tool name to which it belongs.

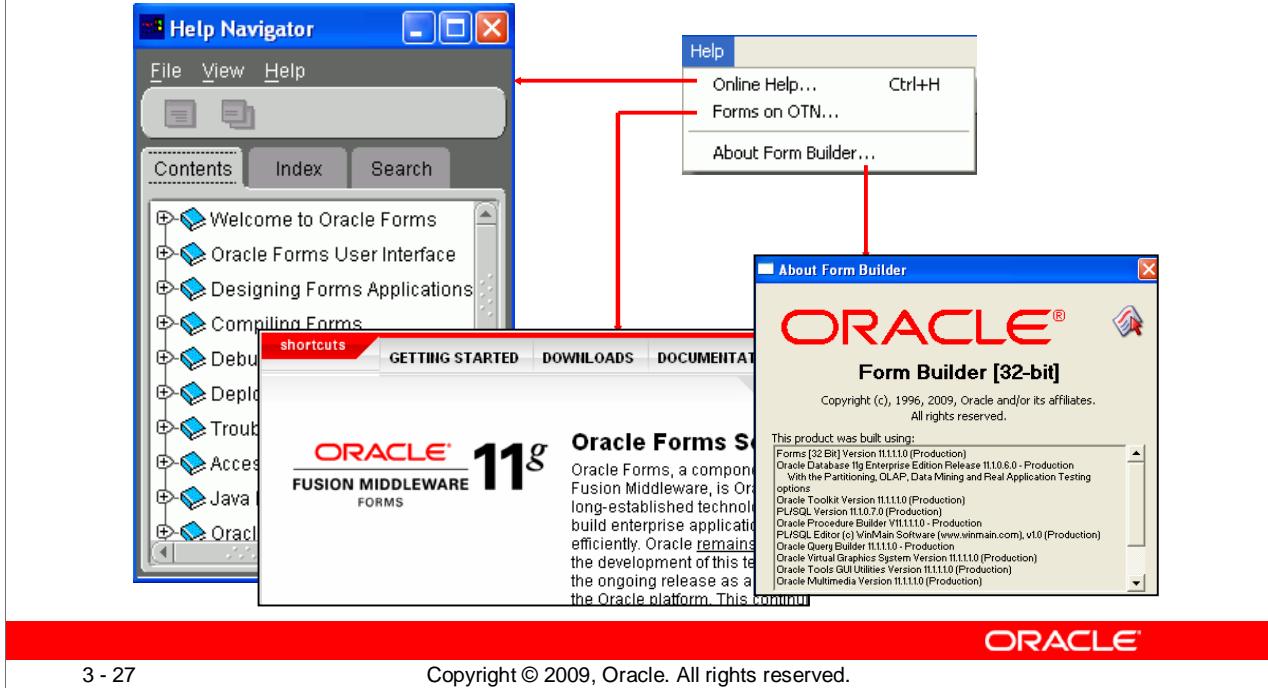
Example:

```
Forms.build_before_run = on
Forms.welcome_dialog = off
```

Oracle Forms Developer reads the preference file whenever you invoke Forms Builder. Oracle Reports Developer reads the preference file whenever you invoke Reports Builder.

Note: The preferences file is an editable text file. If possible, however, you should use the Preferences dialog box to alter the options. The graphics in the slide show how to change a preference in the dialog box and the effect it has on the cauprefs.ora file.

Using the Online Help System



Using the Online Help System

You can invoke online Help from the menu, as shown in the slide. The following table describes the Help menu options in Forms Builder:

Help Menu Option	Description
Online Help	Comprehensive online Help window with three tabs. The Contents tab, shown in the screenshot at the left of the slide, provides access to a variety of manuals and references. There are also Index and Search tabs. The Help key (F1 for Windows NT/95) displays context-sensitive online Help at any place in the Builder.
Forms on OTN	The latest product information on Oracle Technology Network, shown by the middle screenshot in the slide
About Form Builder	Shown at the bottom right of the slide, this is a separate window that shows product components and their version numbers. When you are connected to a database server, it also displays similar information for server-side product components.

You can also invoke context-sensitive online Help from Forms Builder by pressing the Help key (F1 on Windows).

Defining Forms Environment Variables for Design Time

Set on the Forms Builder machine:

- FORMS_BUILDER_CLASSPATH
- UI_ICON
- UI_ICON_EXTENSION
- FORMS_HIDE_OBR_PARAMS

Windows: Modify in Registry
(**REGEDIT.EXE** or **REGEDT32.EXE**)

ORACLE®

3 - 28

Copyright © 2009, Oracle. All rights reserved.

Defining Forms Environment Variables for Design Time

Design time environment variables help govern the behavior of Forms Builder. These environment variables must be set on the machine where Forms Builder is installed. For example, on Windows, you set them in the Windows Registry.

Java class files: Forms Builder needs access to certain Java classes for some of its features, such as Help, the debugger, and the Java importer. You set FORMS_BUILDER_CLASSPATH so that Forms Builder can find the Java classes that it needs during the development and testing of an application.

Icon files: Forms Builder enables you to create buttons with iconic images. You can use .ico, .gif, or .jpg images. You set UI_ICON to the directory path where these images are located. You set the UI_ICON_EXTENSION environment variable to indicate to Forms Builder which type of image to display on buttons. Valid values are ico (the default), gif, or jpg.

Although you can use .ico images to display on iconic buttons within Forms Builder, such images will not be displayed when running the form—use .gif or .jpg files for deployed icons.

Defining Forms Environment Variables for Design Time (continued)

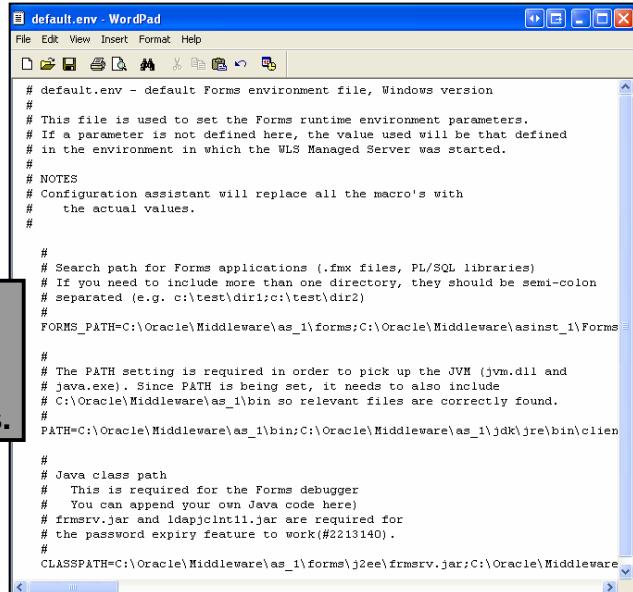
URL parameters: By default, when you run a form from Forms Builder, the parameters that are passed in the URL are hidden. For testing purposes, if you want to be able to see these parameters, such as the form name, you can set FORMS_HIDE_OBR_PARAMS to FALSE (the default is TRUE). OBR stands for one button run, a term used for running a form from within Forms Builder by clicking the Run Form button.

Defining Forms Environment Variables for Run Time

Set on the middle-tier machine (used at run time):

- FORMS_PATH
- ORACLE_PATH
- CLASSPATH

**For Forms deployment,
the settings in the
environment control file
override system settings.**



```
# default.env - default Forms environment file, Windows version
#
# This file is used to set the Forms runtime environment parameters.
# If a parameter is not defined here, the value used will be that defined
# in the environment in which the WLS Managed Server was started.
#
# NOTES
# Configuration assistant will replace all the macro's with
#   the actual values.
#
#
# Search path for Forms applications (.fmw files, PL/SQL libraries)
# If you need to include more than one directory, they should be semi-colon
# separated (e.g. c:\test\dir1;c:\test\dir2)
#
# FORMS_PATH=C:\Oracle\Middleware\as_1\forms:C:\Oracle\Middleware\asinst_1\Forms
#
# The PATH setting is required in order to pick up the JVM (jvm.dll and
# java.exe). Since PATH is being set, it needs to also include
# C:\Oracle\Middleware\as_1\bin so relevant files are correctly found.
#
# Java class path
#   This is required for the Forms debugger
#   You can append your own Java code here)
# frmssrv.jar and ldapjclient11.jar are required for
# the password expiry feature to work(#2213140).
#
# CLASSPATH=C:\Oracle\Middleware\as_1\forms\j2ee\frmssrv.jar:C:\Oracle\Middleware\
```

ORACLE®

3 - 30

Copyright © 2009, Oracle. All rights reserved.

Defining Forms Environment Variables for Run Time

Oracle Forms Developer uses many environment variables. These have default values, all of which you can modify in your own environment for different applications.

Setting Search Paths for Run Time

Forms uses some environment variables set on the middle-tier computer to search at run time for files such as forms, menus, and libraries. This enables you to build applications that are portable across platforms and directory structures by avoiding hard-coded paths in file references.

Forms searches the following paths in order until the required file is found:

- The current working directory
- Directories in FORMS_PATH
- Directories in ORACLE_PATH

Although you could set these variables at the machine level, such as in the Windows Registry, it is preferable to set them in the Forms environment file, which is shown in the slide. Settings in this file override system settings for running a Forms application.

Forms Files to Define Run-Time Environment Variables

- Environment control file:
 - default.env or
 - Other file specified in the Forms configuration file
- Forms configuration file:
 - formsweb.cfg or other
 - Used to specify:
 - System parameters, such as envFile
 - User parameters, such as form and user ID
 - Settings for the Java client
 - Other settings

ORACLE®

3 - 31

Copyright © 2009, Oracle. All rights reserved.

Forms Files to Define Run-Time Environment Variables

In a Windows 32-bit environment, you can use the Windows Registry to modify these paths, except for CLASSPATH, which is set in the System settings of the Control Panel. You can also override these settings at run time in the file that controls the Forms run-time environment, which is the default.env file unless a different file is specified. Using the Forms environment file makes it easier to deploy the application on any platform.

You can specify which file to use as the environment file in the Forms configuration file, formsweb.cfg, where you can set system parameters such as the name of the environment control file. You can also set parameters to control which form to run, the user ID, aspects of the Java client and the HTML file that contains the Java applet, and many other settings.

The default.env and formsweb.cfg files are located in the \\stage\\formsapp\\11.1.1\\formsapp\\config subdirectory of the domain home for the Forms-managed WebLogic server.

Setting Date Formats in the Run-Time Environment

- NLS_DATE_FORMAT
- FORMS_USER_DATE_FORMAT
- FORMS_USER_DATETIME_FORMAT
- FORMS_OUTPUT_DATETIME_FORMAT
- FORMS_OUTPUT_DATETIME_FORMAT
- FORMS_ERROR_DATE_FORMAT
- FORMS_ERROR_DATETIME_FORMAT

ORACLE®

3 - 32

Copyright © 2009, Oracle. All rights reserved.

Setting Date Formats in the Run-Time Environment

Dates in Oracle Forms Developer applications can be fetched from the database, entered by the end user, or defined in the application itself.

Date Format Masks

In the lesson titled “Working with Text Items” you learn how to specify a format mask for a date item in your form. In addition to the format masks a developer might explicitly define, Forms Builder uses several of its own internal masks. You can use the following environment variables to specify the values for these internal masks:

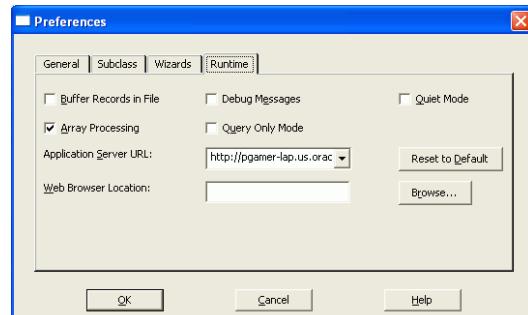
- **Database date format mask:** Each database session within a Forms application has a single database date format mask. A default value for this mask is established by the Oracle server’s initialization parameter. You can override this value in each new database session for a particular client by setting the client’s NLS_DATE_FORMAT environment variable.

Setting Date Formats in the Run-Time Environment (continued)

- **Input date format masks:** This mask (potentially, a set of masks) is used to convert a user-entered string into a native format date value. You can set the environment variables FORMS_USER_DATE_FORMAT or FORMS_USER_DATETIME_FORMAT to specify these format masks.
For example, you can set FORMS_USER_DATE_FORMAT to the value FXFMDD-MM-RRRR. This forces the user to enter values into date items (with no format mask) in the format exemplified by 30-06-97. The RRRR token enables years between 1950 and 2049 to be entered with the century omitted.
- **Output date format masks:** This mask converts dates displayed in output, such as lists of values: FORMS_OUTPUT_DATE_FORMAT or FORMS_OUTPUT_DATETIME_FORMAT.
- **Error date format masks:** This mask converts dates displayed in error messages: FORMS_ERROR_DATE_FORMAT or FORMS_ERROR_DATETIME_FORMAT.

Testing a Form with the Run Form Button

- With the Run Form menu command or button, you can:
 - Run a form from Forms Builder
 - Test the form in a three-tier environment
- The Run Form command takes its settings from Preferences:
 - Edit > Preferences
 - Runtime tab
 - Set Web Browser Location if desired.
 - Set Application Server URL to point to Forms Servlet:



<http://127.0.0.1:9001/formsfrm servlet>

ORACLE®

3 - 34

Copyright © 2009, Oracle. All rights reserved.

Testing a Form with the Run Form Button

The Run Form menu command or button enables you to run a form module in a Web browser from within Forms Builder. This makes it easy to test your application in a three-tier environment, with all components appearing and behaving as they would for a user of the application.

By default, the testing occurs on the development computer where Forms Builder is installed, using the default settings in the Forms Servlet configuration file, with no parameters passed on the URL.

The screenshots show from top to bottom:

- Choosing Run Form from the Program menu
- The Run Form and the Run Form Debug toolbar buttons
- The Runtime tab of the Preferences window, where you set the Application Server URL

Testing a Form with the Run Form Button (continued)

You can modify this three-tier testing environment in the Preferences dialog box by performing the following steps:

1. Click the Runtime tab.
2. Set the Application Server URL: This must be a URL pointing to the Forms Servlet on the middle tier. Note that it is typically on the same machine where you are running Forms Builder. You can also use the `config` parameter to specify a named configuration in the Forms Web configuration file (`formsweb.cfg` by default).

Example for the same machine with the WLS_FORMS managed WebLogic server running on the default port of 9001:

<http://127.0.0.1:9001/formsfrm servlet?config=myapp>

Note: You can enter the default setting in the Application Server URL field by clicking “Reset to Default.”

3. Set the Web Browser Location (needed only if you want to run in a different browser than the default for your machine).

Exiting the Session

The correct way to exit a Forms session is to exit the form (Action > Exit, or click Exit) and then close the browser. If you close the browser without first exiting the form, your session may hang.

You will notice this because you may not be able to recompile the same form, but will receive the error: FRM-30087: Unable to create form file. If this happens, you can open Task Manager and end the `frmweb` process manually.

Summary

In this lesson, you should have learned that:

- The Forms Developer executables are the Forms Builder and the Forms Compiler
- The Forms Developer module types are forms, menus, and libraries
- Forms Builder includes the Object Navigator, the Property Palette, the Layout Editor, and the PL/SQL Editor
- You can use the Object Navigator or the menu and its associated toolbar icons to navigate around the Forms Builder interface



Summary

- The Forms Developer executables are the Forms Builder and the Forms Compiler
- With Forms Builder, which is an Oracle Forms Developer component, you can develop form-based applications for presenting and manipulating data in a variety of ways. Forms Builder enables screen-based queries, inserts, updates, and deletes of data.
- Forms Builder provides powerful GUI and integration features.
- Applications consist of form modules, menu modules, and library documents.

Summary

- The main objects in a form module are blocks, items, and canvases
- The Edit > Preferences dialog box enables you to customize the Forms Builder session
- The Help menu enables you to use the online Help to look up topics, or you can invoke context-sensitive help
- You can set environment variables in the Forms environment file (for run time) or on the development machine (for design time)
- You can use the Run Form button to run a form from within Forms Builder

ORACLE®

3 - 37

Copyright © 2009, Oracle. All rights reserved.

Summary (continued)

- Form modules consist of logical data blocks. A data block is the logical owner of items. Items in one data block do not need to be physically grouped. Items in one data block can span several canvases.
- Environment variables govern the behavior of:
 - Running forms (set in the Forms environment file)
 - Forms Builder (set on the development machine, such as in the Windows Registry for Windows platforms)
- You can run a Forms application from within Forms Builder in order to test it in a browser. You specify the URL to use on the Runtime tab of the Preferences dialog box.

Practice 3: Overview

This practice covers the following topics:

- Setting Forms Builder preferences
- Becoming familiar with the Object Navigator
- Using the Layout Editor to modify the appearance of a form
- Setting environment variables so that the Layout Editor in Forms Builder displays .gif images on iconic buttons



Practice 3: Overview

In this practice, you become familiar with Oracle Forms Builder by performing the following tasks:

- Setting Forms Builder preferences
- Examining the Object Navigator in Forms Builder
- Using the Layout Editor in Forms Builder to modify the appearance of a form
- Setting environment variables so that images display on iconic buttons in the Layout Editor of Forms Builder.

You also run forms from within Forms Builder by using the Run Form button.

Creating a Basic Form Module



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

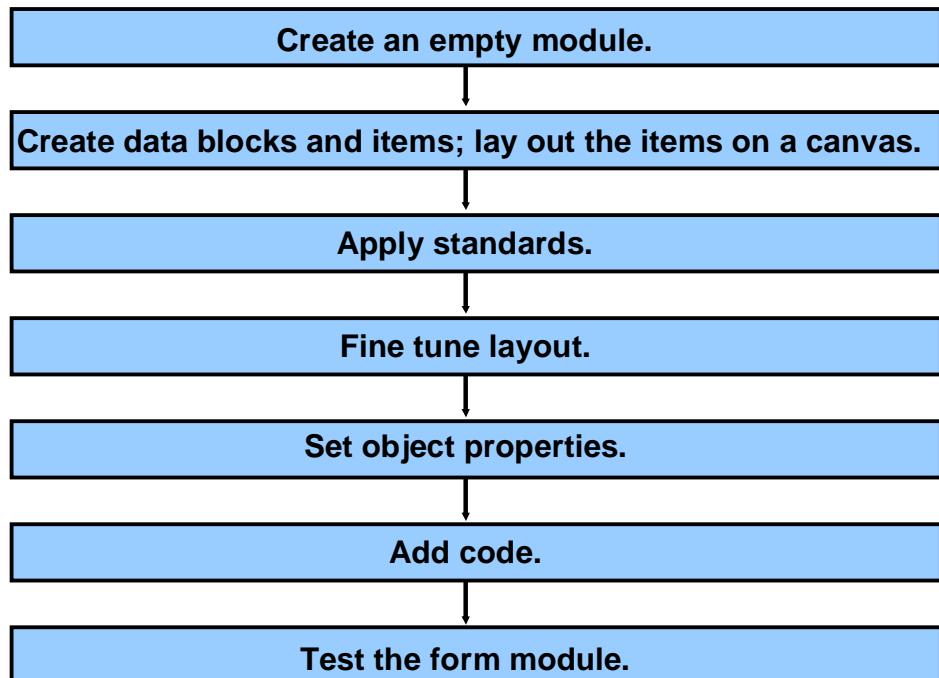
- Create a form module
- Create a data block
- Save and compile a form module
- Identify Forms file formats and their characteristics
- Describe how to deploy a form module
- Explain how to create documentation for a Forms application



Lesson Aim

Oracle Forms applications usually consist of one or more form modules. Each module comprises data blocks that are built using table specifications from the database. This lesson shows you how to create a basic form module and its data blocks. You also learn how to deploy a form module.

Designing Form Modules



ORACLE®

4 - 3

Copyright © 2009, Oracle. All rights reserved.

Designing Form Modules

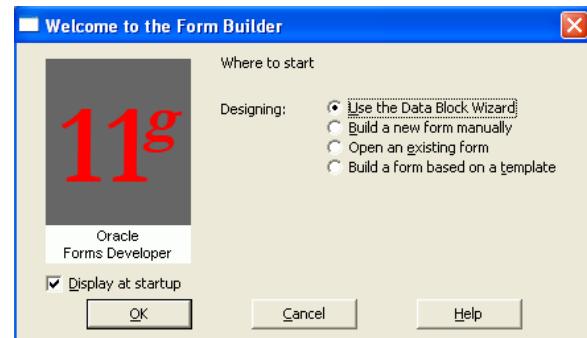
The following actions, along with the tools that are typically used to perform them, are part of creating a form module (although all actions may not be performed for every module):

Action	Forms Builder Tool
Create an empty module.	Object Navigator
Create data blocks and items and arrange the items on a canvas.	Data Block Wizard and Layout Wizard
Apply user interface standards to objects.	Object Library
Fine tune the layout.	Layout Wizard or Layout Editor
Set object properties.	Property Palette
Add code.	PL/SQL Editor
Test the form module.	Run Form button

Creating a New Form Module

Choose one of the following methods:

- Use wizards:
 - The Data Block Wizard
 - The Layout Wizard
- Build the module manually.
- Use the template form.

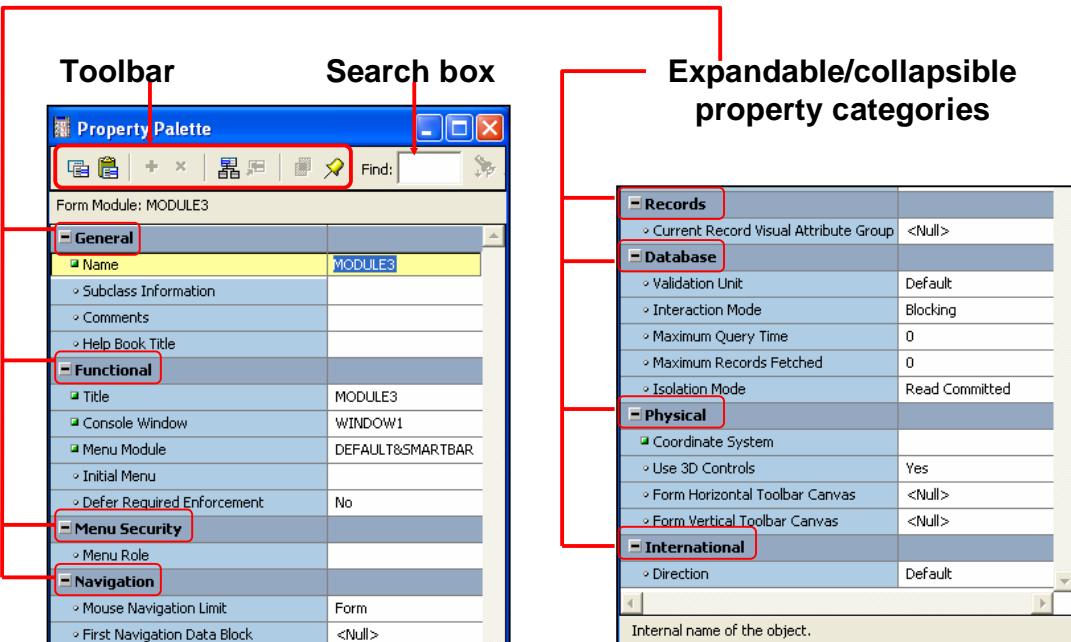


Creating a New Form Module

There are several ways to create a new form module.

- Invoke the Forms Builder component. This takes you to the Forms Builder Welcome page that is shown in the slide (unless you have changed the Preferences to not display it.) Now perform one of the following steps:
 - Select the “Use the Data Block Wizard” option, and follow the required data block creation steps. Then follow the Layout Wizard steps.
 - Select the Build a new form manually option. This takes you to the Forms Builder Object Navigator (automatically creating an empty form module).
 - Select the Build a form based on a template option and use a template form.
- If you are already in the Forms Builder component, you can create a new form module by performing one of the following steps:
 - Double-click the Forms node in the Object Navigator (only when no other form modules are available).
 - Select File > New > Form.
 - Select the Object Navigator node for Forms, and then click the Create icon.

Setting Form Module Properties



Setting Form Module Properties

Each form module consists of several objects. Objects within a form, and the module itself, have properties that define their behavior. You can see the properties of an object and their values in the Property Palette of the object. The screenshots show all the properties of a form module as displayed in the Property Palette.

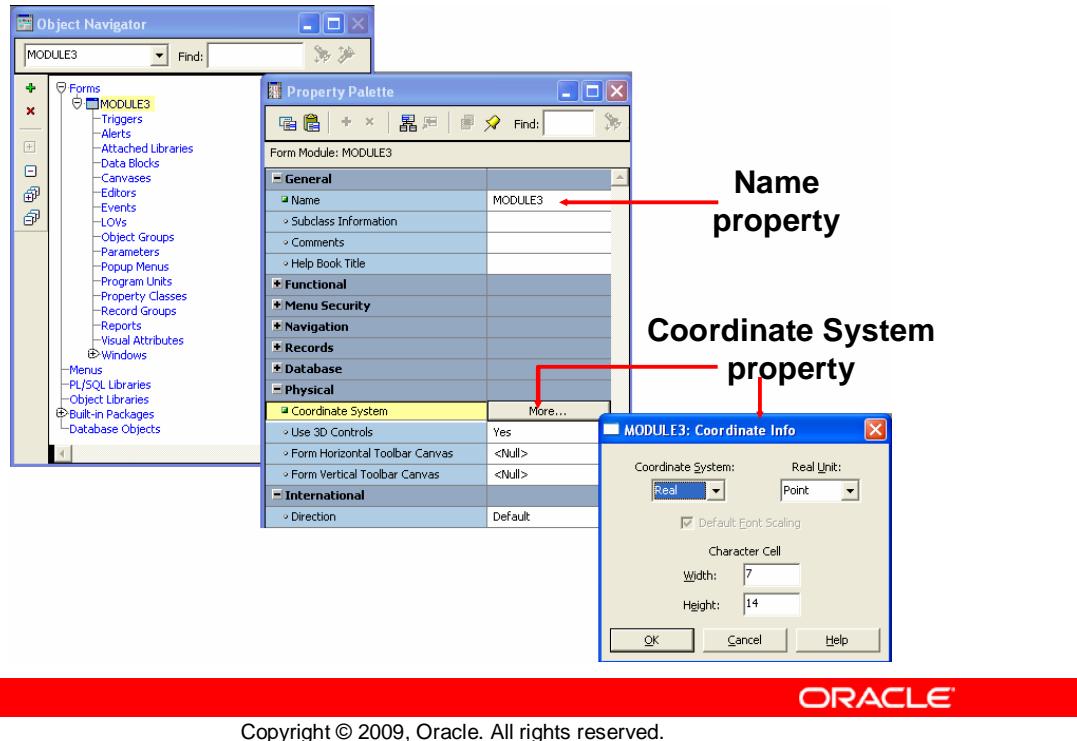
To open the Property Palette of an object, perform one of the following steps:

- Double-click the object's icon in the Object Navigator.
- Double-click the object in the Layout Editor
- Select the object in the Object Navigator and select Tools > Property Palette.
- Right-click the object in the Object Navigator or in the Layout Editor and select Property Palette from the context menu.

A brief description of a selected property appears at the bottom of the palette. To obtain more detailed online Help for any of the properties, select the property and use the Help key (F1), to display a description of that property.

The Property Palette features a toolbar at the top, along with a search box enabling you to easily locate any property. The properties are divided into categories that you can expand or collapse.

Changing the Form Module Name and Coordinate System



Changing the Form Module Name and Coordinate System

You can define the properties of the form module when you first create it or modify the properties later. The properties affect the general behavior of the form and the objects within it. Properties for a form module include the following:

Property	Description
Name	Specifies the internal name of the form module, as it appears in the Object Navigator
Coordinate System	Defines the units used to measure objects in the form and their positions

The screenshots show the module name selected in the Object Navigator and the Property Palette synchronized to display the properties of that selected module. The Property Palette shows the Name property, where you can change the name of the form module. It also shows the Coordinate System property with a More button displayed in the value space. Clicking the button invokes the Coordinate Info dialog box, where you can change the Coordinate System properties.

Changing the Form Module Name and Coordinate System (continued)

Setting the Form Module Name

Forms Builder assigns the name MODULEX to a new form module, where X is the next number available for module names. This name is displayed in the Object Navigator and in the Property Palette.

You should change the default name to a meaningful name in either of the following places:

- In the Object Navigator:
 - Click the form module name.
 - Change the default name as desired and press Enter.
- In the Property Palette (shown in the screenshot)

Note: Follow Oracle naming rules. Do not give two objects of the same type the same name.

The name cannot include Oracle or Forms Builder–reserved words.

Choosing a Unit for the Coordinate System

When you click More in the Property Palette window with the Coordinate System property selected, the Coordinate Info dialog box appears.

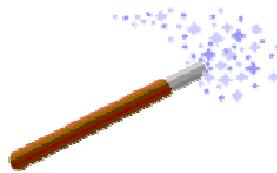
The Coordinate System unit for a form can be one of the following:

- **Real:**
 - Unit can be pixel, centimeter, inch, point, or decipoint.
 - Real units are suitable for GUI applications and enable flexibility and fine alignment when adjusting object positions and sizes.
- **Character:** Units are character cells (default size taken from the default font settings).

The default unit is point (Real). This means that object positions and sizes within the form are measured by this unit. Points provide fine alignment and consistency across different platforms and video devices.

Creating a New Data Block

- Using Forms Builder wizards:
 - Data Block Wizard: Create a data block with associated data source quickly and easily.
 - Layout Wizard: Lay out data block contents for visual presentation.
- Manually
- May require database connection



ORACLE®

4 - 8

Copyright © 2009, Oracle. All rights reserved.

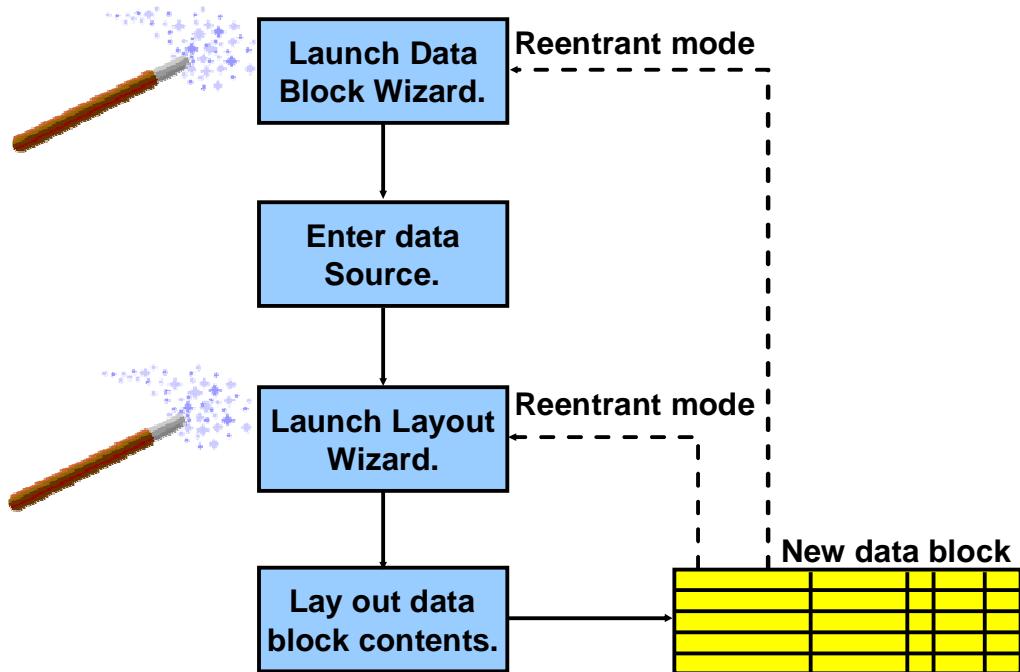
Creating a New Data Block

A form module consists of one or more data blocks and control blocks. Now that you know how to create a new form module, you need to create new data blocks within it.

Block creation involves creating the data block and then laying out its contents for visual presentation. You can create a data block manually or by using the Forms Builder wizards. In this lesson, you learn how to create a new data block based on a database table, using the Data Block Wizard and the Layout Wizard. The slide shows a magic wand, symbolizing a wizard.

Note: Recall that a data block can be based on a table or view, a stored procedure, a FROM clause query, or a transactional trigger. In this course, you use database tables as the source. This requires that you first connect to the database, which you are prompted to do if you use the Data Block Wizard without first connecting. Connecting to the database was covered in the previous lesson.

Using the Wizards to Create a Data Block



Using the Wizards to Create a Data Block

The slide shows the process of creating a new data block by launching the Data Block Wizard, entering the data source, launching the Layout Wizard, and laying out the contents of the data block. You can invoke either wizard in reentrant mode.

Data Block Wizard

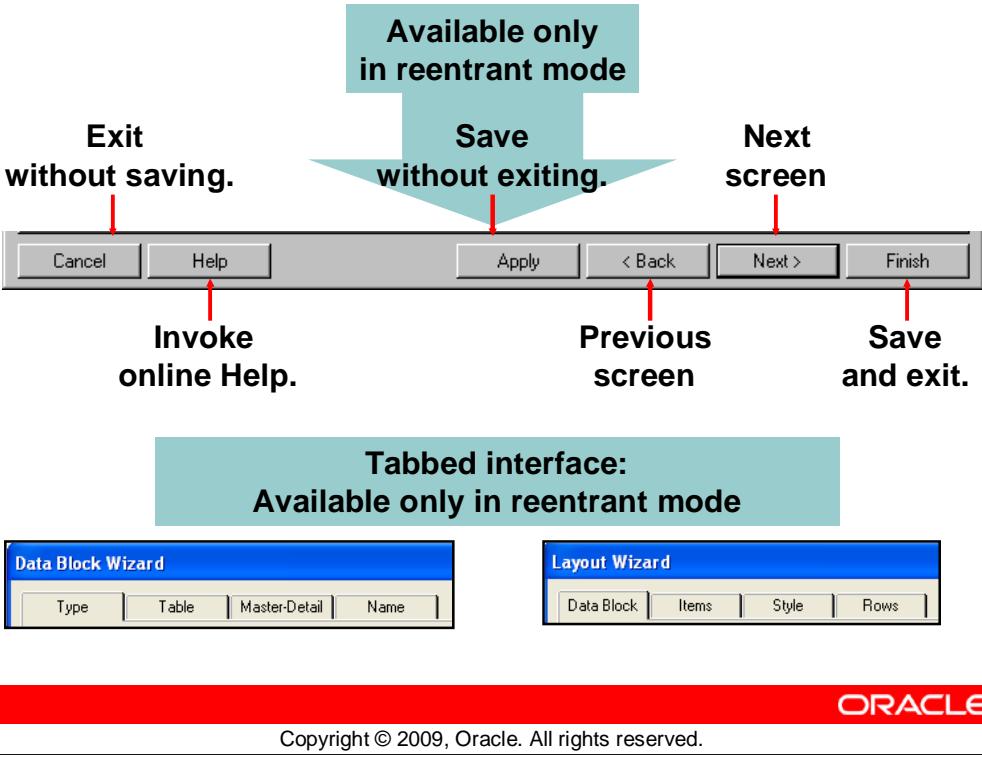
The Data Block Wizard enables you to create (or modify) data blocks quickly and easily for use in your application. The wizard can automatically generate code to enforce integrity constraints in the database.

Layout Wizard

Although the Data Block Wizard enables you to create a new data block easily with its associated data sources, it does not deal with the visual presentation of objects included in the data block. After you create the data block, you need to lay out its contents for user interaction. To accomplish this task quickly and easily, use the Layout Wizard.

Note: The wizards are not the only way to perform a task such as building a data block, but they are usually the simplest. You can build a block manually instead of using the wizards.

Navigating the Wizards



Navigating the Wizards

The Data Block Wizard and the Layout Wizard provide buttons as shown in the slide:

Button	Description
Cancel	Cancels any changes and exits the wizard
Help	Displays online Help text for the current wizard page
Back	Navigates to the previous page in the wizard
Next	Navigates to the next page in the wizard
Apply	Applies your changes without exiting the wizard (available only in reentrant mode)
Finish	Saves any changes and exits the wizard

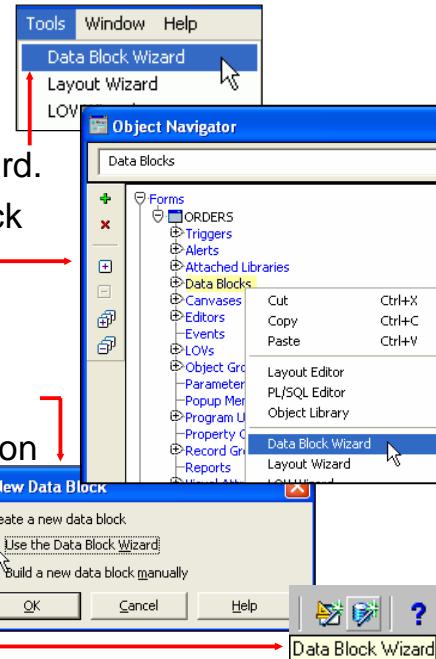
If you click Next or Back before entering all necessary information for a particular wizard page, the wizard prevents you from navigating to another page. Similarly, if you have not entered all necessary information when you click Apply or Finish, the wizard automatically takes you to the page where you can finish entering the required information.

In reentrant mode, which enables you to modify existing blocks or layouts, the wizards have a tabbed interface that enables you to quickly navigate to the section you want to modify. You learn more about modifying data blocks and layouts in the lesson titled “Creating a Master-Detail Form.”

Launching the Data Block Wizard

In Forms Builder, do one of the following:

- Select Tools > Data Block Wizard.
- Right-click and select Data Block Wizard.
- Select the Data Blocks node and click Create icon; select “Use the Data Block Wizard.”
- Use the Data Block Wizard button on the toolbar.



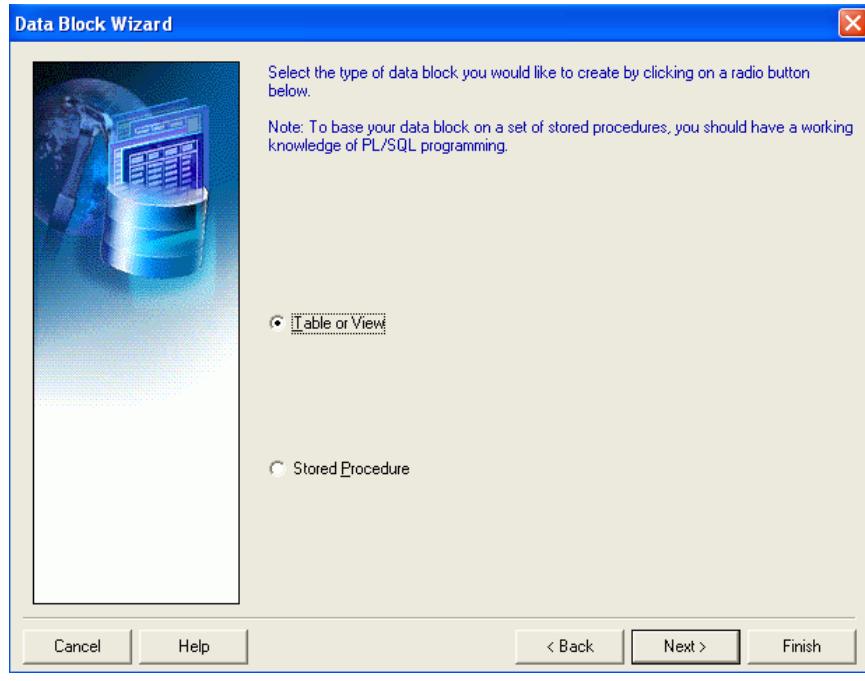
Launching the Data Block Wizard

To launch the Data Block Wizard to create a new data block, perform one of the following:

- In the Forms Builder, do one of the following:
 - Select Tools > Data Block Wizard from the Forms Builder default menu system.
 - Right-click and select Data Block Wizard.
 - In the Object Navigator, select the Data Blocks node, then click the Create icon. In the New Data Block dialog box, select the “Use the Data Block Wizard” option.
 - Click Data Block Wizard on the toolbar.
- If you are not already in Forms Builder, launch Forms Builder and select the “Use the Data Block Wizard” option on the Forms Builder Welcome page.

The screenshots show the Tools menu, the context menu, the Data Block Wizard option, and the toolbar.

Data Block Wizard: Type Page



ORACLE®

4 - 12

Copyright © 2009, Oracle. All rights reserved.

Data Block Wizard: Type Page

Use the Data Block Wizard to create a new data block with its associated data sources. The Data Block Wizard consists of several pages. To create a new data block, you must interact with each page.

Welcome Page

Click Next to continue.

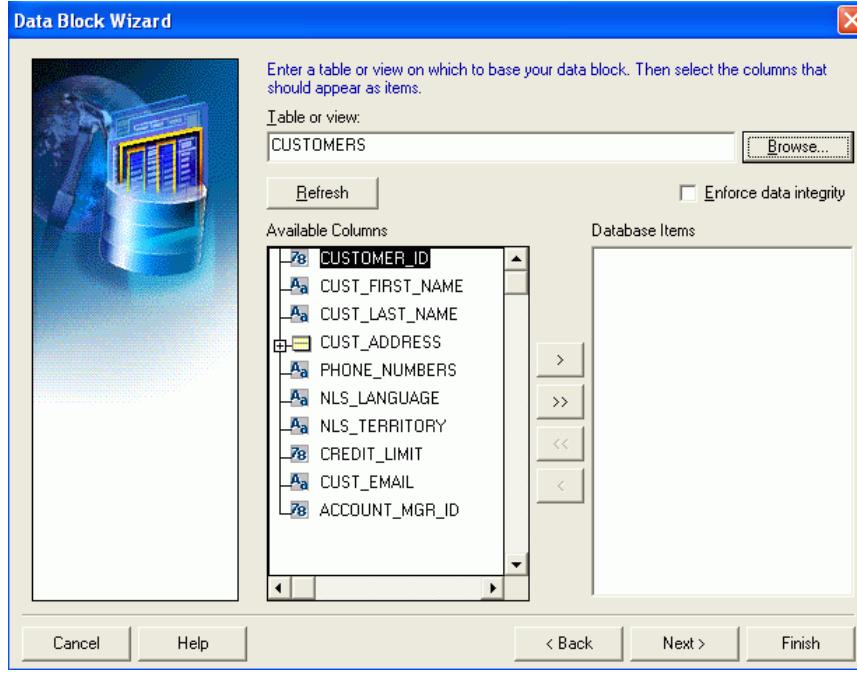
Type Page (shown in the screenshot)

Choose between one of two data source types:

- Table or View
- Stored Procedure

Select the Table or View (default) option.

Data Block Wizard: Table Page



ORACLE®

4 - 13

Copyright © 2009, Oracle. All rights reserved.

Data Block Wizard: Table Page

On the Table page (shown in the screenshot):

1. Enter the table or view name for the data source name, or click Browse and select a name from a dialog box.
2. Click Refresh to display a list of columns in the selected table or view. If you are not connected to the database, the Connect box is displayed.
3. Select the columns you want to include in the data block. (To select more than one column, press and hold Ctrl and then select the columns.)
4. Click the double-right arrow or the double-left arrow to include or exclude all columns, or click the right arrow or the left arrow to include or exclude selected columns only.
5. Select the Enforce data integrity check box if you want the wizard to enforce the database integrity constraints.

Note: If there is at least one other existing block in the current module, the next page that displays is the Master-Detail page, where you can associate the new data block with other master data blocks. This page is discussed in the lesson titled “Creating a Master-Detail Form.”

Data Block Wizard: Finish Page



ORACLE®

4 - 14

Copyright © 2009, Oracle. All rights reserved.

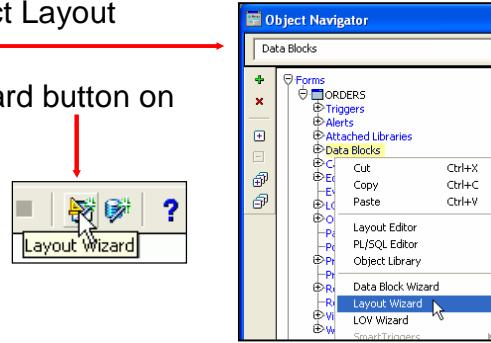
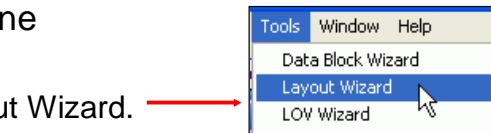
Data Block Wizard: Finish Page

On the Finish Page (shown in the screenshot) select the “Create the data block, then call the Layout Wizard” option. Click Finish to create the new data block and immediately invoke the Layout Wizard.

Note: You have the option of exiting the Data Block Wizard at this stage, without immediately invoking the Layout Wizard. If you do so, you can either lay out the data block manually or invoke the Layout Wizard at a later time to lay out the items of a data block. To invoke the Layout Wizard at a later time, select the data block in the Object Navigator, and then select Tools > Layout Wizard.

Launching the Layout Wizard

- Launch automatically from the Data Block Wizard.
or
- In Forms Builder, do one of the following:
 - Select Tools > Layout Wizard.
 - Right-click and select Layout Wizard.
 - Use the Layout Wizard button on the toolbar.



ORACLE®

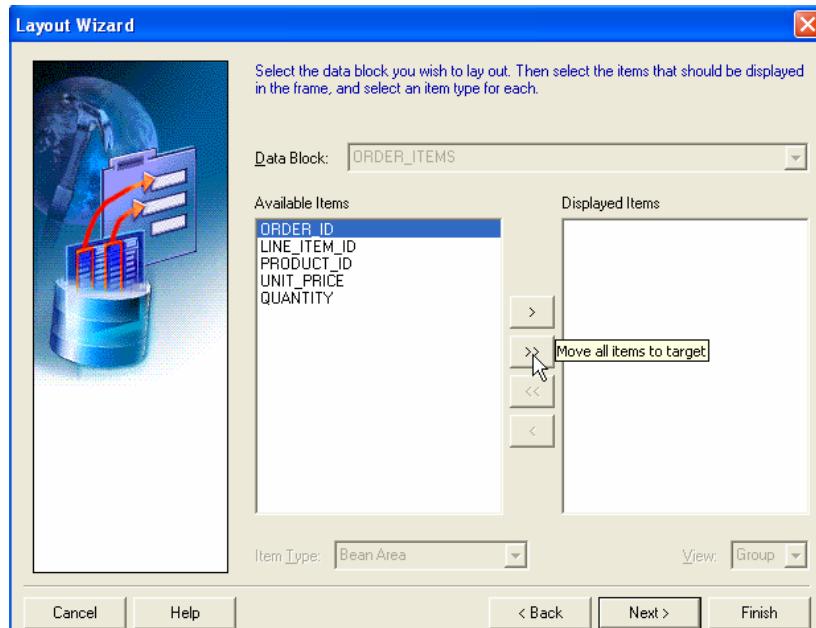
Launching the Layout Wizard

To launch the Layout Wizard to create a new layout, perform one of the following steps:

- Launch automatically from Data Block Wizard.
- In the Forms Builder, do one of the following:
 - Select Tools > Layout Wizard from the Forms Builder default menu system.
 - Right-click and select the Layout Wizard option.
 - Click Layout Wizard on the toolbar.

The screenshot at the top of the slide shows the last page of the Data Block Wizard that can optionally launch the Layout Wizard. The other screenshots show invoking the Layout Wizard from Tools menu, the context menu, and the toolbar.

Using the Layout Wizard: Canvas and Data Block Pages



ORACLE®

4 - 16

Copyright © 2009, Oracle. All rights reserved.

Using the Layout Wizard: Canvas and Data Block Pages

Use the Layout Wizard to lay out the data block items for visual presentation quickly and easily. The Layout Wizard consists of several pages. You must interact with each page.

Canvas Page

1. Select New Canvas from the Canvas pop-up list to get a new canvas on which to display the data block items.
2. Select Content as the canvas type in the Type pop-up list.

Data Block Page (shown in the screenshot)

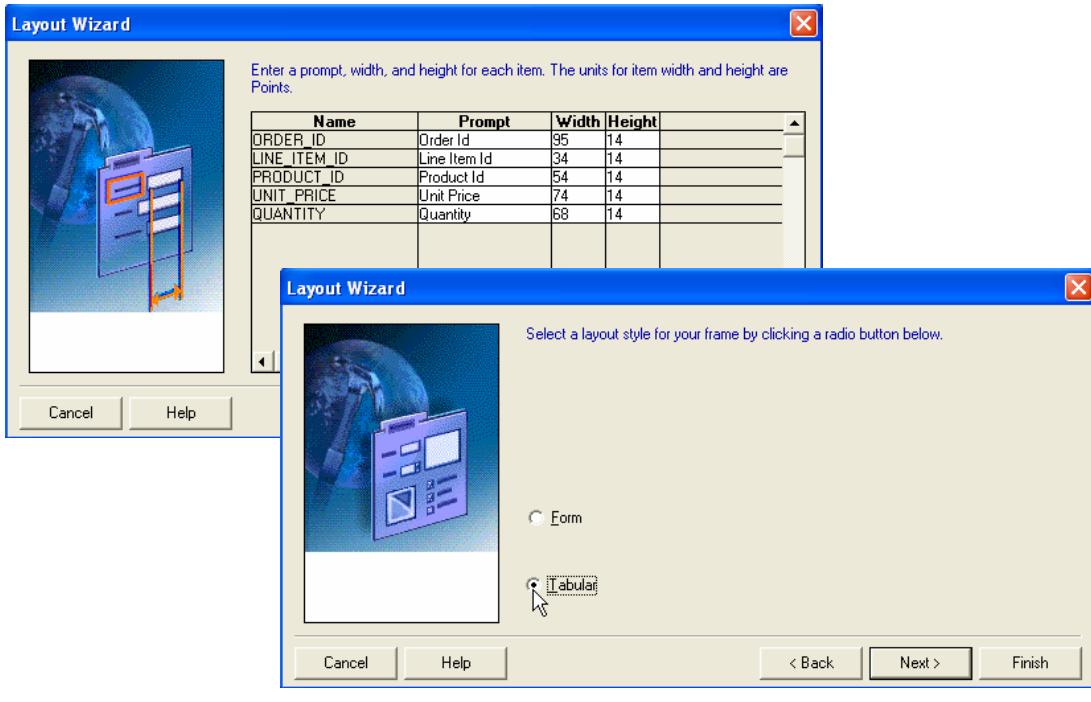
1. Select the items that you want to display in the data block frame. (To select more than one column, press and hold Ctrl and then select the columns.)
2. Click the double-right arrow or double-left arrow to include or exclude all items, or click the right arrow or the left arrow to include or exclude selected items only. You can also drag selected items from one list to another.

Note: To lay out the items in a particular sequence, drag the items into that sequence.

3. You can use the Item Type pop-up list to select a type for each item. The default type is Text for each item.

Note: An item type can also be changed later to something else, such as a pop-up list or a radio group.

Using the Layout Wizard: Items and Style Pages



ORACLE®

4 - 17

Copyright © 2009, Oracle. All rights reserved.

Using the Layout Wizard: Items and Style Pages

Items Page

Specify prompt text and display width and height for each display item.

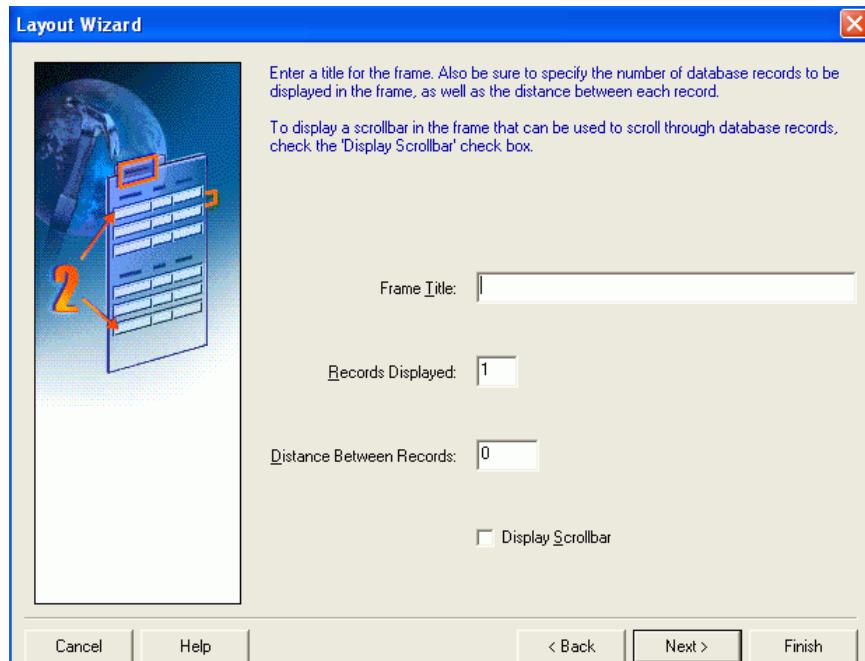
Style Page

Select a layout style for your frame. Your options are:

- Form (usually used to create single-record data blocks)
- Tabular (usually used to create multirecord data blocks)

The slide shows both the Items and the Style pages.

Using the Layout Wizard: Records and Finish Pages



ORACLE®

4 - 18

Copyright © 2009, Oracle. All rights reserved.

Using the Layout Wizard: Records and Finish Pages

Records Page (shown in the screenshot)

1. Enter a title in the Frame Title field.
2. Enter the number of records that you want to display at run time in the Records Displayed field.
3. Enter the physical distance (in the coordinate system unit of the form) between records if you are displaying more than one record at a time.
4. You can select the Display Scrollbar check box to display a scroll bar next to the frame (common for multirecord data blocks).

Finish Page

Click Finish to create a new frame and lay out the selected items for the new data block. The Layout Wizard steps are complete.

Note: After you complete the Layout Wizard steps, you can view the layout in the Layout Editor, where you can customize or modify the layout if necessary.

Data Block Functionality

After you create a data block with the wizards, Forms Builder automatically creates:

- A form module with database functionality including query, insert, update, and delete
- A frame object
- Items in the data block
- A prompt for each item
- Triggers needed to enforce database constraints if the Enforce data integrity check box is selected

ORACLE®

4 - 19

Copyright © 2009, Oracle. All rights reserved.

Data Block Functionality

After you create a new data block by using the wizards, Forms Builder automatically creates the following objects for you:

- A new form module with a default menu (Basic database functionality such as querying, inserting, updating, and deleting is automatically available on the items in the base-table block when you run the new form.)
- The new data block is created with default property values. These values can be modified to change the behavior of the form.
- A frame object to arrange the items within the new data block
- An item for each database table column included in the data block (Each item is assigned default property values to match the underlying column specifications.)
- A prompt for each item in the data block (The default prompt is the name of the column.)

In addition, Forms Builder may create triggers to validate user input if you select the Enforce data integrity check box on the Table page of the Data Block Wizard.

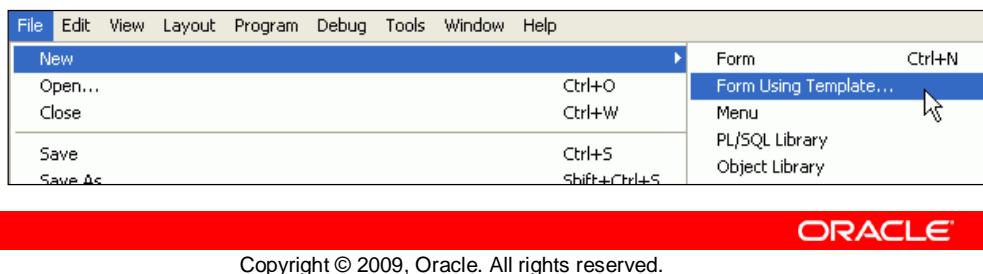
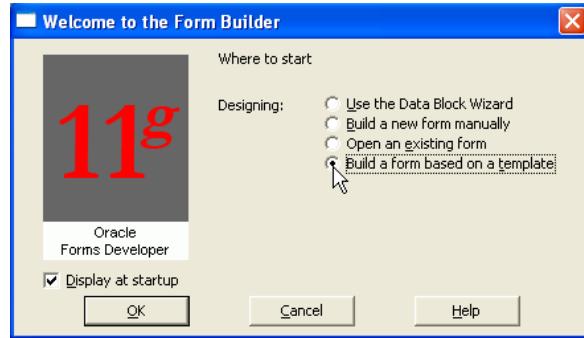
Template Forms

To use a form template:

- Select the template option from the Welcome dialog box

Or

- Select File > New > Form Using Template



Template Forms

You can create a new form based on standard template forms, so that you can provide other team members with a default starting point. Templates typically include generic objects, such as graphics, toolbars, and program units. You can define standard window layouts, standard toolbars, and other common objects that you want to include in new forms.

Creating a Form Based on a Template

To create a form based on a template, perform one of the following:

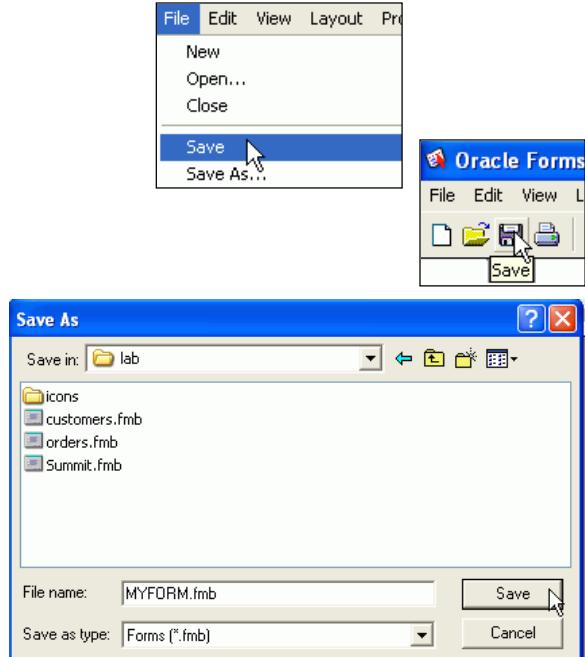
- Start Forms Builder. In the “Welcome to the Form Builder” dialog box, select the Build a form based on a template option, and then click OK.
- Select File > New > Form Using Template from the menu.

After choosing to use a template, Forms Builder presents you with a file dialog where you can browse to the form you want to use as a template.

Saving a Form Module

To save the form module:

- Select File > Save
 - or
 - Click the Save icon
- Enter a file name
- Navigate to the desired location
- Click Save



Saving a Form Module

To save the form module definition, perform one of the following steps:

- Select File > Save.
- Click the Save icon on the toolbar.

Both of these options display the Save As dialog box for the initial save. In the dialog box, do the following:

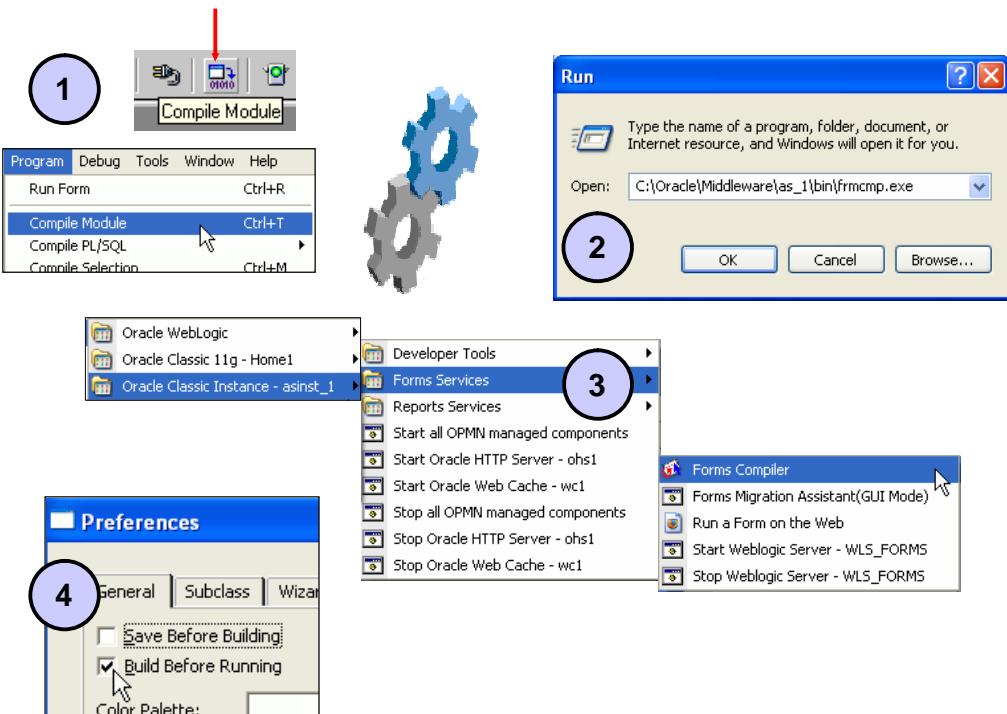
1. Enter a file name.
2. Navigate to the directory where you want to save the file.
3. Click Save.

The screenshots show the menu option, the toolbar, and the Save As dialog box.

Note

- When you save a form, a .fmb file is produced. This saved definition of a form in the file system is not executable, and can be opened only by Forms Builder.
- When you work with more than one module at a time, Forms Builder separately keeps track of the changes that you make to each module. When you execute a Save command, only the current module is saved.

Compiling a Form Module



Compiling a Form Module

Before you can run a form, you must compile an executable (.fmx) file from the design (.fmb) file that you created in the Forms Builder. Compiling a form (or menu) module creates the needed executable file. There are several ways to compile a form:

Action	Type of Compilation
1. With module open in Forms Builder, select Program > Compile Module (or click the Compile Module icon).	Explicit
2. Launch the Forms Compiler component from the command line or the Windows Run dialog box (frmcmp.exe).	Explicit
3. Launch the Forms Compiler component from the Windows Start menu.	Explicit
4. Set the Build Before Running preference.	Implicit

Each of these options is depicted in the screenshots in the slide.

Note: Compiling and saving are two independent tasks. Performing one does not automatically accomplish the other.

Module Types and Storage Formats

Form module			
Menu module			
PL/SQL Library			
Object Library			

ORACLE®

Module Types and Storage Formats

When you create form modules, menu modules, and library documents in the Forms Builder, they are stored in source files (.fmb, .mmb, and .pll) that have a binary format and are portable across platforms. The executable application files (.fmx, .mmx, and .plx) are also in a binary format; however, they are not portable across platforms.

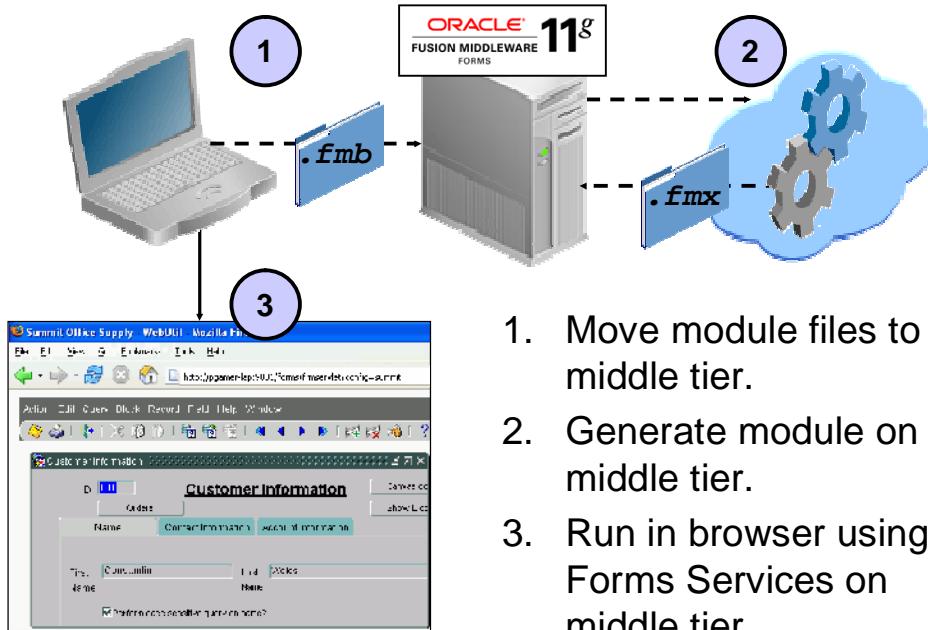
The graphics in the slide show the various file types. The first column of graphics shows the portable binary files, the second column of graphics shows the executables, and the third column of graphics shows the text files that you can create for form modules, menus, PL/SQL libraries, and object libraries.

Module Types and Storage Formats (continued)

Module/ Document	Extension	Storage Format	Portable
Form	.fmb	Form module binary	Yes
	.fmx	Form module executable; executable	No
	.fmt	Form module text	Yes
Menu	.mmb	Menu module binary	Yes
	.mmx	Menu module executable; executable	No
	.mmt	Menu module text	Yes
PL/SQL Library	.pll	PL/SQL Library document binary	Yes
	.plx	PL/SQL Library document executable (no source)	No
	.pld	PL/SQL Library document text	Yes
Object Library	.olb	Object Library module binary	Yes
	.olt	Object Library module text	Yes

Note: .pll is portable but requires recompilation because it contains both source and compiled code.

Deploying a Form Module



Deploying a Form Module

Although you may test a form on your client machine, for production applications you usually deploy the module to a middle-tier machine. This machine may be running on a different platform; if so, you need to recompile the module after you transfer it to the middle tier.

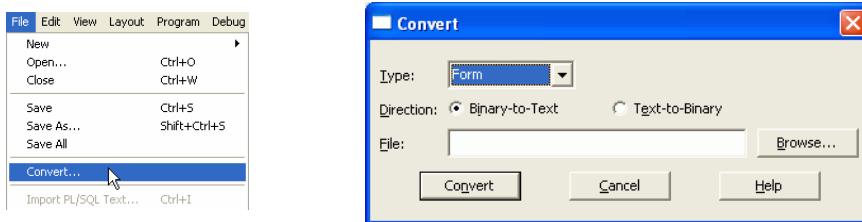
You can use an FTP utility to move the .fmb and other needed files to the middle-tier machine, into a directory specified in FORMS_PATH, as shown in the slide. If the platform is the same as your development platform, you can move the .fmx and other executable files there as well. If it is a different platform, you can invoke the Forms Compiler on the middle tier to recompile the module files, as shown in the slide.

After the executables have been placed on the middle tier, you can invoke the application in a browser, using a URL that points to the Forms Servlet on the middle-tier Web server.

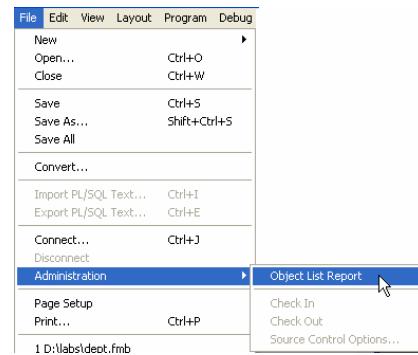
Note: FORMS_PATH and other environment variables are set for run time on the middle tier as described in the lesson titled “Working in the Forms Environment.”

Producing Text Files and Documentation

- Convert a binary file to a text file.



- Create an ASCII file for a form module.



ORACLE®

4 - 26

Copyright © 2009, Oracle. All rights reserved.

Producing Text Files and Documentation

The files normally produced by saving and generating modules are in binary format. To convert a binary file to text, perform the following:

- Select File > Convert, as shown in the first screenshot in the slide.
This opens the Convert dialog box.
- In the Convert dialog box:
 - Select the type of module (Form, Menu, PL/SQL Libraries, and Object Libraries), the file to convert, and the direction (Binary-to-Text).
 - Click Convert. This produces a text file for the module with the name <module>.fmt.

To produce documentation for your module, perform the following:

- Select the module to be documented in the Object Navigator.
- Select File > Administration > Object List Report. This produces an ASCII file with the name <module>.txt.

You can also produce documentation in other ways not covered in this course:

- Use Forms API to produce custom documentation of the module.
- Convert the module to a .xml file with a separate utility included with Forms.

Summary

In this lesson, you should have learned that:

- To create a form module, you create an empty module, then add data blocks and other elements
- You can create a data block manually or with the Data Block Wizard and the Layout Wizard
- You can save and compile a form module using the File and Program menus or from the toolbar
- You can store form, menu, and library modules in text format (useful for documentation), in a portable binary format, or a nonportable binary executable format
- To deploy a form module, you move it to the application server machine and also may need to generate it

ORACLE®

4 - 27

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you should have learned about:

- Building a new form module by using the following methods:
 - Forms Builder wizards
 - Manually
 - Template form
- Using the Data Block Wizard to create a new data block with its associated data sources quickly and easily
- Using the Layout Wizard to quickly lay out the new data block contents for user interaction
- Saving the form module to preserve its definition; compiling it to get an executable file; running the form module to test it
- Using several module types and storage formats that are available for form modules, menu modules, PL/SQL Library documents, and Object Library modules, including a text format that can be used for documentation

Practice 4: Overview

This practice covers the following topics:

- Creating a new form module
- Creating a data block by using Forms Builder wizards
- Saving and running the form module



Practice 4: Overview

In this practice, you create a new form module. You create a single-block form that displays a single record.

Creating a Master-Detail Form

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

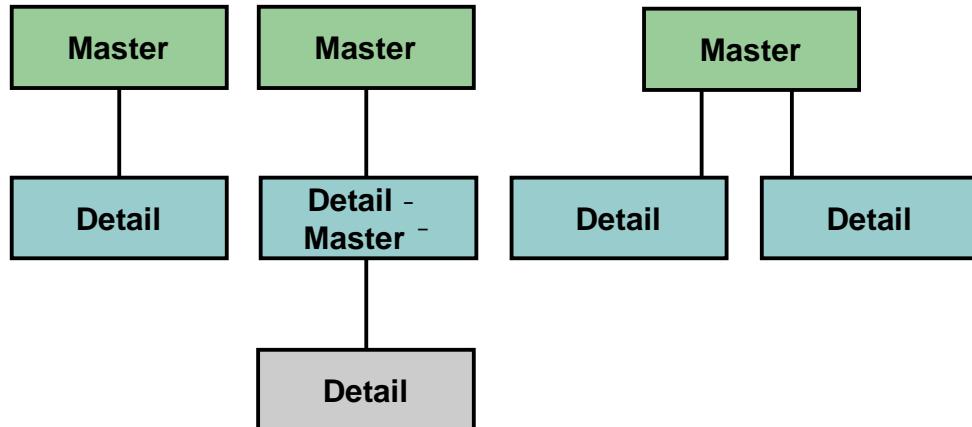
- Create data blocks with relationships
- Modify a data block
- Modify the layout of a data block
- Run a master-detail form



Lesson Aim

A common type of application is to show a record or records from one table along with associated records from another table. Forms Developer gives you the ability to quickly define additional data blocks and to define relations between the new blocks and any existing blocks in the form. This lesson shows you how to create a master-detail form and how to modify a data block and its layout.

Creating Data Blocks with Relationships



ORACLE®

5 - 3

Copyright © 2009, Oracle. All rights reserved.

Creating Data Blocks with Relationships

A form module can contain one or more data blocks. Each data block can stand alone or be related to another data block. The slide shows the following:

- A master block with one detail block
- A master block with one detail block that is the master for another detail block
- A master block with two detail blocks

Master-Detail Relationship

A master-detail relationship is an association between two data blocks that reflect a primary key/foreign key relationship between the database tables on which the two data blocks are based. The master data block is based on the table with the primary key, and the detail data block is based on the table with the foreign key. A master-detail relationship equates to the one-to-many relationship in the entity relationship diagram.

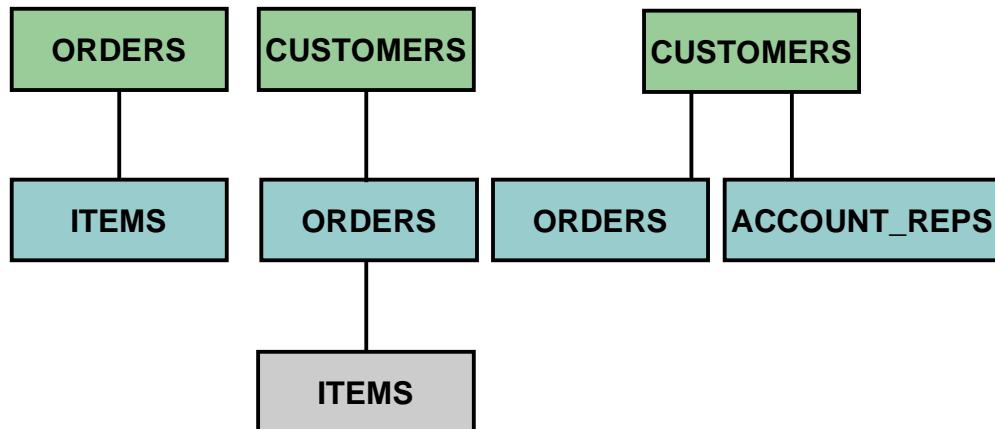
A Detail Block Can Be a Master

You can create block relationships in which the detail of one master-detail link is the master for another link.

A Master Block Can Have More Details

You can create more than one detail blocks for a master block.

Block Relationship: Examples



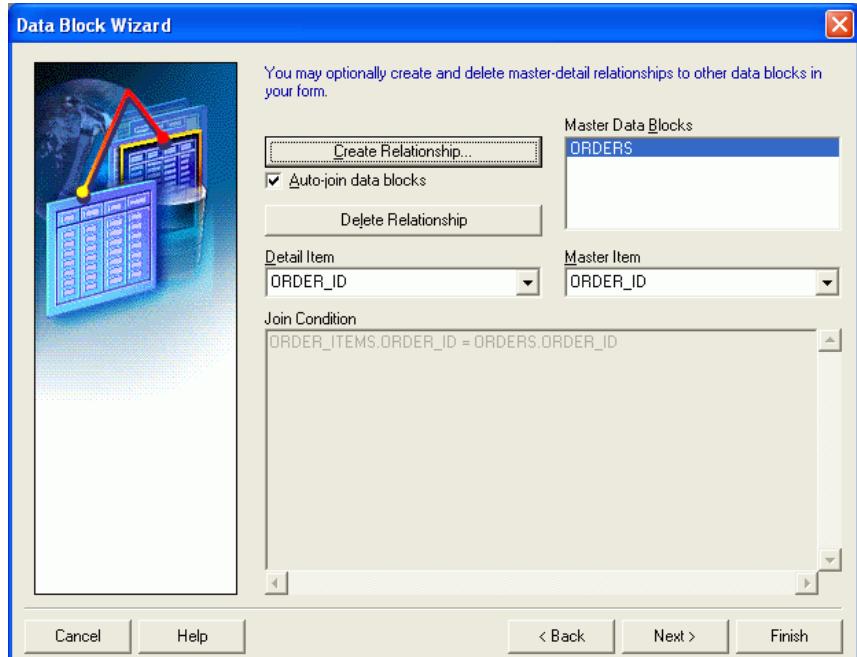
ORACLE®

Blocks Relationship: Examples

The following are examples of the master-detail structure:

- **Master-detail:** ORDERS to ITEMS, shown at the left of the slide
- **Master-detail-detail:** CUSTOMERS to ORDERS to ITEMS, shown in the middle of the slide
- **Master-2*detail:** CUSTOMERS to ORDERS and CUSTOMERS to ACCOUNT_REPS, shown at the right of the slide

Data Block Wizard: Master-Detail Page



ORACLE®

5 - 5

Copyright © 2009, Oracle. All rights reserved.

Data Block Wizard: Master-Detail Page

You can build a master-detail form module either by creating a relation between a master and detail block explicitly or by using the Data Block Wizard to create it implicitly.

1. Create the master block as described in the lesson titled “Creating a Basic Form Module.”
2. Invoke the Data Block Wizard in the Object Navigator.
3. Follow the same steps as before to create a new data block in the Data Block Wizard until you come to the Master-Detail page, shown in the slide. On this page, select the Auto-join data blocks check box and click Create Relationship.

Note: If the Auto-join data blocks check box is not selected, the Data Block dialog box is displayed with a list of all data blocks in the form without any foreign key constraint names.

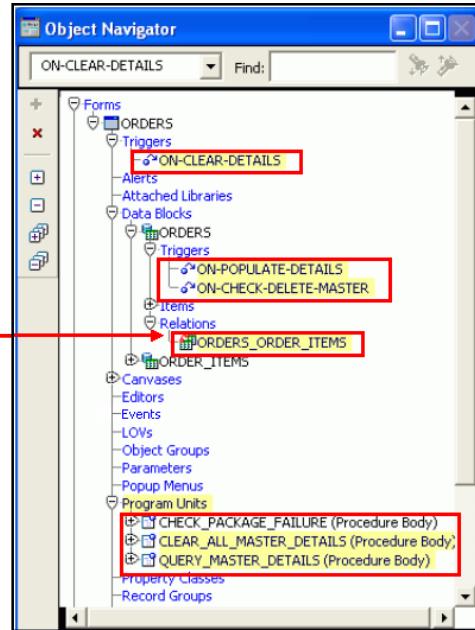
Data Block Wizard: Master-Detail Page (continued)

4. Select a master data block in the Data Block dialog box and click OK. The wizard automatically creates the join condition between the detail and master data blocks in the Join Condition field and displays the name of the master data block in the Master Data Blocks field.
Note: If the Auto-join data blocks check box is not selected, the wizard does not automatically create the join condition between the detail and master data blocks. You must use the Detail Item and Master Item pop-up lists to create a join condition manually.
5. Click Next, and then complete the Data Block Wizard steps. Perform the Layout Wizard steps as described earlier in the lesson titled “Creating a Basic Form Module” to finish creating and laying out the detail data block.
Note: The master data block must exist in the form module before you create the detail block.

You can also create a relation by invoking the Data Block Wizard in reentrant mode.

Examining the New Relation

- New relation object created in Object Navigator under master data block node
- Default name assigned: `<MasterDataBlock>_<DetailDataBlock>`
- Triggers and program units generated automatically



Examining the New Relation

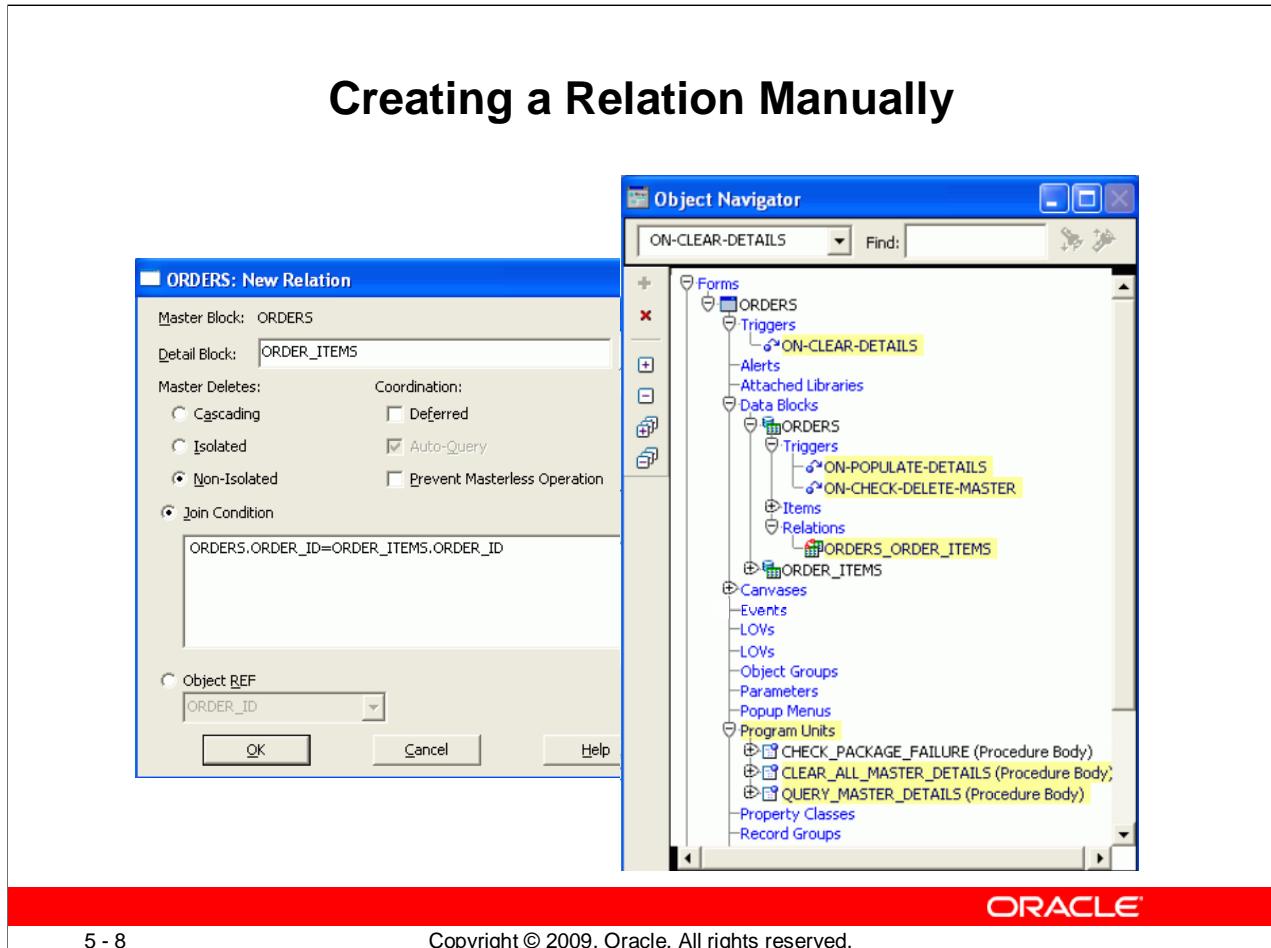
After you create a master-detail form module, the Data Block Wizard automatically creates a form object that handles the relationship between two associated data blocks. This object is called a *relation*. The following tasks occur automatically:

- The new relation object is created under the master data block node in the Object Navigator with default properties.
- The relation is given the following default name:
`<MasterDataBlock>_<DetailDataBlock>`—for example, CUSTOMERS_ORDERS.
- Triggers and program units are generated to maintain coordination between the two data blocks.

The screenshot shows the following objects that are created in the Object Navigator for a relation:

- A form-level On-Clear-Details trigger (you learn more about triggers in the lesson titled “Introduction to Triggers”)
- Block-level triggers for the master block: On-Populate-Details and On-Check-Delete-Master
- The relation itself
- Three program units: CHECK_PACKAGE_FAILURE, CLEAR_ALL_MASTER_DETAILS, and QUERY_MASTER_DETAILS

Creating a Relation Manually



5 - 8

Copyright © 2009, Oracle. All rights reserved.

Creating a Relation Manually

You can create a relation either implicitly with the Data Block Wizard, or explicitly in the Object Navigator.

Explicit Relations

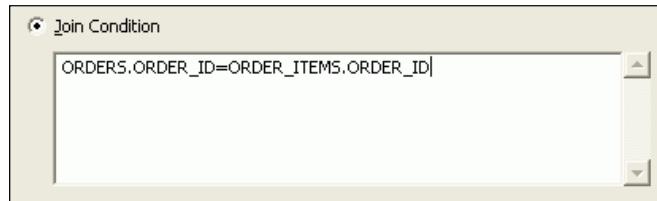
If a relation is not established when default blocks are created, you can create your own. To explicitly create a relation, perform the following steps:

1. Select the Relations node under the master block entry in the Object Navigator.
2. Click the Create icon. The New Relation window is displayed.
3. Specify the name of the detail block.
4. Choose your master delete property.
5. Choose your coordination property.
6. Specify the join condition.
7. Click OK.

The new relation, new triggers, and new program units are highlighted in the Object Navigator. Like implicitly created relations, the relation is given the default name of MasterDataBlock_DetailDataBlock—for example, CUSTOMERS_ORDERS. The same PL/SQL program units and triggers are created automatically when you explicitly create a relation as when the relation is created implicitly.

Defining the Join Condition

- The join condition creates a primary key/foreign key link between blocks.
- Define a join condition by using:
 - Block and item names (not table and column names). Do not precede names with colon.
 - SQL equijoin syntax



ORACLE®

5 - 9

Copyright © 2009, Oracle. All rights reserved.

Defining the Join Condition

Use a join condition to:

- Create links between blocks using SQL
- Alter links between blocks using SQL

Define a join condition using:

- Usual SQL equijoin condition syntax (necessary because Forms copies the value from the master block to the related item in the detail block)
- Block names instead of the base table names (do not precede with colon)
- Item names that exist in the form module rather than base table column names

The screenshot shows the part of the New Relation dialog box where you define the join condition. The example shows the condition:

ORDERS . ORDER_ID=ORDER_ITEMS . ORDER_ID

Running a Master-Detail Form Module

- Automatic block linking for:
 - Querying
 - Inserting
- Default deletion rules:
Cannot delete master record if detail records exist

Order Id	Line Item Id	Product Id	Unit Price
2458	1	3117	38
2458	2	3123	79
2458	3	3127	488.4
2458	4	3134	17
2458	5	3143	15

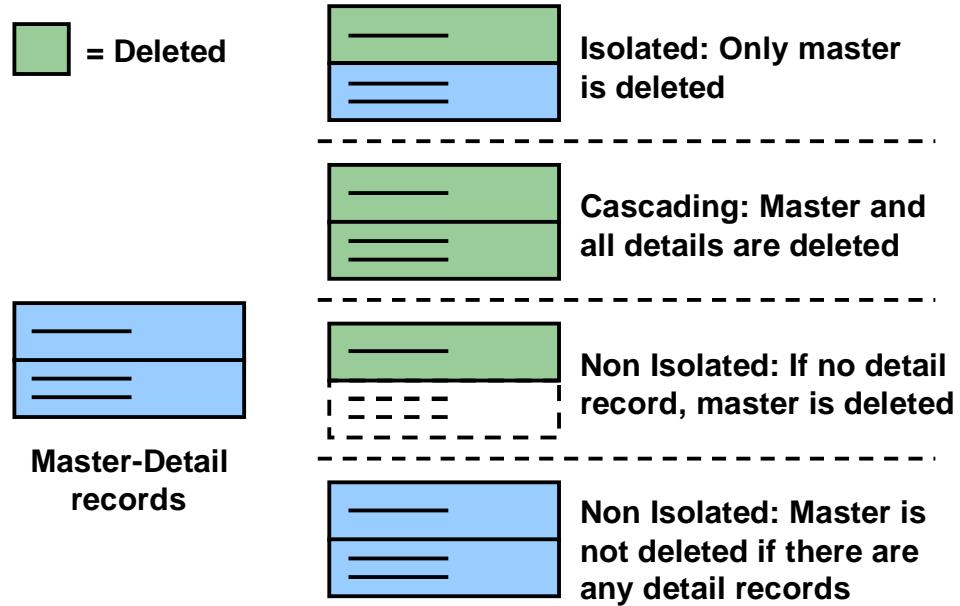
Running a Master-Detail Form Module

When you run your master-detail form module, you will find that:

- Querying the master data block immediately by default retrieves corresponding detail records, as shown in the screenshot
- Deleting a master record is prevented if detail records exist
- Inserting a detail record automatically associates it with the currently displayed master

Note: You can change the above behavior by modifying the relation object properties.

Setting Delete Record Behavior



Setting Delete Record Behavior

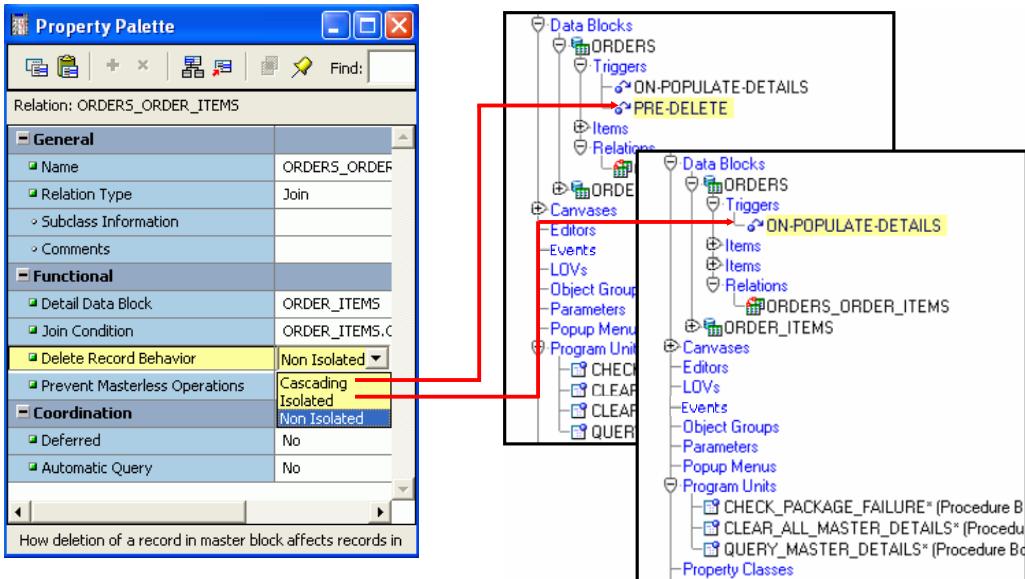
By setting the Delete Record Behavior property, you can prevent, propagate, or isolate deletion of a record in a master block when corresponding records exist in the detail block. For example, you can delete all corresponding line items when an order is deleted.

Property Value	Use
Non Isolated	Prevents the deletion of the master record when detail records exist; the master record is deleted only if no detail records exist
Cascading	Deletes the detail records when a master record is deleted
Isolated	Deletes only the master record

The graphics show what happens to master and detail records with different settings of the Delete Record Behavior property, as described in the text beside each graphic.

Note: Although deleting with the cascading property may remove many detail records, the commit message shows only the number of records deleted from the master block.

Delete Record Behavior and Triggers



ORACLE®

Delete Record Behavior and Triggers

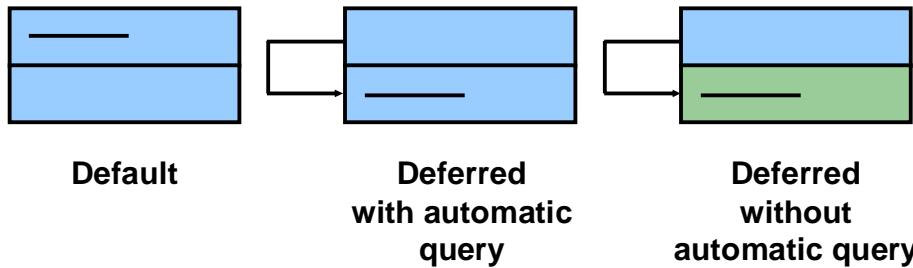
Changing the Delete Record Behavior property value affects the block-level triggers of the master block as follows:

- If you change the Delete Record Behavior property from the default of Non Isolated to Cascading, the On-Check-Delete-Master trigger is replaced with the Pre-Delete trigger.
- If you change the Delete Record Behavior property from the default of Non Isolated to Isolated, the On-Check-Delete-Master trigger is removed.

The screenshots show the block-level triggers in the Object Navigator when the Delete Record Behavior property is set to:

- **Cascading:** On-Populate-Details and Pre-Delete
- **Isolated:** On-Populate-Details only

Setting Coordination Properties



ORACLE®

5 - 13

Copyright © 2009, Oracle. All rights reserved.

Setting Coordination Properties

Setting the two coordination properties (Deferred and Automatic Query) controls how the detail records are displayed when a master block is queried. For example, you can defer querying the line items for an order until the operator navigates to the item block.

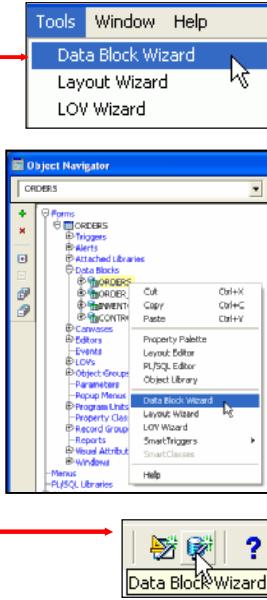
Coordination Property	Use
Default (depicted in first slide graphic)	Forces coordination of blocks to occur whenever the master record is changed by a user or a trigger
Deferred with Automatic Query (depicted in second slide graphic)	Postpones potentially expensive detail query processing until the cursor visits the related blocks
Deferred without Automatic Query (depicted in first third graphic)	Allows entry of additional query criteria in the detail block before querying
Prevent Masterless Operations	Ensures that the detail block cannot be queried or used to insert records when a master record is not displayed

Note: In the New Relation dialog box, setting the Deferred property to Yes enables the Auto Query check box.

Modifying the Structure of a Data Block

You can modify the structure by using:

- Reentrant Data Block Wizard:
 1. Select frame or object in the Layout Editor, or data block or frame in Object Navigator.
 2. Select Tools > Data Block Wizard, or right-click and select Data Block Wizard, or click Data Block Wizard.
- Object Navigator:
 - Create or delete items.
 - Change item properties.
- Block Property Palette: Change property values.



ORACLE®

Modifying the Structure of a Data Block

After you create a data block, you may want to customize or modify it by performing one of the following:

- Reenter the Data Block Wizard and use it to make the changes.
- Make manual changes, such as adding or deleting items, in Object Navigator.
- Change the property values of the block by using the Property Palette.

Invoking the Data Block Wizard in Reentrant Mode

A very powerful feature of the Data Block Wizard is its ability to operate in reentrant mode. Use reentrant mode to modify the data block, even if the block was not originally created with the Data Block Wizard.

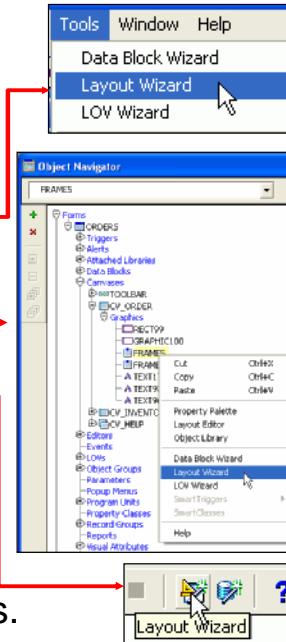
To invoke the Data Block Wizard in reentrant mode, perform the following steps:

1. Select the frame or a component of the block in either the Object Navigator or the Layout Editor.
2. Invoke the Data Block Wizard by performing one of the following steps, depicted in the screenshots:
 - Select Tools > Data Block Wizard from the menu.
 - Right-click and select Data Block Wizard from the pop-up menu.
 - Click Data Block Wizard.

Modifying the Layout of a Data Block

You can modify the layout by using:

- Reentrant Layout Wizard:
 - Select frame in Object Navigator or Layout Editor.
 - Select Tools > Layout Wizard,
or
right-click and select Layout Wizard, or
click Layout Wizard.
- Layout Editor:
 - Select Tools > Layout Editor.
 - Make changes manually.
- Frame Property Palette: Change values.



Modifying the Layout of a Data Block

You may want to customize or modify the layout of the data block items on the canvas. You can do this by performing one of the following steps:

- Reenter the Layout Wizard (see the next section) and use it to make the changes.
- Select Tools > Layout Editor to invoke the Layout Editor and make changes manually in the editor.
- Change the property values of the frame in its Property Palette.

Invoking the Layout Wizard in Reentrant Mode

A very powerful feature of the Layout Wizard is its ability to operate in reentrant mode. Use reentrant mode to modify the layout of items in an existing frame, even if the frame was not originally created with the Layout Wizard.

Modifying the Layout of a Data Block (continued)

You can invoke the Layout Wizard in reentrant mode from the Object Navigator or the Layout Editor:

1. Select the appropriate frame in the Object Navigator (under the Canvases node) or in the Layout Editor.
2. As shown in the screenshots, select Tools > Layout Wizard,
or
right-click the frame and select the Layout Wizard option,
or
if you are in the Layout Editor, click Layout Wizard.

Note: Before you reenter the Layout Wizard, it is important to select the correct frame in the Object Navigator or the Layout Editor. If you overlook this when you reenter the Layout Wizard, you may create an additional frame instead of modifying the current frame.

Either method takes you to the Data Block page in the Layout Wizard. Use Next and Back as you do when not in reentrant mode, or go directly to a certain page by clicking its tab.

Summary

In this lesson, you should have learned that:

- You can create data blocks with relationships by using the Data Block Wizard or by manually creating a Relation object
- When you run a master-detail form, block coordination is automatic depending on properties of the Relation object
- You can modify a data block manually or with the Data Block Wizard in reentrant mode
- You can modify the layout manually or with the Layout Wizard in reentrant mode

ORACLE®

5 - 17

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you should have learned about:

- Creating data blocks with a master-detail relationship
- Modifying the data block layout:
 - Using reentrant wizards
 - Changing frame properties

Practice 5: Overview

This practice covers the following topics:

- Creating a master-detail form module
- Modifying data block layout by using the Layout Wizard in reentrant mode
- Saving and running the form module



Practice 5: Overview

In this practice, you create a new form module that displays the master-detail information.

- Create a master-detail form module called Orders. Create a master block based on the ORDERS table and a detail block based on the ITEMS table. Create a third data block that is not related to any other block in the form module. Base this block on the INVENTORIES table, and manually create a relation with the block based on the item table. Use the Forms Builder wizards to create all three data blocks.
- Invoke the Layout Wizard in reentrant mode, and change the layout of the ITEMS and INVENTORIES data blocks.
- Save and run the new form module on the Web.

Working with Data Blocks and Frames



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Identify the components of the Property Palette
- Manage object properties
- Create and use Visual Attributes
- Control the behavior and appearance of data blocks
- Control frame properties
- Create blocks that do not directly correspond to database tables
- Delete data blocks and their components

ORACLE®

6 - 2

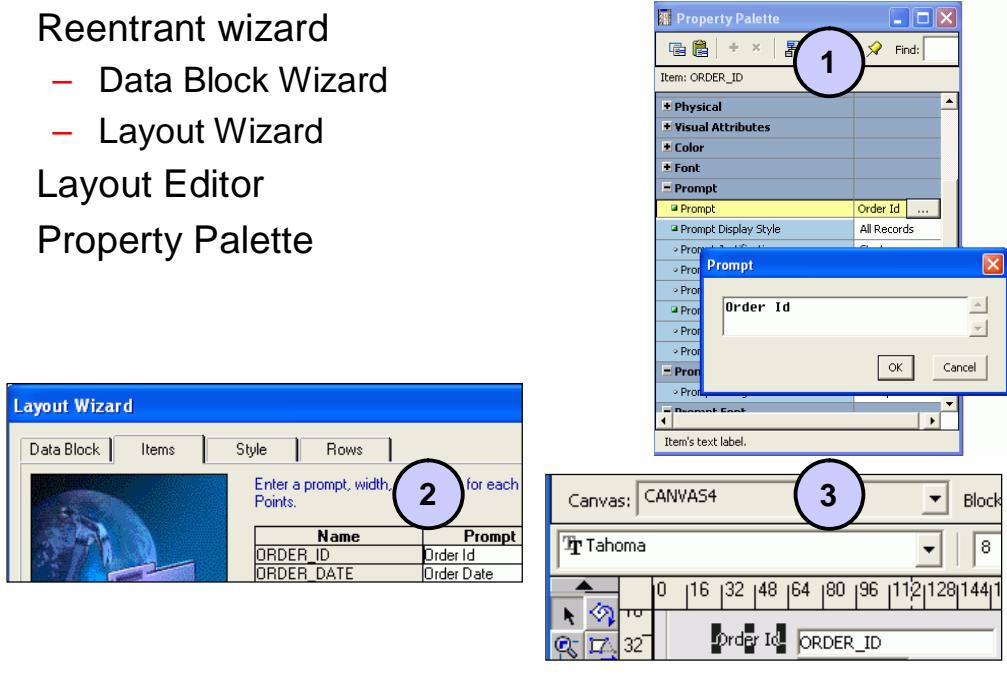
Copyright © 2009, Oracle. All rights reserved.

Lesson Aim

In this lesson, you learn how to customize existing data blocks and modify frames. You also learn how to include blocks that are not associated with the database.

Managing Object Properties

- Reentrant wizard
 - Data Block Wizard
 - Layout Wizard
- Layout Editor
- Property Palette



Managing Object Properties

There are three ways to modify properties of Forms Builder objects:

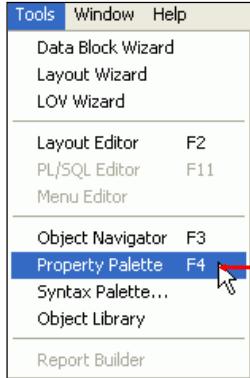
- **Reentrant Wizards:** You can modify data block and layout properties through the reentrant wizards, as explained in the previous lesson.
- **Layout Editor:** If the object appears on a canvas, you can modify properties by using the graphical Layout Editor.
- **Property Palette:** You can set individual properties for each Forms Builder object in its Property Palette.

The wizards, the Layout Editor, and the Property Palette all depict object properties. Changes made in one tool are reflected in the others. In the example in the slide, the prompt for Order_Id is shown identically in:

1. Property Palette
2. Reentrant layout wizard
3. Layout Editor

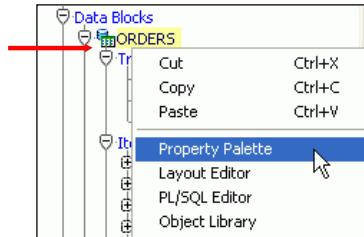
You can use the Property Palette to control the behavior and appearance of any Forms Builder object with a greater degree of granularity. In the Property Palette, you can fine-tune objects that you have initially created in the wizards or the Layout Editor.

Displaying the Property Palette



To display the Property Palette, use one of the following methods:

- Select Tools > Property Palette (or use the shortcut key).
- Double-click the object icon in the Object Navigator.
- Double-click the object in the Layout Editor.
- Right-click the object icon in the Object Navigator.
- Right-click the object in the Layout Editor.



ORACLE®

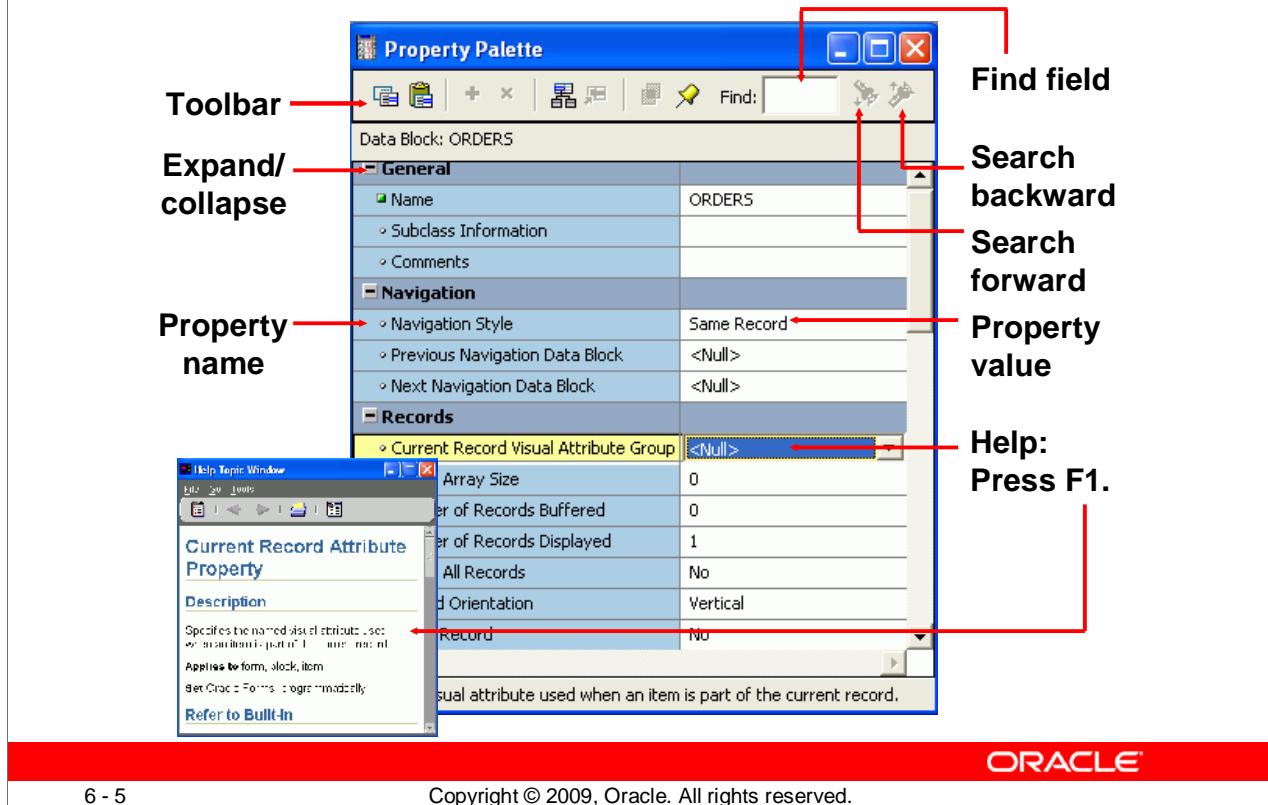
Displaying the Property Palette

Every object in a form module, as well as the form module itself, has properties that dictate the object's behavior. When an object is first created, it is automatically assigned several property values by default. You can change these property values in the Property Palette.

To display the Property Palette of an object, use one of the following methods:

- Select the object in the Object Navigator, and then select Tools > Property Palette from the menu, as shown in the first screenshot.
- Double-click the object icon for the object in the Object Navigator (except for code objects and canvases).
- Double-click an item in the Layout Editor.
- Right-click the object in the Layout Editor or the object icon in the Object Navigator. From the pop-up menu, select the Property Palette option, as shown in the second screenshot.

Property Palette: Features



6 - 5

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

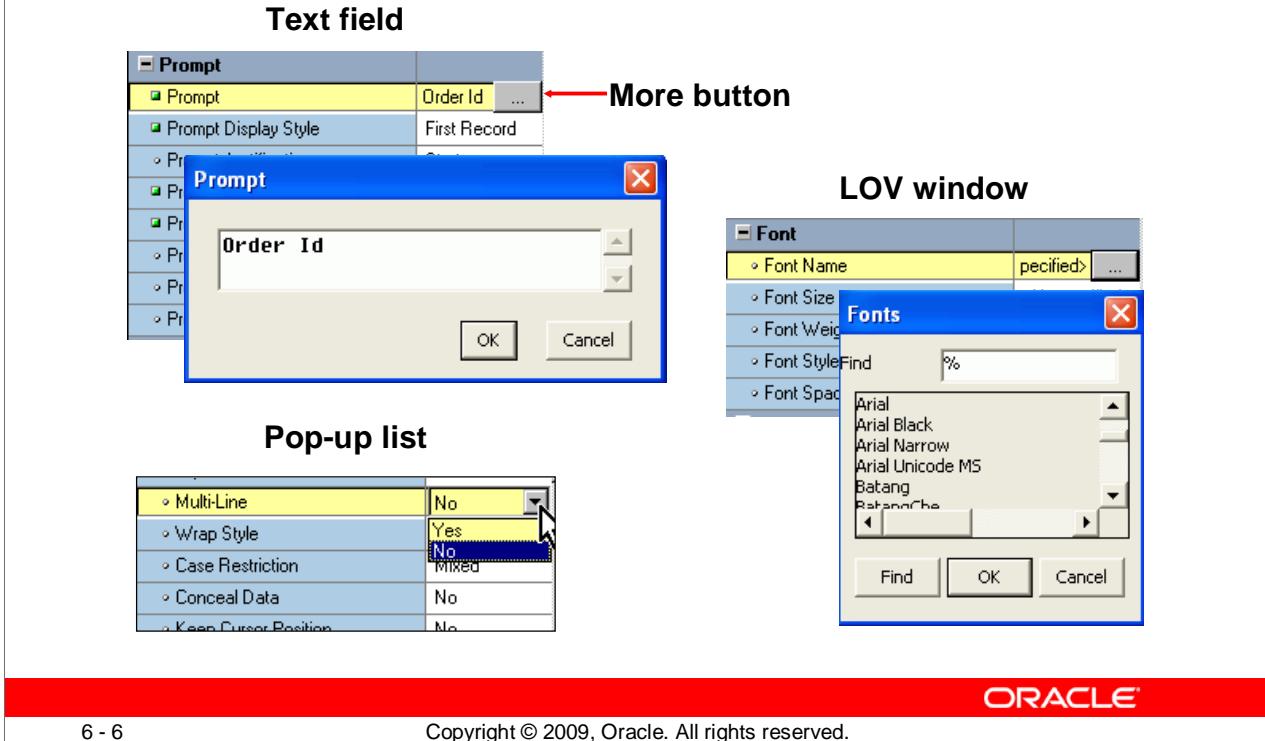
Property Palette: Features

The following are the features of the Property Palette:

Feature	Description
Property list	Displays a two-column list of names and values of properties that are valid for a particular object. Properties are grouped under functional headings or nodes. You can expand or collapse a node by using the plus and minus icons beside the node name.
Find field	Enables you to quickly locate a particular property. The Search Forward and Search Backward buttons enhance your search.
Toolbar	Consists of a series of buttons that provide quick access to commands
Help	Enables you to obtain description, usage, and other information about any property by pressing F1 with the property selected

The screenshots show each of these features of the Property Palette and the Help window for a particular property.

Using the Property Palette



Using the Property Palette

Each form object has various types of properties. Properties are manipulated differently, depending on the property type. The following is a summary of the controls that are used in the Property Palette:

Property Control	Description
Text field	A Text field is displayed where the current property can be set by entering a text value. For longer text values, an iconic button also appears, enabling you to open a text editor, as shown in the top-left screenshot in the slide.
Pop-up list	This appears where a fixed set of values, such as Yes or No, is allowed for the property. Click the down arrow to open the list, as shown in the bottom-left screenshot, and then select a value.
LOV window	LOVs occur where a potentially large list of possible values is available. Click the button in the property value column to invoke an LOV, as shown in the screenshot at the right of the slide.
More button	Use this when more complex settings are needed. Click More to open the extra dialog box, similar to the example in the screenshot at the top-left of the slide.

Differentiating Property Palette Icons

Changed		Visual Attribute Group	VISUAL_A1
Default		Prompt Visual Attribute Group	DEFAULT
Overridden		Color	
		Foreground Color	magenta
Inherited		Background Color	gray

ORACLE®

6 - 7

Copyright © 2009, Oracle. All rights reserved.

Differentiating Property Palette Icons

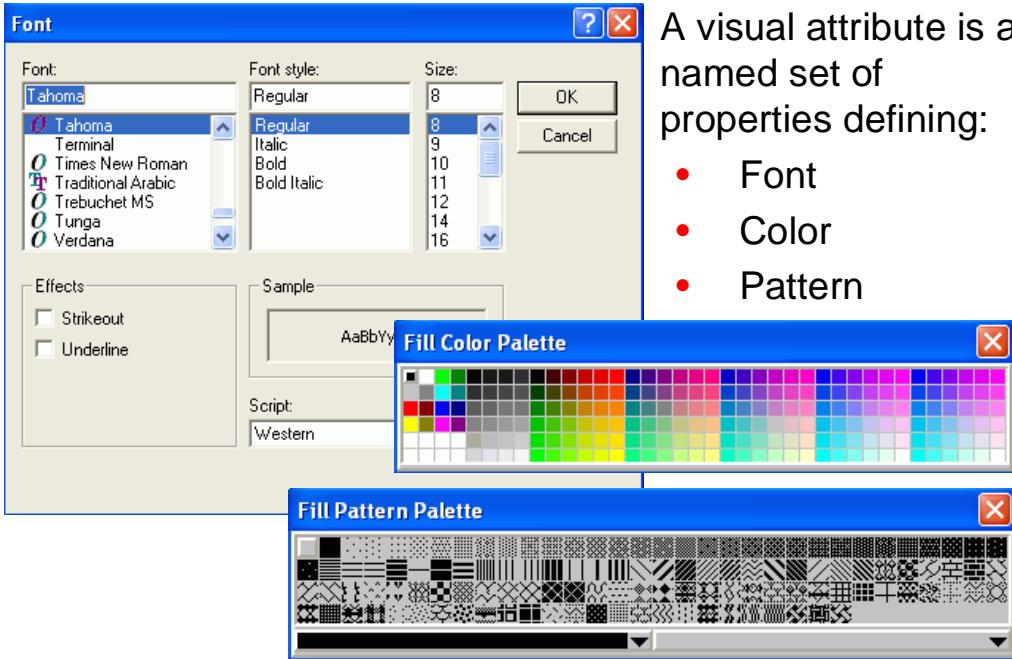
Each property in a Property Palette has an icon to its left. The following is a summary of these icons and their description:

Icon	Description
	Specifies that the property value is the default value
	Specifies that the property value has been changed from the default
	Specifies that the property value is inherited
	Specifies that the property value was inherited but has been overridden

Note: After you activate the Property Palette, its window remains open until you close it. The window automatically displays the properties of each object you visit in the Layout Editor or the Object Navigator. This is because, by default, the list of properties in the Property Palette is synchronized whenever you select an object.

You can turn the synchronization on or off for a specific palette by clicking Pin/Unpin on the Property Palette toolbar.

Visual Attributes: Overview



A visual attribute is a named set of properties defining:

- Font
- Color
- Pattern

Visual Attributes: Overview

Visual attributes are the font, color, and pattern properties that you set for form and menu objects. The screenshots show dialogs boxes for setting font, fill color, and fill pattern.

You can create a named set of such properties; this named set is called a Visual Attribute, which is another object that you can create in the Object Navigator with properties such as font, color, and pattern combinations. Set the Visual Attribute Type property of the Visual Attribute to Title if you plan to apply it to objects such as frame titles, or to Prompt if it is used for prompts. Otherwise, set Visual Attribute Type to Common, which is the default.

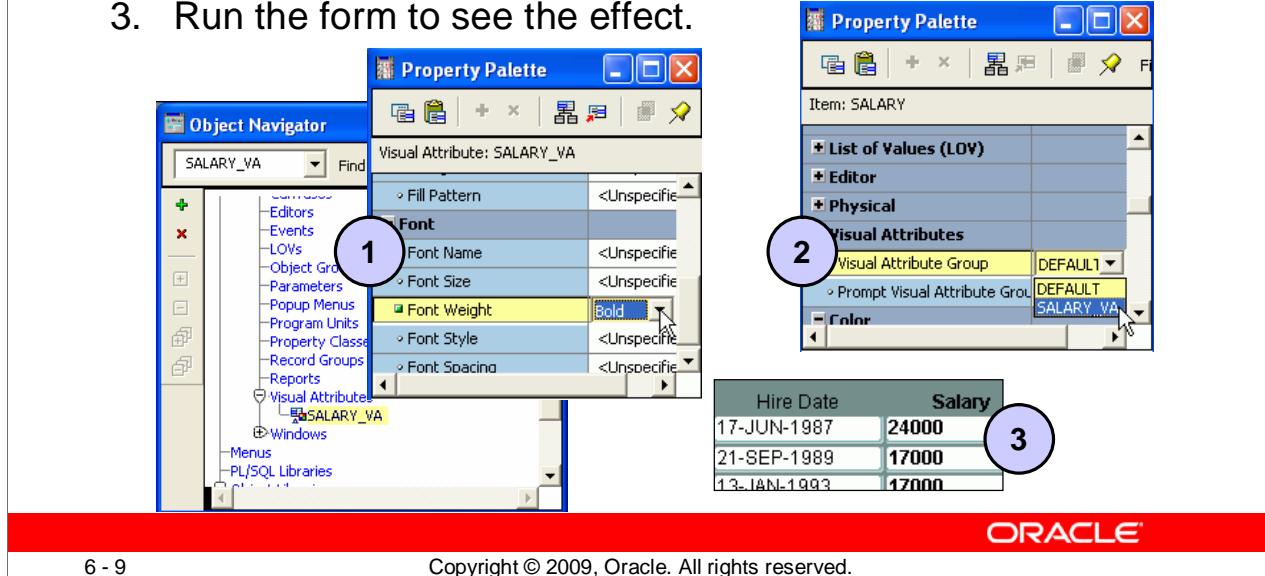
Every interface object in a Forms application has a property called Visual Attribute Group, which determines how the individual Visual Attribute settings of an object are derived. The Visual Attribute Group property can be set to Default, NULL, or the name of a Visual Attribute object. Blocks have a Current Record Visual Attribute Group property that defines the Visual Attribute to be used for the current record in the block.

Partial Visual Attributes

You can define a Visual Attribute by setting only the properties that you want to be inherited by the objects that use them. This means that you can apply a Visual Attribute that changes the font color without having to set the font name.

Using Visual Attributes

1. Create a Visual Attribute.
2. Set the Visual Attribute-related property of an object to the desired Visual Attribute.
3. Run the form to see the effect.

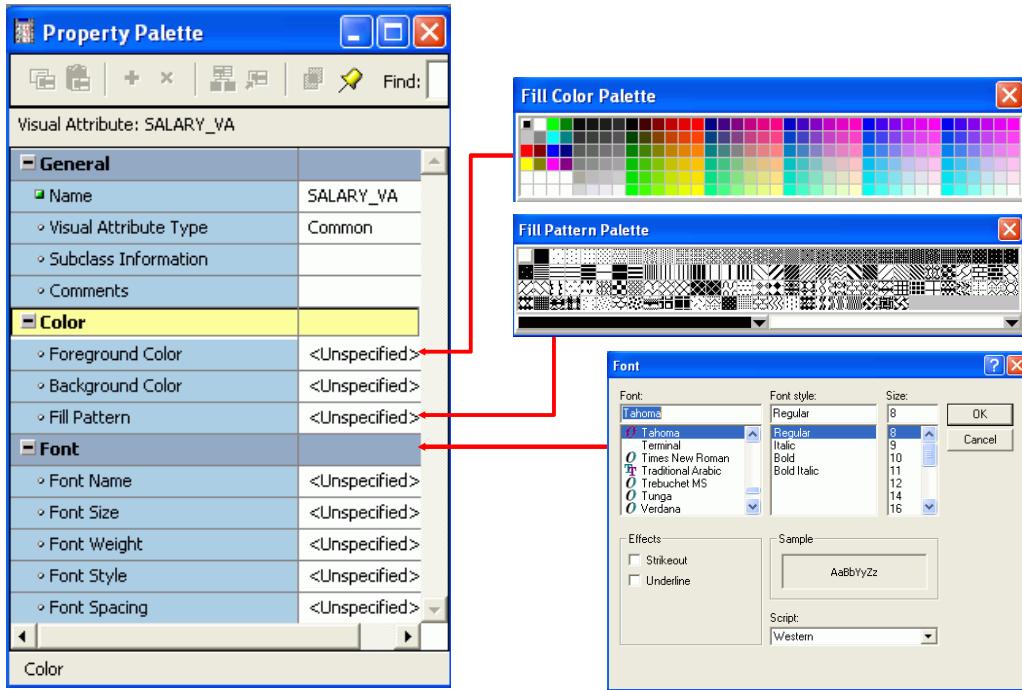


Using Visual Attributes

To use a Visual Attribute, perform the following steps:

1. Create the Visual Attribute:
 - Select the Visual Attributes node in the Object Navigator.
 - Click Create.
 - Invoke the Property Palette for the Visual Attribute and set the desired font, color, and pattern properties. You need set only those properties that you want to use. The screenshot at the left shows defining a Visual Attribute named SALARY_VA with a font weight of bold.
2. Set the object properties to use the new Visual Attribute:
 - For items and canvases, set the Visual Attribute Group property. The top-right screenshot shows setting the Visual Attribute Group property for the Salary text item to SALARY_VA, the Visual Attribute that was defined.
 - For blocks, you can set the Current Record Visual Attribute Group property.
3. Run the form to see the changes. The bottom-right screenshot shows that at run time, the Salary text item appears in a bold font.

Using Font, Pattern, and Color Pickers



ORACLE®

Using Font, Pattern, and Color Pickers

When you create Visual Attributes, you can use the Font, Pattern, and Color pickers to select the font, pattern, and color.

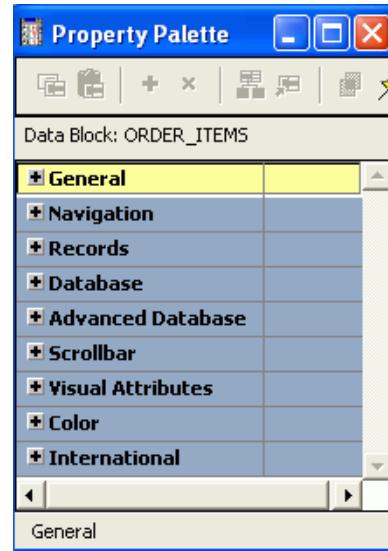
When changing a font from the Property Palette, you can click the Font group itself to invoke the Font Picker. This enables you to select all font properties from one window; otherwise, you can select each font property and invoke individual windows or pop-up lists that are specific to that property.

The screenshots show the fill color picker, the pattern picker, and the font picker that you can invoke from the Property Palette. When you click the value of one of the visual properties, such as Foreground Color, Fill Pattern, or Font, a More button appears that can invoke the appropriate interface for setting the property.

Controlling Data Block Behavior and Appearance

Data Block Property groups:

- General
- Navigation
- Records
- Database
- Advanced Database
- Scrollbar
- Visual Attributes
- Color
- International

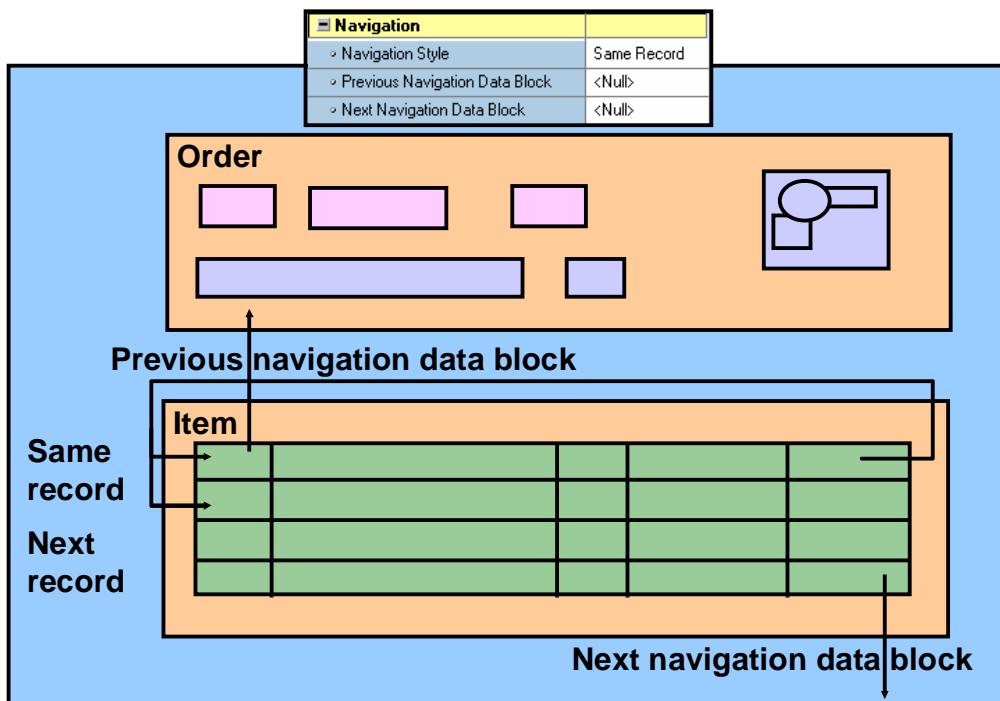


Controlling Data Block Behavior and Appearance

Each data block has several properties. These properties are divided into groups as shown in the slide.

You can modify the properties of the data block to control both the appearance and the behavior of the block.

Setting a Block's Navigation Properties

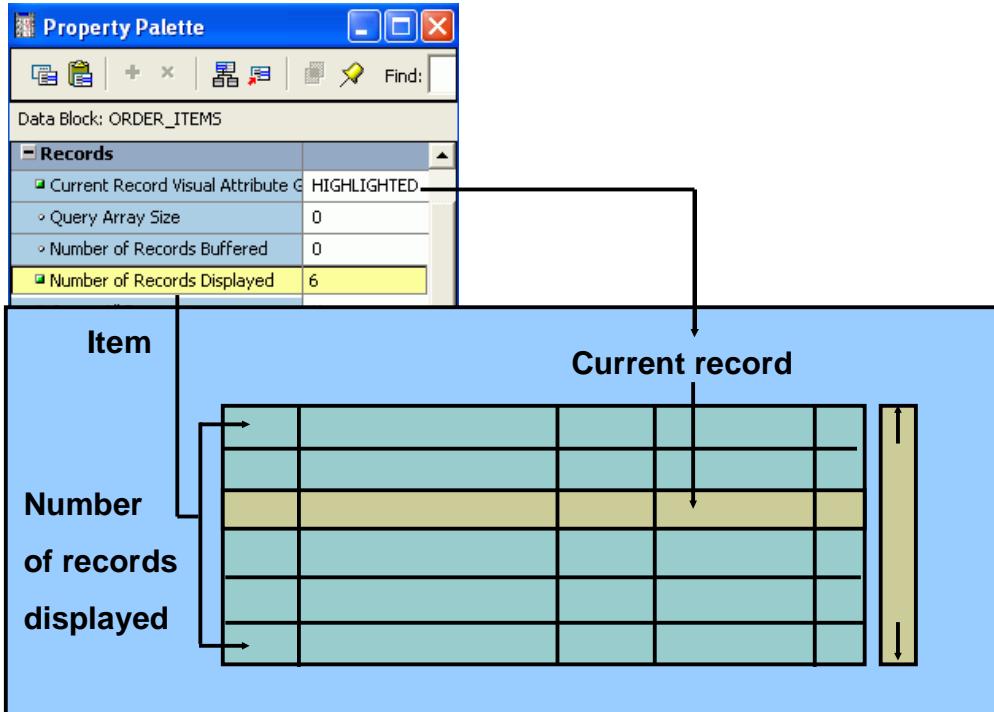


Setting a Block's Navigation Properties

The graphics show that, by default, when you navigate beyond the last item in a record, Forms returns you to the beginning of the same record. When you perform an operation to move to the previous or next data block at run time, Forms moves control to the previous or next adjacent data block in the sequence that these blocks appear in the Object Navigator. The screenshot shows the navigation properties that you can use to change default behavior:

- **Navigation Style:** With this property, you can change this behavior to navigate to the next record or data block instead. The valid settings are Same Record (default), Change Record, or Change Data Block. For example, if you want the cursor to move to the next record when you reach the end of the current record, set the Navigation Style property for the block to Change Record.
- **Previous/Next Navigation Data Block:** These properties enable you to name the previous or next data block, rather than using the default Object Navigator sequence.

Setting a Block's Records Properties



Setting a Block's Records Properties

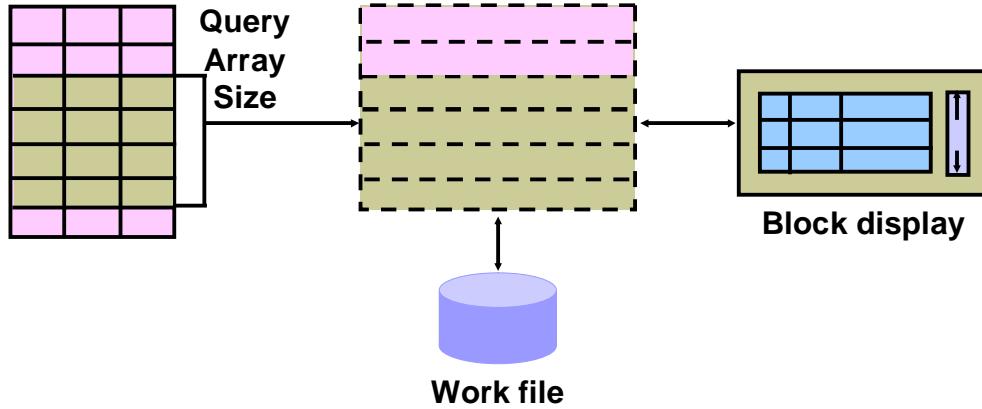
The screenshot shows the properties of a block that pertain to records:

- **Current Record Visual Attribute Group:** This is the Visual Attribute that is used to highlight the current record in the data block.
- **Number of Records Displayed:** This property specifies the maximum number of records that the data block can display on the canvas at one time and how many records you can see at once. If you change this value, make sure there is enough room on the canvas layout for the number of records, or the objects may overlap. The slide shows this property set to 6, with 6 rows displayed at run time.

Setting a Block's Records Properties

Records	
Current Record Visual Attribute Group	<Null>
Query Array Size	4
Number of Records Buffered	0
Number of Records Displayed	3
Query All Records	No
Record Orientation	Vertical
Single Record	No

Records buffered



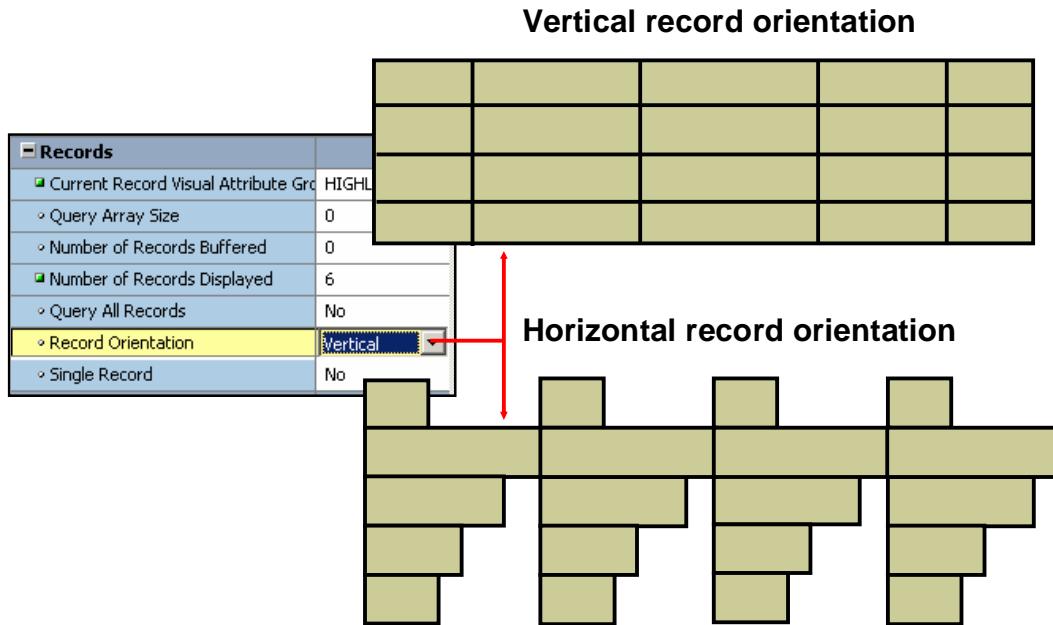
ORACLE®

Setting a Block's Records Properties (continued)

- **Query Array Size:** This specifies the maximum number of records that Forms should fetch from the database at one time. A lower value in this property value means faster response time; however, a larger value means fewer calls to the database for records, thereby resulting in reduced overall processing time. When set to 0, this property defaults to the Number of Records Displayed.
- **Number of Records Buffered:** This is the minimum amount of buffer space retained for holding queried records in the data block. The minimum setting allowed is the value of the Number of Records Displayed property plus 3. Forms buffers any additional records to a temporary disk file. A higher value improves processing speed, but uses more memory.

The screenshot shows Query Array Size set to 4 and Number of Records Buffered set to 0. The result of these settings is that 4 records at a time are fetched by the query into the record buffer, whose size, since Number of Records Buffered is 0, is 6 (the minimum size of Number of Records Displayed plus 3.)

Setting a Block's Records Properties



Setting a Block's Records Properties (continued)

- **Query All Records:** This property specifies whether all the records matching the query criteria should be fetched when a query is executed. (This property is necessary to support the Calculated Field feature.)
- **Record Orientation:** This property determines the orientation of records in the data block—horizontal or vertical. When you set this property, Forms Builder adjusts the display position of items in the data block accordingly. The top graphic shows vertical record orientation, with records displayed with each record below the previous one, while the lower graphic shows horizontal orientation, with records displayed beside one another.
- **Single Record:** This property specifies that the control block should always contain one record. Set this property to Yes for a control block that contains a summary calculated item. You cannot set this property to Yes for a data block.

Setting a Block's Database Properties

Use properties in the Database group to control:

- Type of block—data or control block
- Query, insert, update, and delete operations on the data block
- Data block's data source
- Query search criteria and default sort order
- Maximum query time
- Maximum number of records fetched

Database	
• Database Data Block	Yes
• Enforce Primary Key	No
• Query Allowed	Yes
• Query Data Source Type	Table
<input checked="" type="checkbox"/> Query Data Source Name	ORDERS
<input checked="" type="checkbox"/> Query Data Source Columns	
• Query Data Source Arguments	
• Alias	
• Include REF Item	No
• WHERE Clause	
<input checked="" type="checkbox"/> ORDER BY Clause	order_id
• Optimizer Hint	
• Insert Allowed	Yes
• Update Allowed	Yes
• Locking Mode	Automatic
• Delete Allowed	Yes
• Key Mode	Automatic
• Update Changed Columns Only	No
• Enforce Column Security	No
• Maximum Query Time	0
• Maximum Records Fetched	0

ORACLE®

Setting a Block's Database Properties

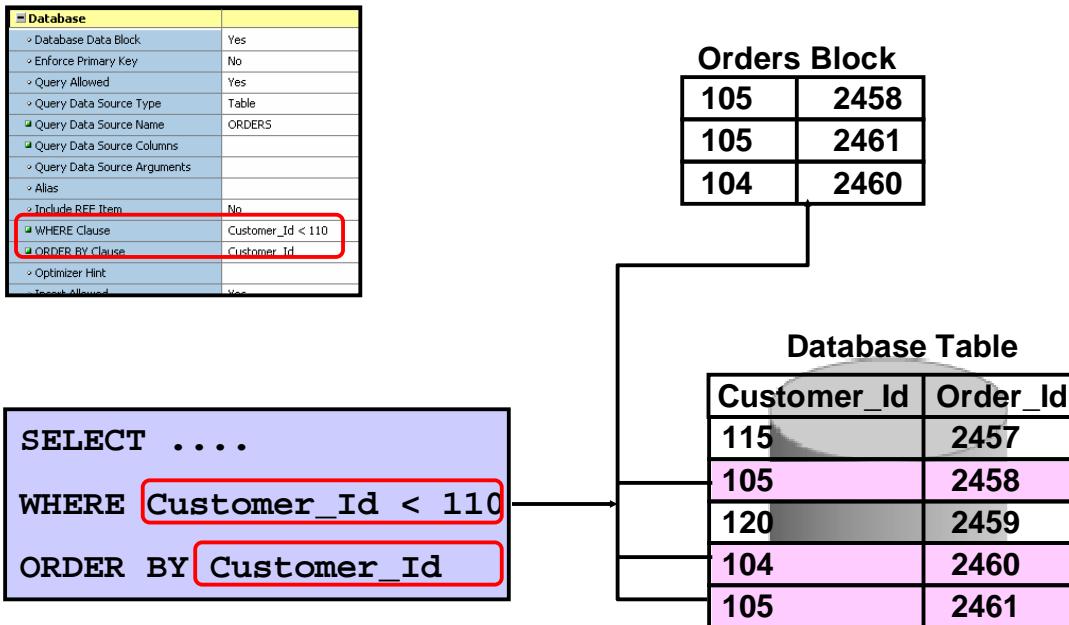
In the Database group of the Property Palette, you can set numerous properties to control interaction with the database server, as depicted in the screenshot. Some of these properties are:

- **Database Data Block:** Set to Yes if the data block is based on a database object, and No if it is a control block
- **Enforce Primary Key:** Controls whether Forms checks for uniqueness before inserting or updating records in the base table, in order to avoid committing duplicate rows in the database. A value of Yes means that Forms checks that inserted or updated records are unique before an attempt is made to commit possible duplicate rows.
- **Query Allowed/Insert Allowed/Update Allowed/Delete Allowed:** Control whether the associated operations can be performed on the data block records
- **Query Data Source Type:** Specifies the type of the query data source for the data block. Possible values for this property are None, Table, Procedure, Transactional Triggers, or the FROM clause query.
- **Query Data Source Name:** Specifies the name of the query data source for the data block. This property is used only if the type of the query data source is Table, the FROM clause query, or Procedure.

Setting a Block's Database Properties (continued)

- **Query Data Source Columns:** Specifies in a dialog box, the name and data type of the columns associated with the query data source. This property is used only if the type of the query data source is Table, the FROM clause query, or Procedure.
- **Query Data Source Arguments:** Specifies in a dialog box, the names, data types, and values of the arguments that are to be passed to the procedure for querying data. This property is valid only when the Query Data Source Type property is set to Procedure.
- **Optimizer Hint:** Specifies a hint string that Forms passes to the Optimizer when constructing implicit SQL on the data block. The Optimizer can improve the performance of database transactions.
- **Locking Mode/Key Mode:** Controls how Forms handles records and transactions when the data block is primarily associated with non-Oracle data sources. The default settings are usually appropriate for data blocks connected with an Oracle database.
- **Update Changed Columns Only:** By default, this property value is set to No, so that all columns are included in the default UPDATE statement; Forms can reuse the same SQL statement for multiple updates without the database having to reparse each time in its System Global Area (SGA). When this property is set to Yes, only those items updated by the operator are written to their corresponding database columns. If the operator commonly updates or inserts records with only one or two columns, this can save network traffic. However, the database must reparse the UPDATE statement each time, which can degrade performance.
- **Enforce Column Security:** When this property is set to Yes, items in the data block can be updated only if the current user has privileges to update the corresponding database columns.
- **Maximum Query Time:** Provides the option to abort a query when the elapsed time of the query exceeds the value of this property; useful when the Query All Records property is set to Yes
- **Maximum Records Fetched:** Provides the option to abort a query when the number of records fetched exceeds the value of this property; useful when the Query All Records property is set to Yes

Setting a Block's Database Properties: Query Clauses



Setting a Block's Database Properties: Query Clauses

There are two database properties that affect how the block's data query is constructed:

- **WHERE Clause:** Specifies a SQL condition that is attached to every default SELECT statement associated with the data block through implicit SQL; used to define general restrictions on the rows that this data block may fetch. This clause is automatically appended (ANDed) with any conditions supplied by the operator in Enter Query mode.
- **ORDER BY Clause:** Defines a default order for records displayed from a query. The operator can alter this order by using the Query/Where dialog box at run time.

The graphics show how the block's WHERE Clause and ORDER BY clause are used in the WHERE clause and the ORDER BY clause of the SQL statement that is sent to the database. The WHERE clause is appended to any existing query criteria (in this case there are none) to further restrict the rows returned by the query. The ORDER BY clause determines the display order in the data block.

Setting a Block's Scroll Bar Properties

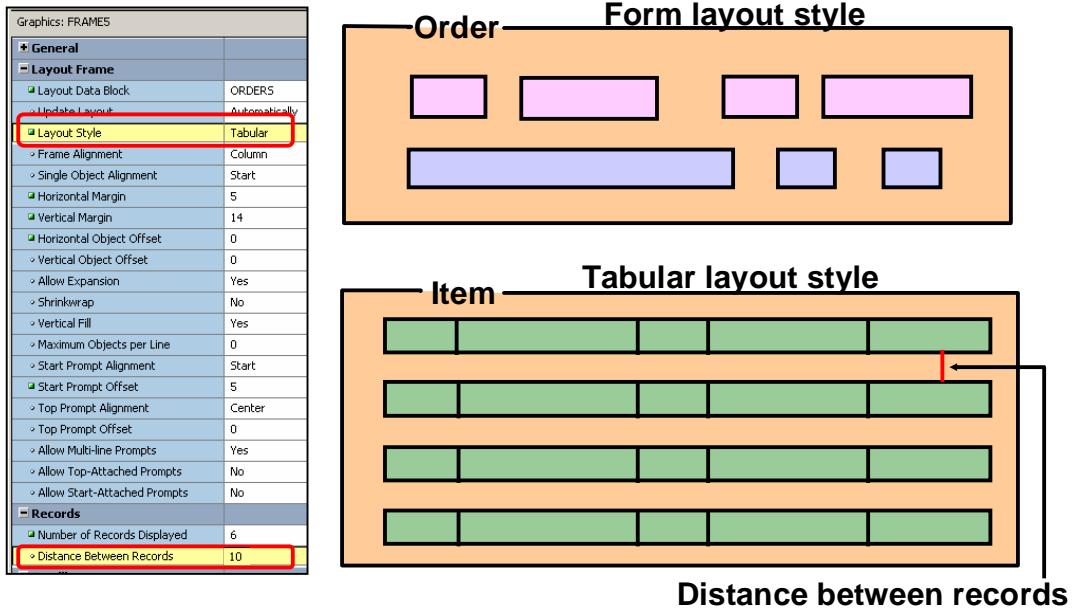


Setting a Block's Scroll Bar Properties

In the Scrollbar group of the Property Palette, you can set numerous properties to the appearance and function of the data block's scroll bar. Some of these properties are:

- **Show Scroll Bar:** Specifies whether Forms Builder should create a scroll bar for the data block. To delete an existing scroll bar, set this property to No.
- **Scrollbar Canvas/Tab Page:** Specifies the canvas and tab page on which the data block scroll bar is displayed. The specified canvas or tab page must exist in the form.
- **Scrollbar Orientation:** Specifies whether the scroll bar should be displayed horizontally or vertically
- **Scrollbar X/Y Position:** Specifies the x and y coordinates (measured in the coordinate system units of the form) where the scroll bar displays on the canvas. The default value for both the coordinates is 0. These properties are shown in the slide.
- **Scrollbar Width/Length:** Specifies the width and height of the scroll bar, as shown in the slide
- **Reverse Direction:** Determines whether the scroll bar should scroll in reverse

Controlling Frame Properties



Controlling Frame Properties

The selections that you make in the Layout Wizard when creating a data block are recorded as properties of the resulting layout frame object. You can change frame properties to modify the arrangements of items within a data block. The main frame properties are as follows:

- **Layout Data Block:** Specifies the name of the data block with which the frame is associated. The items within this data block are arranged within the frame. A data block can be associated with only one frame. You cannot arrange a block item within multiple frames.
- **Update Layout:** Specifies when the frame layout is updated. Valid settings are:
 - **Automatically:** The layout is updated whenever you move or resize the frame, or modify any frame layout property.
 - **Manually:** The layout is updated whenever you use the Layout Wizard to modify the frame, or in the Layout Editor, when you click Update Layout or select the Layout > Update Layout menu option.
 - **Locked:** The layout is locked and cannot be updated.
- **Layout Style:** Specifies the layout style for the items within the frame. Choose between Form and Tabular styles.

Controlling Frame Properties (continued)

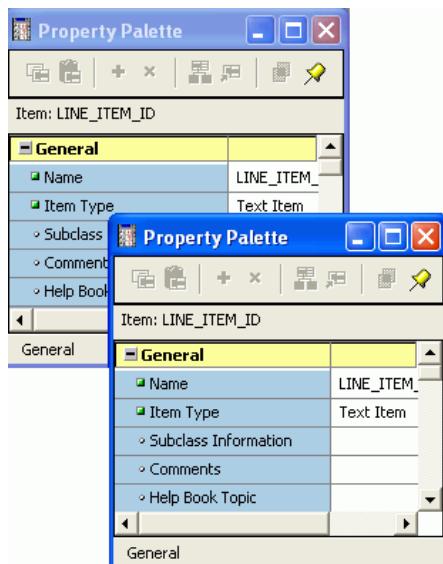
- **Distance Between Records:** Specifies the physical distance (measured in the form's coordination system units) with which to separate records displayed in the frame
- **X/Y Position:** Specifies the x and y coordinates (measured in the form's coordination system units) of the frame's position on the canvas
- **Width/Height:** Specifies the width and height of the frame (measured in the form's coordination system units)

The screenshot shows the Property Palette for a frame, and the graphics illustrate two of the frame's properties. The top graphic shows a form layout style, while the bottom one shows a tabular layout style. The size of the space between the records of a multirecord block is determined by the Distance Between Records property.

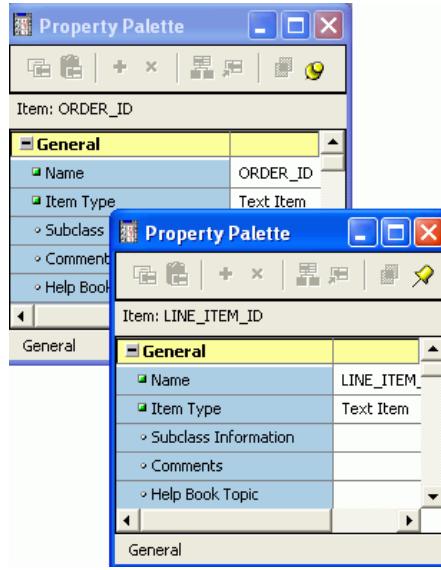
Note: You can arrange a frame as well as the objects within it manually in the Layout Editor.

Displaying Multiple Property Palettes

Two palettes for one item:



Two palettes for two items:



ORACLE®

Displaying Multiple Property Palettes

More Than One Property Palette for One Object

You may want to see more properties for an object than there is room for in a single Property Palette. To display the properties of an object in multiple Property Palettes, as shown by the screenshots to the left side of the slide, perform the following steps:

1. Open a Property Palette for the object.
2. Press and hold the Shift key and double-click the object icon for the object in the Object Navigator or the Layout Editor.

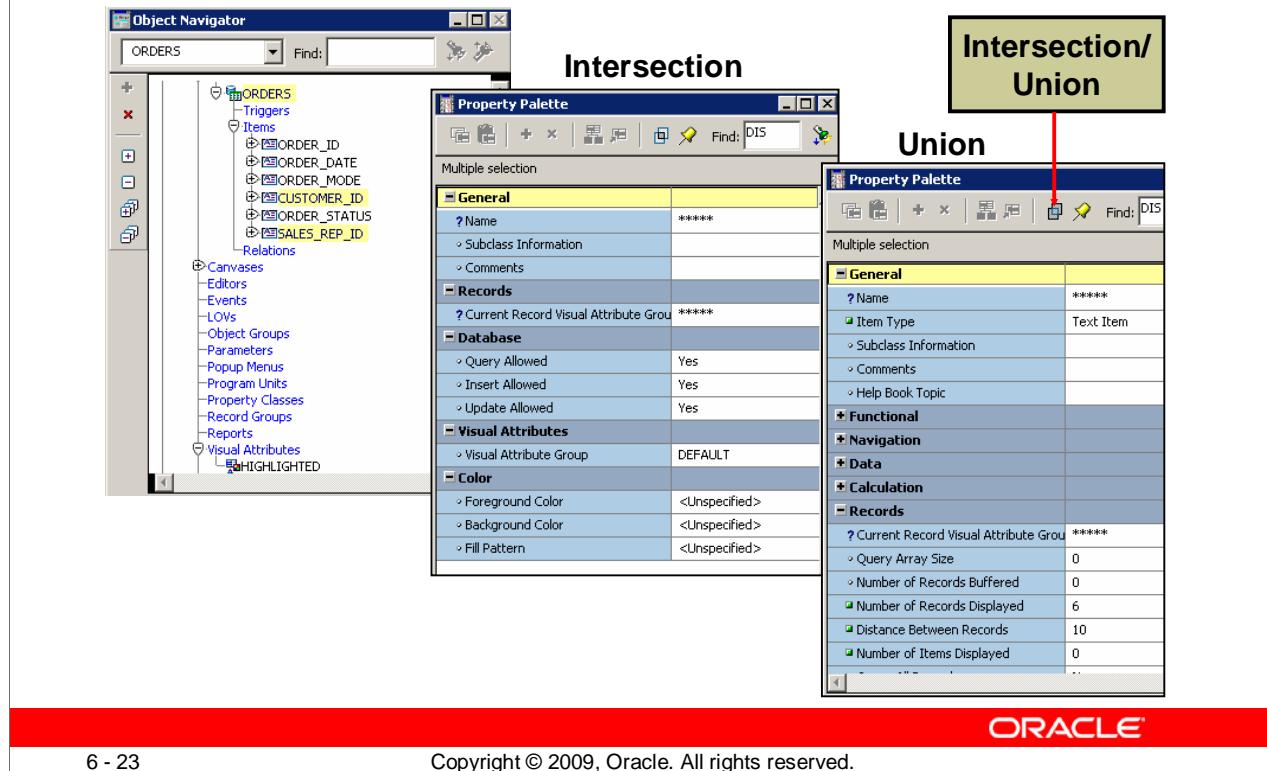
More Than One Property Palette for Multiple Objects

You may want to display properties for multiple objects simultaneously. To display the Property Palettes for multiple objects at the same time, as in the screen shots to the right side of the slide, perform the following steps:

1. Open the Property Palette of the first object.
2. Click Pin/Unpin on the toolbar to "freeze" this palette.
3. Invoke the Property Palette for another object. This Property Palette appears in a separate window.

If the second window is on top of the first one, drag it so that both windows are visible.

Setting Properties on Multiple Objects



6 - 23

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Setting Properties on Multiple Objects

You can view and set the properties of several objects simultaneously, whether they are the same or different object types. You can select the objects in the Object Navigator and display a combination of the properties in the Property Palette. The combination may be:

- **Intersection:** A subset in which you display only the common properties of the selected objects (This is the default set operator.)
- **Union:** A superset in which you display both the common properties and the unique properties of the selected objects

Where there are differing values for a property across the selected objects, you see ***** in the property value. This changes to a definitive value after you enter a new value in the Property Palette. This new value then applies to each of the selected objects to which the property is relevant.

The screenshot to the left side of the slide shows selecting several objects, while the other two screenshots show how the Property Palette appears for an intersection (middle screenshot) and for a union (right screenshot) of properties. Note that two of the properties show a line of asterisks for the value, meaning that the selected objects have different values for those properties.

Setting Properties on Multiple Objects (continued)

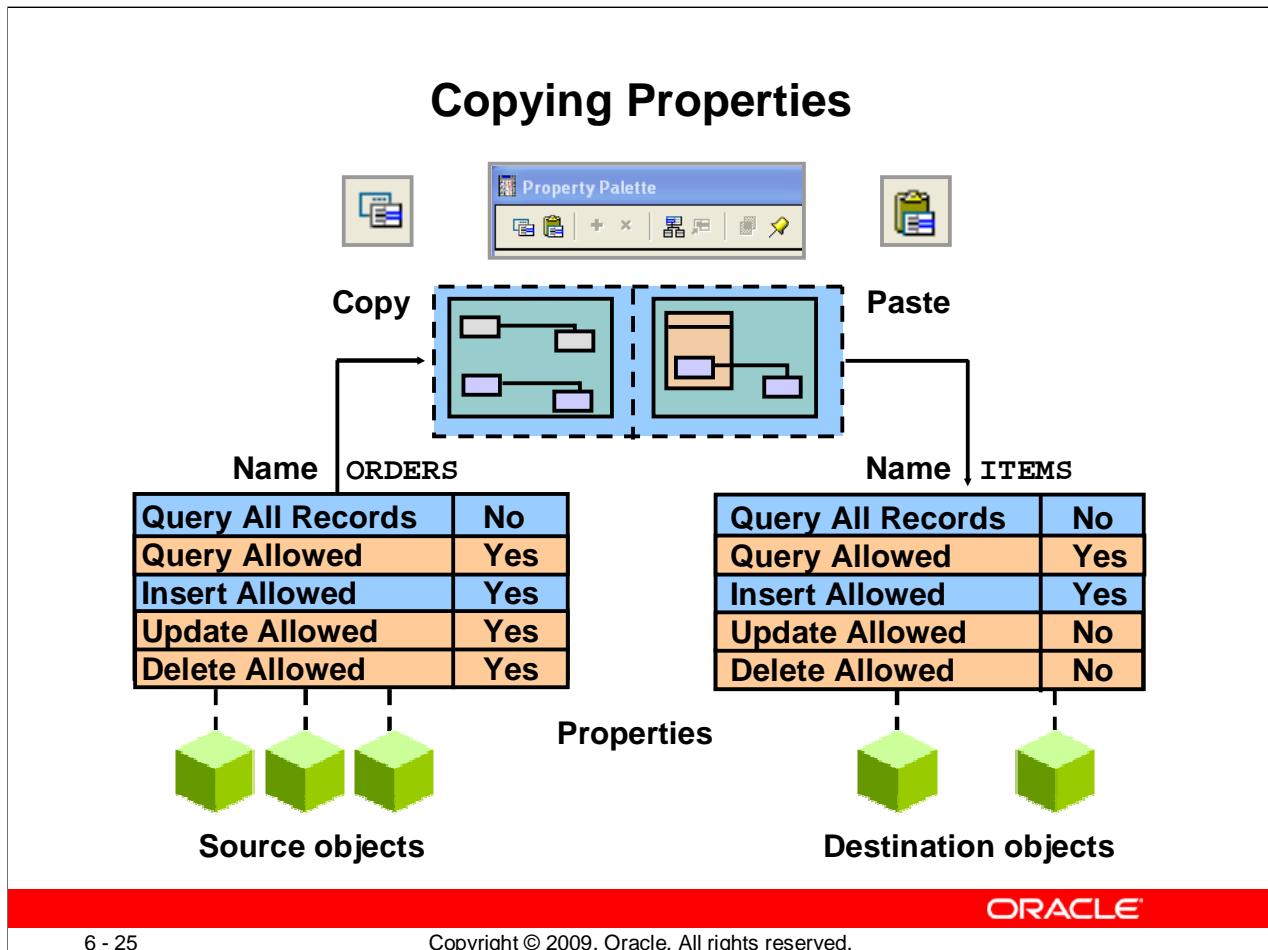
How to Set Properties on Multiple Objects

To set properties on multiple objects at one time, perform the following steps:

1. Open the Property Palette for one of the objects.
2. Press and hold the Ctrl key and select each object in the Object Navigator or the editors whose properties are to be viewed or changed in combination. The selected objects are highlighted.
3. Set the Intersection/Union button from the toolbar in the Property Palette to the desired set type. This button toggles between the two options.
4. Change the displayed properties, as required. Your changes are applied to all selected objects with these properties.

Note: With a union, you may see some properties that are not relevant to all of the selected objects. Changes to a property are applied only to objects that have the property.

Copying Properties



Copying Properties

You can copy the properties and values from the Property Palette to a buffer, so that they can be applied (pasted) to other objects in your design session. To copy properties, perform the following steps:

1. In the Property Palette, display and set the properties that are to be copied. This may be from one object or a combination of objects.
2. Select the properties to be copied:
 - To copy all property settings from the Property Palette, select Edit > Select All.
 - To copy the selected property settings only, press and hold Ctrl and select each property individually. This graphics at the left of the slide shows two properties being copied.
3. Click Copy Properties on the toolbar of the Property Palette.
4. From the Object Navigator, select the object into which the properties are to be copied.
5. In the Property Palette, click Paste Properties. The selected object receives values from all copied properties that are relevant to their object types, as shown in the graphics to the right side of the slide, where the pasted property values are applied.

Note: It is possible to copy the property settings of an object to objects of different types. In this case, properties that do not apply to the target object are ignored.

Copying Properties (continued)

Property Classes

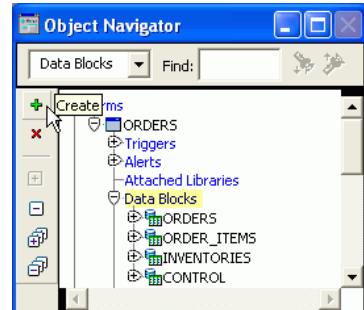
When you display a list of properties (from either one object or a combination of objects) in the Property Palette, the list of property names and associated values can be saved for future application to other objects. This is known as a property class, which is a Forms Builder object in its own right.

Objects can inherit some of their properties from a linked property class, so their properties change automatically if the associated properties are changed in the property class.

Property classes are discussed in more detail in the lesson titled “Sharing Objects and Code”.

Creating a Control Block

- Click the Data Blocks node.
- Click the Create icon.
or
Select Edit > Create.
- Select the Build a new data block manually option in the New Data Block dialog box.



ORACLE®

6 - 27

Copyright © 2009, Oracle. All rights reserved.

Creating a Control Block

A control block is a block that is not associated with any database, and its items do not relate to any columns within any database table.

This means that Forms does not perform an automatic query when the operator issues an Enter Query or Execute Query command, nor does it issue an automatic Insert, Update, or Delete for the block when the operator saves changes to the database.

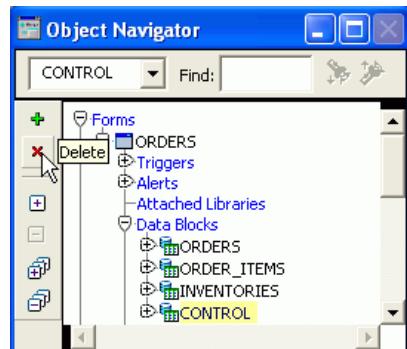
How to Create a Control Block

1. Click the Data Blocks node in the Object Navigator.
2. Click the Create icon on the toolbar, or select Edit > Create from the menu.
3. In the New Data Block dialog box, select the Build a new data block manually option.
4. Open the Property Palette of the new block and change its name.

Note: Because there are no database columns on which to base control block items, a control block has no items until you manually add them later.

Deleting a Data Block

- Select a data block for deletion.
- Click the Delete icon.
or
Press Delete.
- Click Yes in the alert box.



ORACLE®

Deleting a Data Block

To delete a block, perform the following steps:

1. Select the block to be deleted in the Object Navigator.
2. Click the Delete icon on the toolbar, or press Delete.
3. An alert is displayed for delete confirmation, as shown in the screen shot at the bottom left of the slide. Click Yes to delete the block.

Note: Deleting a block also deletes its subordinate objects (items and triggers). If the block was a master or detail block in a relation, the relation is also deleted. However, the frame border and its title will remain. Delete the frame manually in the Layout Editor.

Summary

In this lesson, you should have learned that:

- The Property Palette:
 - Contains property names and values that enable you to modify Forms objects
 - Has tools to search for properties, inherit properties, expand or collapse property categories, and pop-up lists and dialog boxes for various properties
 - Shows different icons for default, changed, inherited, and overridden properties
- Block properties control the behavior and appearance of data blocks
- Frame properties control how block items are arranged
- You can create blocks that do not directly correspond to database tables by choosing to create the block manually rather than using the Data Block Wizard
- Deleting a block deletes all of its components

ORACLE®

6 - 29

Copyright © 2009, Oracle. All rights reserved.

Summary

- Modify the data block properties in its Property Palette to change its behavior at run time.
- Data blocks have Navigation, Database, Records, Scrollbar, and other properties.
- Database properties include WHERE Clause, ORDER BY Clause, Query Data Source Type, and Maximum Records Fetched.
- You can change frame properties to modify the arrangements of items within a data block.
- You can copy properties between data blocks and other objects.
- You can view and change the properties of several objects together. You can use Intersection or Union settings to connect their properties in the Property Palette.

Practice 6: Overview

This practice covers the following topics:

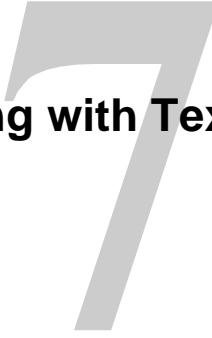
- Modifying data block properties
- Modifying frame properties
- Creating a Visual Attribute
- Creating a control block



Practice 6: Overview

In this practice, you will perform the following tasks:

- Change properties in the CUSTOMERS data block and frame to change run-time appearance and behavior and design-time behavior of the frame.
- Change properties in the ITEMS and INVENTORIES data blocks to change their run-time appearance and behavior.
- Change the frame properties of all the data blocks in the ORDERS form to change their run-time appearance and to keep any layout changes you make manually in the Layout Editor.
- Create a Visual Attribute in the ORDERS form and use it to highlight the current record in the ITEMS and INVENTORIES data blocks at run time. Use the multiple selection feature in the Property Palette.
- Create a control block in the ORDERS form.
- Create a control block in the CUSTOMERS form.
- Save and run the forms after the changes are applied.



Working with Text Items

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe text items
- Create a text item
- Modify the appearance of a text item
- Control the data in a text item
- Alter the navigational behavior of a text item
- Enhance the relationship between the text item and the database
- Add functionality to a text item
- Display helpful messages

ORACLE®

7 - 2

Copyright © 2009, Oracle. All rights reserved.

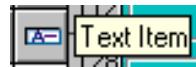
Lesson Aim

The default item type in an Oracle Forms application is the text item or field. You have seen how creating a new data block based on a table creates text items for each selected column from that table. This lesson shows you how to customize text items to change their appearance and behavior.

Text Item: Overview

What is a text item?

- Default item type
- Interface object for:
 - Querying
 - Inserting
 - Updating
 - Deleting
- Behavior defined in the Property Palette



ORACLE®

7 - 3

Copyright © 2009, Oracle. All rights reserved.

Text Item: Overview

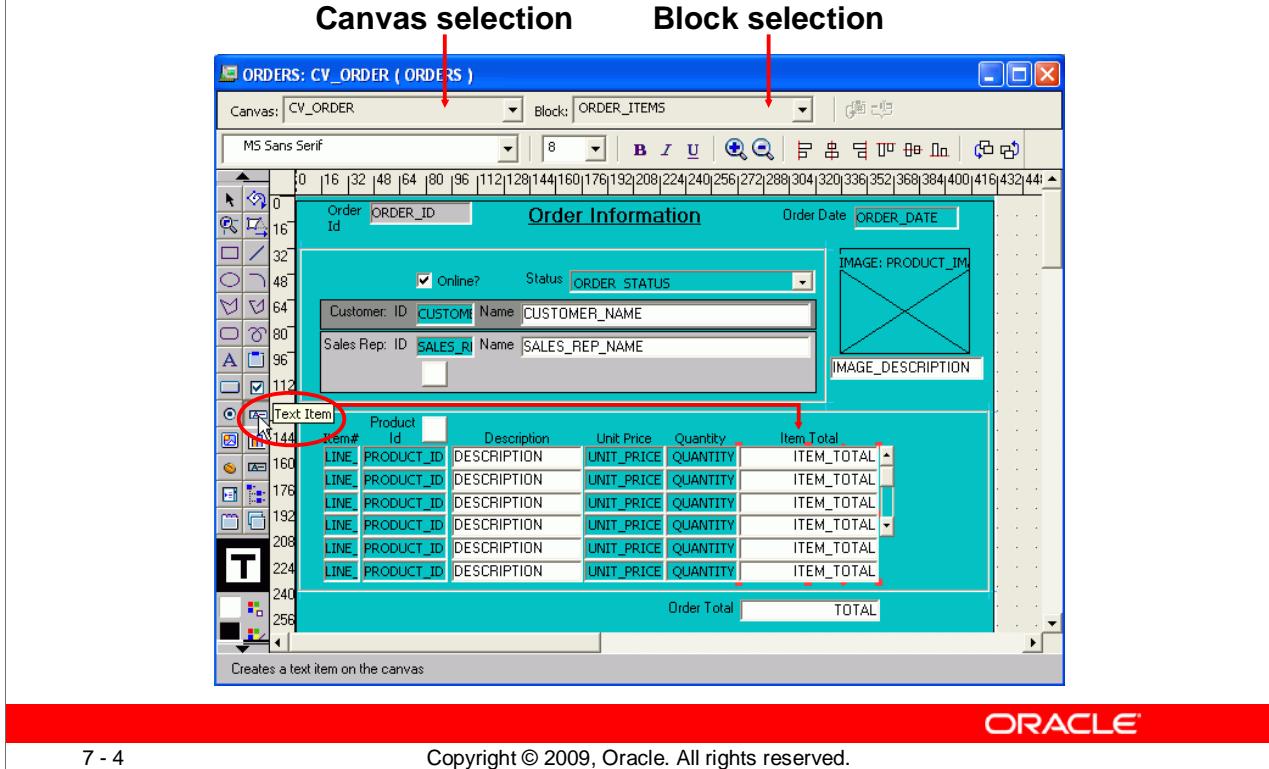
A text item is an interface object with which you can query, insert, update, and delete data. A text item usually corresponds to a column in the database table. When an item is first created, its default type is text.

The item type determines the properties available in the Property Palette. In this lesson, you look at the properties of a text item. The remaining item types are covered in subsequent lessons.

Use the Property Palette to define, alter, or examine the characteristics of items.

The screenshot in the slide shows the tool for creating a text item in the Layout Editor.

Creating a Text Item



Creating a Text Item

You can create a text item by doing one of the following:

- Converting an existing item into a text item
- Using the Text Item tool in the Layout Editor
- Using the Create icon in the Object Navigator
- Using the wizards

How to Create a Text Item in the Layout Editor

1. Invoke the Layout Editor, as shown in the screenshot.

It is important to point to the correct data block where you want to create the text item. In the Layout Editor, select the data block from the Block pop-up list.

2. Click the Text Item tool.

3. Click the canvas.

The text item appears.

4. Double-click the text item.

The text item Property Palette appears.

5. Set the item properties as required.

Creating a Text Item (continued)

How to Create a Text Item in the Object Navigator

1. Locate the block in which you want to create the item.
2. Select the Items node.
3. Click the Create icon.

A new item entry is displayed in the Object Navigator.

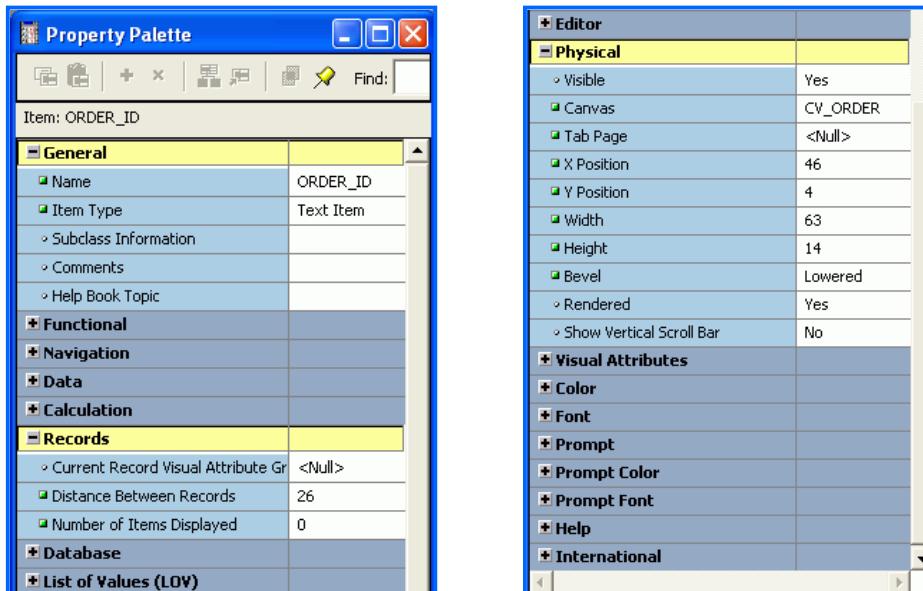
4. Double-click the icon to the left of the new item entry.

The Property Palette appears.

5. Set all item properties as required, keeping the Type property set to Text Item.

Note: To display an item at run time, you must assign the item to a canvas. Do this in the Property Palette of the text item by setting the Canvas property to the desired canvas.

Modifying the Appearance of a Text Item: General and Physical Properties



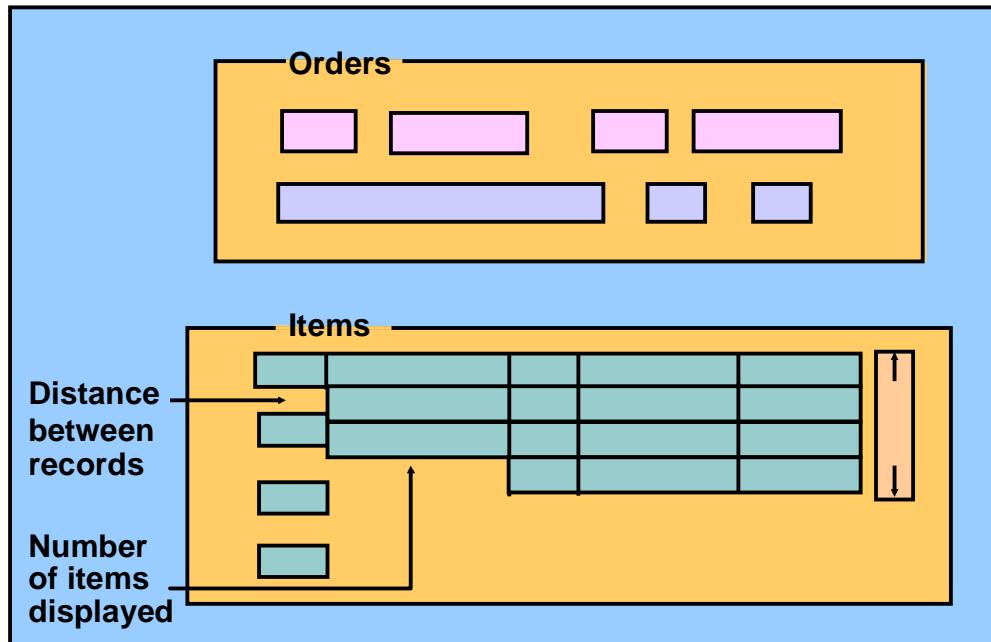
Modifying the Appearance of a Text Item: General and Physical Properties

The properties of an item are divided into several groups, as shown in the screenshots of the Property Palette in the slide.

You can affect the way the text item is displayed by altering its General, Physical, Records, Font, and Color group properties. To view descriptions of any of these properties, click the property in the Property Palette and press F1. The following are some of the properties:

- **General:**
 - **Item Type:** Selects the type of item you want to create (pop-up list)
- **Physical:**
 - **Visible:** Determines whether the item is displayed
 - **Canvas:** Determines the canvas on which the item is displayed. If left unspecified, the item is said to be a *Null canvas* item, and will not display at run time or in the Layout Editor.
 - **X and Y Position:** Sets the x and y coordinates of the item relative to the canvas
 - **Width and Height:** Sets the width and height of the item in the current form coordinate units
 - **Bevel:** Controls appearance of bevel around the item; can also be set to Plain (flat) or None

Modifying the Appearance of a Text Item: Records Properties



ORACLE®

7 - 7

Copyright © 2009, Oracle. All rights reserved.

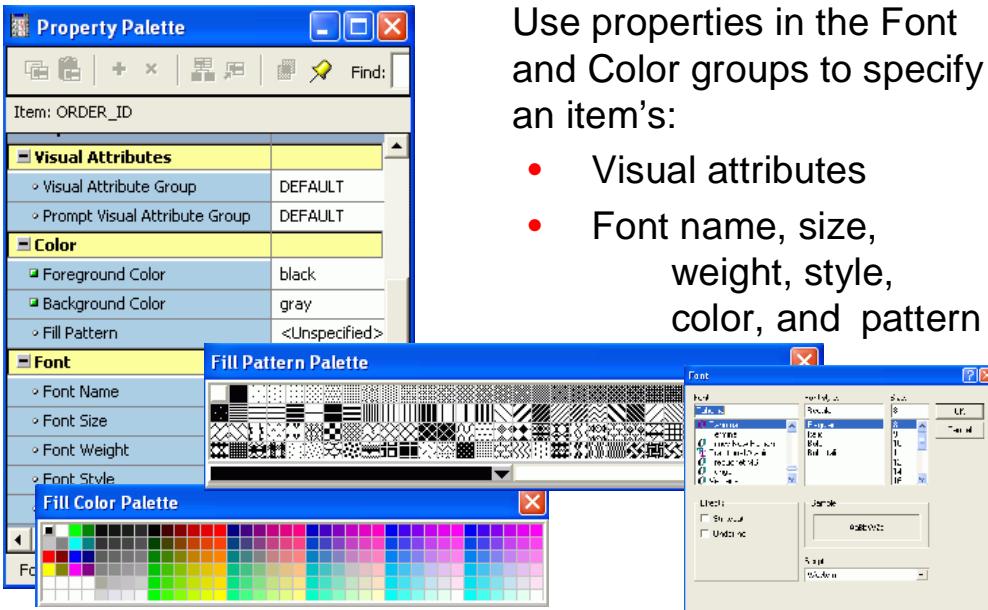
Modifying the Appearance of a Text Item: Records Properties

- **Records:**

- **Current Record Visual Attribute Group:** Specifies the name of the visual attribute to use when the item is part of the current record
- **Distance Between Records:** Specifies the amount of space between instances of the item in a multirecord block
- **Number of Items Displayed:** Specifies the number of item instances that are displayed for the item when the item is in a multirecord block

The graphic shows four records displayed in the Items block. The first item has a nonzero value for the Distance Between Records property, showing that you can specify this setting on a per-item basis.

Modifying the Appearance of a Text Item: Font and Color Properties



Use properties in the Font and Color groups to specify an item's:

- Visual attributes
- Font name, size, weight, style, color, and pattern

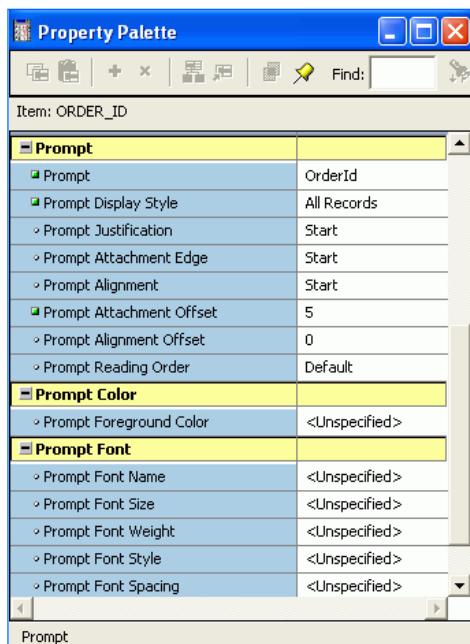
Modifying the Appearance of a Text Item: Font and Color Properties

- **Visual Attributes:** You set the Visual Attribute Group for an item or its prompt to specify how the visual attributes are derived (select DEFAULT or a named Visual Attribute).
- **Color:** The properties in this section specify the foreground color, background color, and fill pattern for the item. You select these from color and pattern pickers.
- **Font:** The properties in this section determine the font used for the item, along with its size, weight, style, and spacing. You can double-click the Font group to display a Font dialog box enabling you to set all these properties at once, or you can click the individual properties to select each from an appropriate control, such as a pop-up list or list of values (LOV).

The screenshots show the Property Palette with the fill pattern and fill color pickers and the font dialog box.

Note: When the form module does not contain any named Visual Attribute objects, the pop-up list for the Visual Attribute Group property shows only DEFAULT or unspecified. An item that has the Visual Attribute Group property set to DEFAULT, or that has individual attribute settings left unspecified, inherits those settings from the canvas to which it is assigned.

Modifying the Appearance of a Text Item: Prompts



- A prompt specifies the text label that is associated with an item.
- Several properties are available to arrange and manage prompts.
- Use prompt properties to change the appearance of an item prompt.

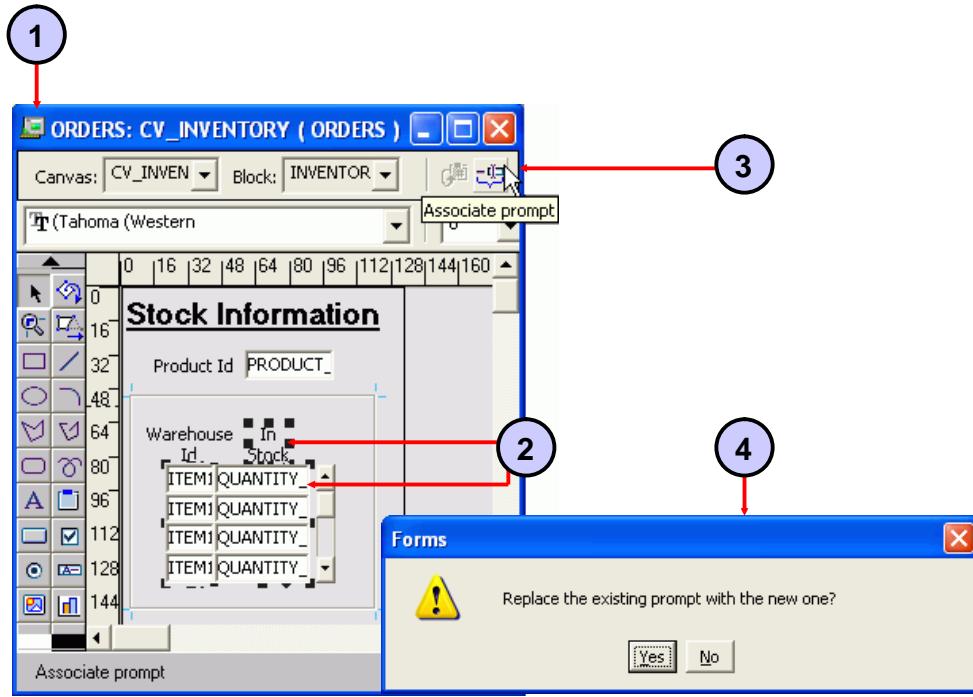
Modifying the Appearance of a Text Item: Prompts

You can control the appearance of the prompt, or label, of a text item using properties in the following groups:

- **Prompt:** You can set the prompt text and other properties, such as:
 - **Display Style:** Pop-up list with choices of First Record, Hidden, and All Records
 - **Attachment Edge:** Specifies the item edge to which the prompt is attached
 - **Attachment Offset:** Specifies the distance between the item and its prompt
- **Prompt Color:** The property in this section specifies the foreground color for the item prompt. You can select this from a color picker.
- **Prompt Font:** The properties in this section determine the font that is used for the item prompt, along with its size, weight, style, and spacing. You can double-click the Prompt Font group to display a Font dialog box, which enables you to set all these properties at once. Alternatively, you can click the individual properties to select each from an appropriate control, such as a pop-up list or LOV.

The screenshot shows the Property Palette for a text item with the Prompt groups highlighted.

Associating Text with an Item Prompt



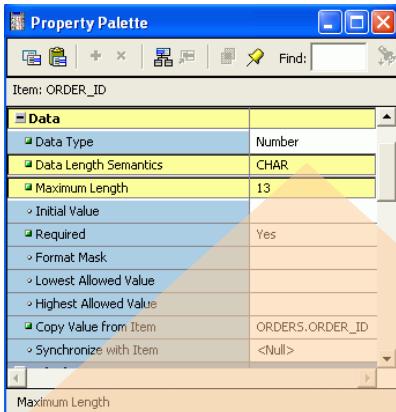
Associating Text with an Item Prompt

The Forms Builder Layout Editor has a tool called Associate Prompt, which enables you to create a prompt for an item using any boilerplate text displayed in the editor. To create a prompt-item association using the Associate Prompt tool, perform the following steps:

1. Open the Layout Editor window.
2. Select the item and boilerplate text you want as the item's prompt in the editor.
3. Click the Associate Prompt tool.
4. If you are replacing an existing prompt, click Yes in the dialog box.

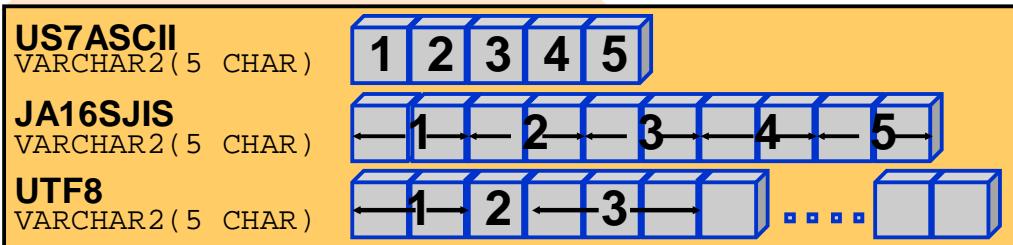
The screenshot in the slide shows the Layout Editor with a text item and boilerplate text selected. The Associate Prompt tool has been selected. A dialog box appears asking if the existing prompt should be replaced.

Controlling the Data of a Text Item



Use the properties in the Data group to control the data:

- Type
- Length
- Format
- Value



Controlling the Data of a Text Item

The properties in the Data group of the text item are used to control the way data is displayed and entered. You can see descriptions of any of these properties by clicking the property in the Property Palette, and then by pressing F1. The following are some of the properties:

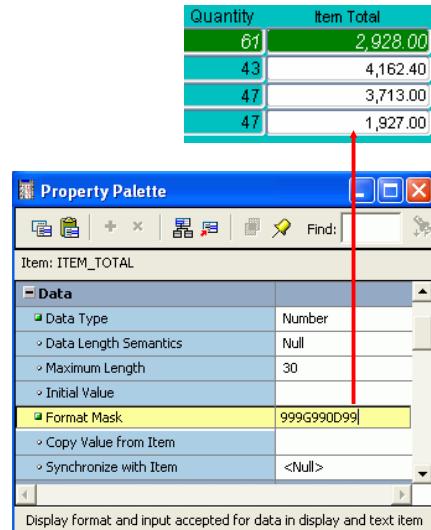
- **Data Type:** This enables you to choose CHAR, DATE, DATETIME, and NUMBER; the others listed are for backward compatibility.
- **Data Length Semantics:** You can set to Null, BYTE, or CHAR to be compatible with multiple character sets. If Data Length Semantics is CHAR, the correct amount of storage will be automatically allocated as required for the Maximum Length with either a single-byte or multibyte character set.
- **Maximum Length:** You can use this to specify the maximum length of the data value that can be stored in an item. If the Maximum Length exceeds the display width of the item, Forms automatically enables the end user to scroll horizontally.
Note: In the example in the slide, whether the form operator is using a single-, double-, or variable-byte character set, the right amount of storage is allocated. To hold the same value if the Data Length Semantics had been set to BYTE, the Maximum Length would have needed to be 5 for single-byte, 10 for double-byte, and an unknown value for a variable-byte character set.

Controlling the Data of a Text Item: Format

Format masks:

- Standard SQL formats, such as:
 - Dates FXDD-MON-RR
 - Numbers L099G990D99
- Nonstandard formats: Use double quotation marks for embedded characters,
" (" 099 ") " 099 " – " 0999.

Note: Allow for format mask's embedded characters when defining the Width property.



Controlling the Data of a Text Item: Format

- **Format Mask:** You can specify any format mask that is valid for the data type. Use the Format Mask property to specify the format in which the user sees the item value.
 - Use standard SQL formatting syntax for dates and numbers (for example, DD/MM/RR and \$99 ,999 . 99). The screenshots show that a number such as 4162 . 4 with a format of 999G990D99 looks like 4 ,162 . 40 at run time.
 - Enclose non-SQL-standard embedded characters within double quotation marks (for example, hyphen [-] and comma [,]).

It is recommended that you avoid creating individual masks if the general purpose masks suffice (see the lesson titled “Working in the Forms Environment”).

- **FX Format Mask:** The FX format mask in a date value ensures that the date is entered exactly as defined. Element D is for decimal and G is a group (thousands) separator.
Example: With a date format of DD/MM/RR, valid entries are 10/12/00, 10 12 00, 10-DEC-00, or 101200. You can enter any character to represent the (/) in the value. Allow for the embedded characters of the format mask when defining the Width property. Embedded characters are used only for display purposes and are not stored. However, if the date format mask is FXDD/MM/RR, that same date must be entered as 10/12/00.

Note: For a complete list of format elements, see *Oracle Database SQL Language Reference 11g Release 1 (11.1)*, available on OTN.

Controlling the Data of a Text Item: Values

Initial values:

- Are used for every new record
- Can be overwritten
- Must be compatible with item's data type
- Use:
 - Raw value
 - System variable
 - Global variable
 - Form parameter
 - Form item
 - Sequence

ORACLE®

7 - 13

Copyright © 2009, Oracle. All rights reserved.

Controlling the Data of a Text Item: Values

- **Required:** This specifies whether Forms will allow the item to have a null value. When you create a data block, Forms derives this value from the existence of a NOT NULL constraint on the database column, but you can change the value.
- **Lowest/Highest Allowed Value:** You can use this to specify the range of accepted values.
- **Initial Value:** This specifies the default value assigned to an item whenever a record is created; can be set to select from a sequence; must be compatible with the item data type. If the Lowest/Highest Allowed values are specified, the initial value cannot be outside the range.

Controlling the Data of a Text Item: Values (continued)

Creating an Initial Value

You can use any one of the following values to issue an initial item value whenever a new record is created:

- Raw value

Example: 340, RICHMOND

- System variable

- Variables giving current application server *operating system* date/time:

Variable	Format
\$\$DATE\$\$	DD-MON-RR
\$\$DATETIME\$\$	DD-MON-YYYY hh:mi[:ss]
\$\$TIME\$\$	hh:mi[:ss]

- Variables giving current *database* date/time:

Variable	Format
\$\$DBDATE\$\$	DD-MON-YYYY
\$\$DBDATETIME\$\$	DD-MON-YYYY hh:mi[:ss]
\$\$DBTIME\$\$	hh:mi[:ss]

- Global variable

Example: :GLOBAL.CUSTOMER_ID

- Form parameter

Example: :PARAMETER.SALES_REP_ID

- Form item

Example: :ORDERS.ORDER_ID

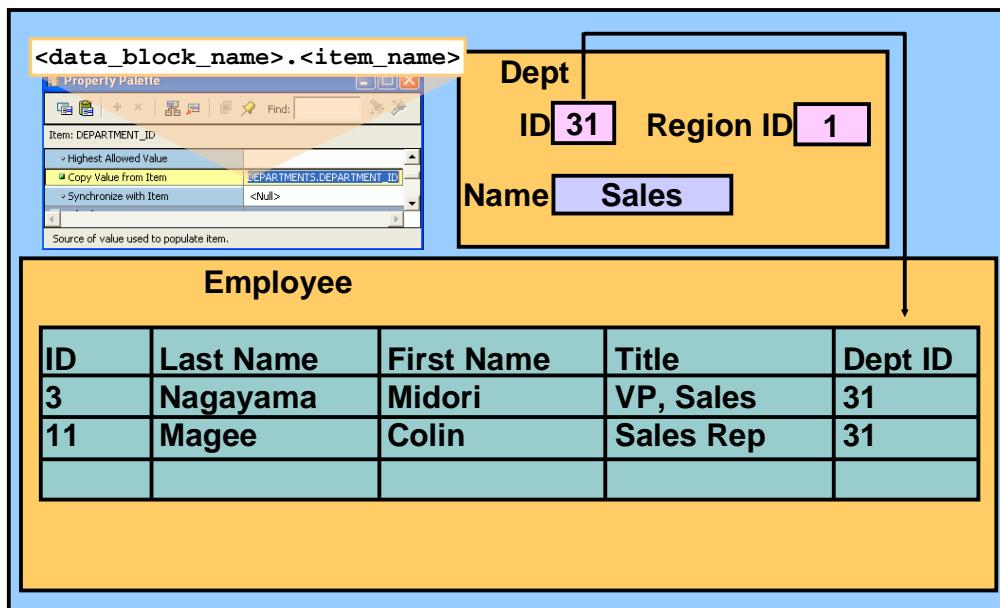
- Sequence

The initial value can reference a sequence in the database. Forms automatically writes generated sequence numbers into the text item. To use a sequence, enter:

:SEQUENCE.<sequence name>.NEXTVAL

Example: :SEQUENCE.ORDERS_SEQ.NEXTVAL

Controlling the Data of a Text Item: Copy Value from Item



ORACLE®

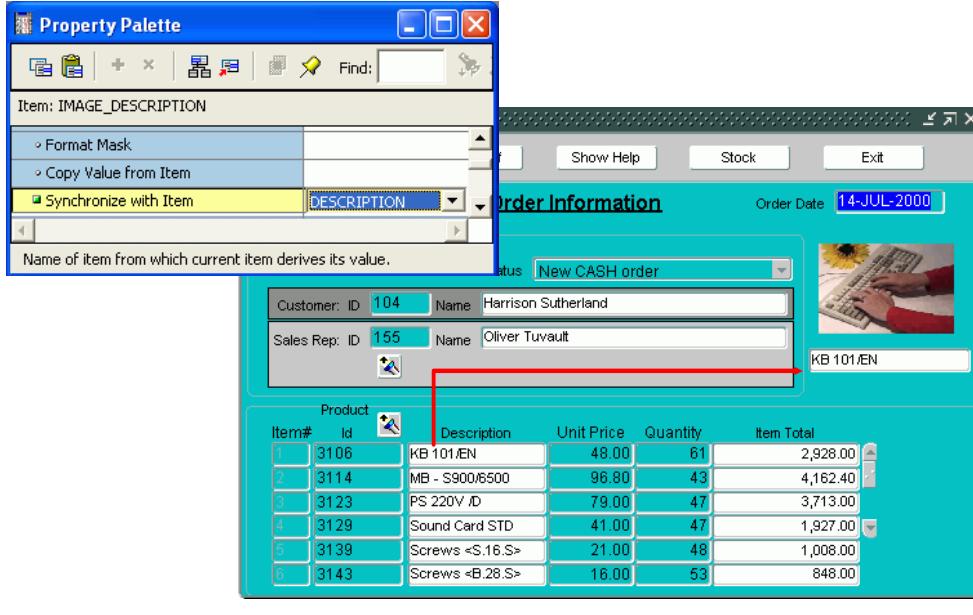
Controlling the Data of a Text Item: Copy Value from Item

- **Copy Value from Item:** This specifies the source of the value that Forms uses to populate the item. When you define a master-detail relation, Forms Builder sets this property automatically on the foreign key items in the detail block. In such cases, the “Copy Value from Item” property names the primary key item in the master block whose value gets copied to the foreign key item in the detail block whenever a detail record is created or queried.

Note: The text item should disable input; otherwise, the user can violate the foreign key relationship. To prevent this, set the Enabled property to No for the foreign key item, or do not display it at all.

The example in the slide shows that the department ID item on the Employee block (detail) has “Copy Value from Item” set to DEPARTMENTS . DEPARTMENT_ID from the master block.

Controlling the Data of a Text Item: Synchronize with Item



ORACLE®

7 - 16

Copyright © 2009, Oracle. All rights reserved.

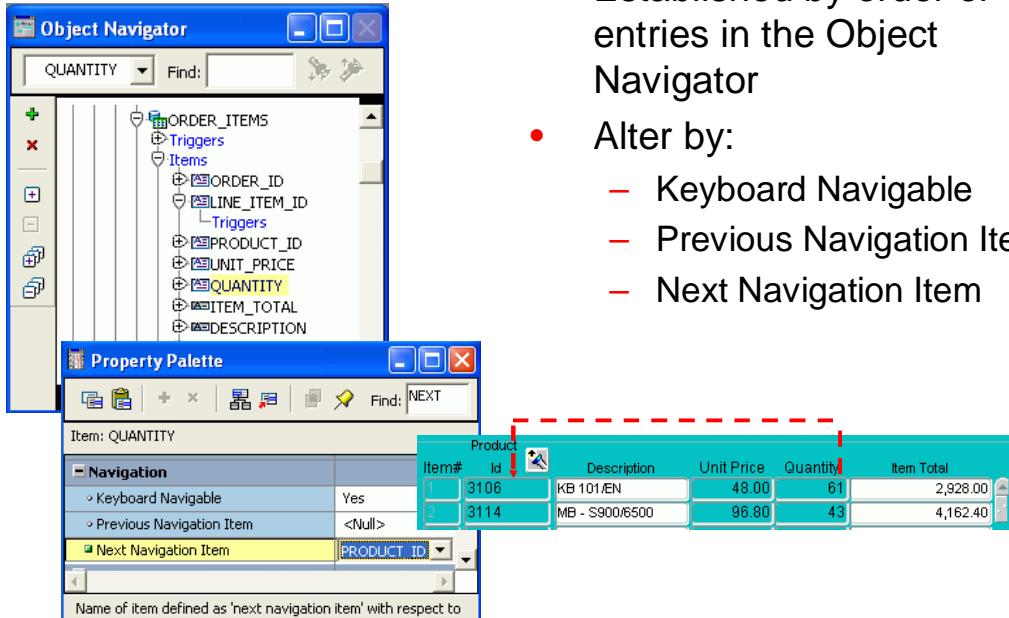
Controlling the Data of a Text Item: Synchronize with Item

- **Synchronize with Item:** You can use this property to specify the name of the item from which the current item should derive its value and to synchronize the values of the two items, so that they effectively mirror each other. When the end user or the application changes the value of either item, the value of the other item also changes.

The screenshots show that the caption of the product picture, which is on the control block, is synchronized with the product description in the data block.

Altering Navigational Behavior of Text Items

- Established by order of entries in the Object Navigator
- Alter by:
 - Keyboard Navigable
 - Previous Navigation Item
 - Next Navigation Item



Altering the Navigational Behavior of Text Items

You can see the default navigational sequence of items in the Object Navigator because the item entries are displayed in the navigational order. However, you can also use the Navigation group properties to control the navigational behavior of a text item.

Navigation Property	Function
Keyboard Navigable	Determines whether you can navigate to an item during default navigation with the function keys or menu items and place focus on it. When this property is set to No, Forms skips over the item and enters the next navigable item in the default navigation sequence.
Previous Navigation Item	Determines the previous item to be visited when you navigate out of the current item
Next Navigation Item	Determines the next item to be visited when you navigate out of the current item

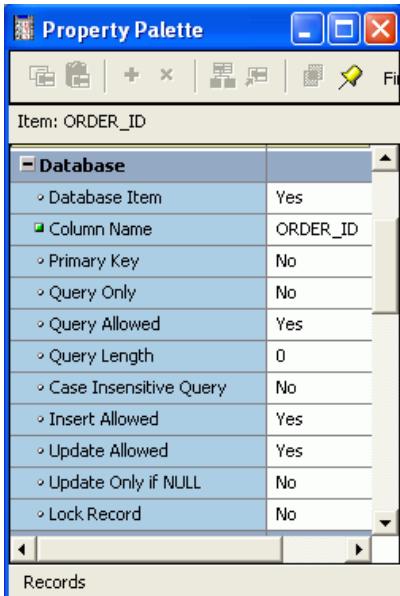
The slide graphic shows setting Next Navigation Item for the Quantity item to navigate to the Product_Id item.

Note: The next or previous navigation item must be in the same block as the current item.

Enhancing the Relationship Between Text Item and Database

Use the properties in the Database group to control:

- Item's data source—base table item or control item
- Query, insert, and update operations on an item
- Maximum query length
- Query case



ORACLE®

Enhancing the Relationship Between Text Item and Database

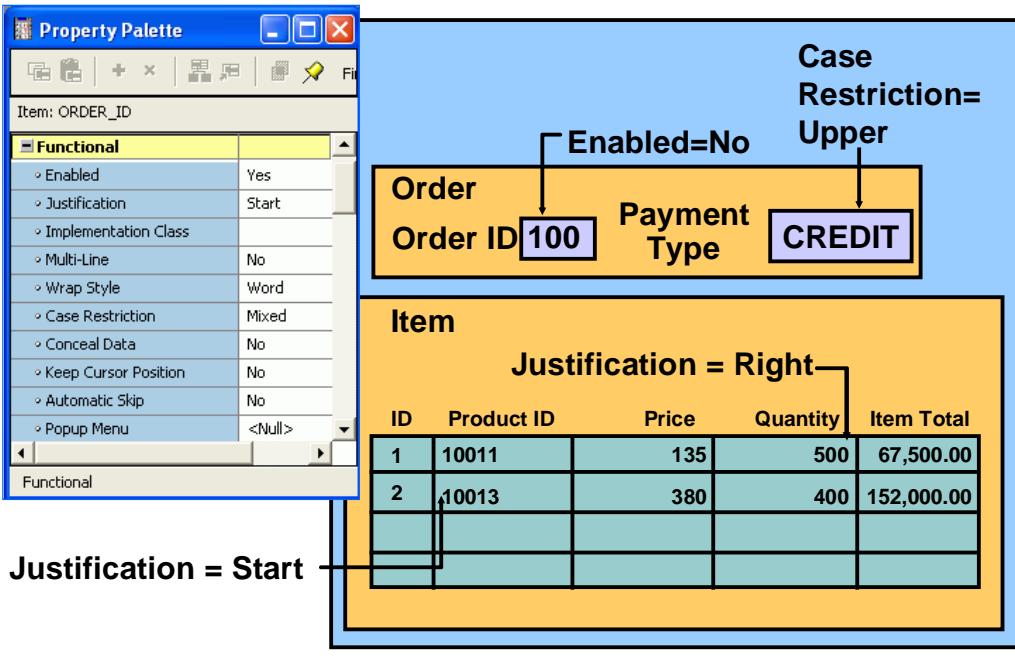
You can alter or enhance the way in which a text item interacts with its corresponding database column by setting the Database group properties. Some of these are:

- **Database Item:** Indicates whether the item is a database column
- **Query/Insert/Update Allowed:** Controls whether data manipulation language (DML) operations are allowed
- **Query Length:** Specifies the maximum length of query criterion in Enter Query mode
- **Case Insensitive Query:** Controls whether case is recognized in query processing

The screenshot shows the Database group on the Property Palette for a text item.

Note: When you create an item in a data block, Forms Builder assumes that the item is a data item, sets its Database Item property to Yes, and automatically includes it in any SELECT, UPDATE, and INSERT statements issued to the database. If an item that you are creating is a control item, you must explicitly set its Database Item property to No.

Adding Functionality to a Text Item



Adding Functionality to a Text Item

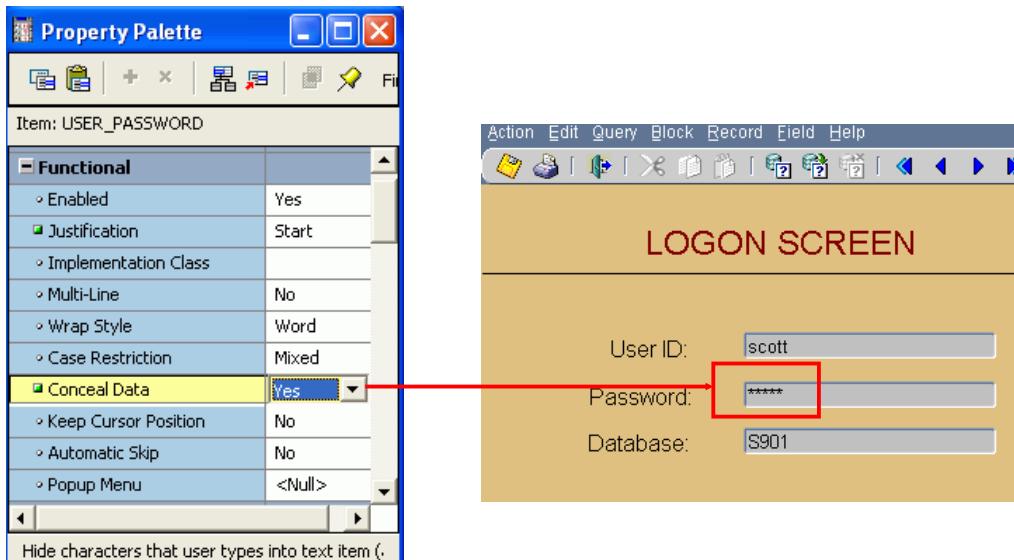
Augment the default functionality of a text item by introducing some of the additional features you can set in the Functional group of the Property Palette. Some of these are depicted in the slide or will be discussed in the subsequent slides. For descriptions of other properties in the Functional group, select the property in the Property Palette and press F1.

The screenshot shows the Functional group in the Property Palette for a text item. The graphics illustrate how setting the Enabled, Case Restriction, and Justification properties affect the functionality of the text item at run time.

Note

- If the Enabled property set to No, the item is disabled. If you want the item to appear normally but do not want users to change it, do the following:
 - Set Insert Allowed to No.
 - Set Update Allowed to No.
 - Set Enabled to Yes.
- A pop-up menu is a context-sensitive menu that enables users to access common functions and commands quickly. It is a top-level object in the Object Navigator and belongs to a form module (as opposed to a form menu, which belongs to a separate menu module).

Adding Functionality to a Text Item: Conceal Data Property



Adding Functionality to a Text Item: Conceal Data Property

Conceal Data: Hides characters that the operator types into the text item. This setting is typically used for password protection. Select Yes to disable the echoing back of data entered by the operator; with this setting, the entered value is displayed as an asterisk for each character entered as shown in the slide.

Note: Conceal Data set to Yes is valid only for single-line text items.

Adding Functionality to a Text Item: Keyboard Navigable and Enabled

The Keyboard Navigable and the Enabled properties work in concert with one another:

- Set both properties to allow or disallow navigation and interaction with text item.
- When Enabled is set to Yes, Keyboard Navigable can be set to Yes or No.
- When Enabled is set to No, the item is always nonnavigable.

ORACLE®

7 - 21

Copyright © 2009, Oracle. All rights reserved.

Adding Functionality to a Text Item: Keyboard Navigable and Enabled

You can set the Keyboard Navigable and Enabled properties for items to specify whether operators can navigate to and interact with them. The Enabled property determines whether end users can use the mouse to manipulate an item. The following table describes the behavior of combinations of these settings:

Enabled	Keyboard Navigable	Navigation Behavior
Yes	Yes	Item is included during default navigation. The item can be navigated to and manipulated with the mouse.
Yes	No	Item is excluded during default navigation. The item can be navigated to and manipulated with the mouse.
No	No	Item is excluded during default navigation. The item cannot be navigated to and manipulated with the mouse.
No	Yes	Item is excluded during default navigation. The item cannot be navigated to and manipulated with the mouse. The Keyboard Navigable property is also effectively set to No.

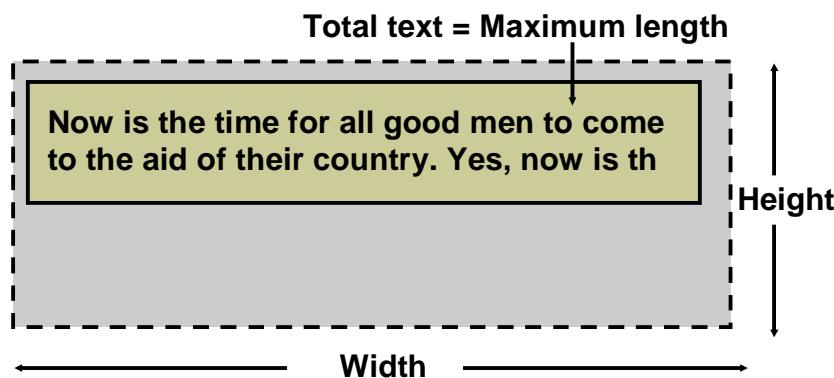
Adding Functionality to a Text Item: Multi-Line Text Items

Multi-Line:

- Possible values: Yes/No
- Cannot use for numeric or date values
- Should also set width, height, font size, maximum length

Wrap Style possible values:

- None
- Character
- Word



Adding Functionality to a Text Item: Multi-Line Text Items

Multi-Line: Determines whether the text item displays in a single-line or multi-line region. Use multi-line text items to display and/or edit such items as addresses, comments, or descriptions. The data in a multi-line text item must be of Char, Alpha, or Long data type; not numeric or date.

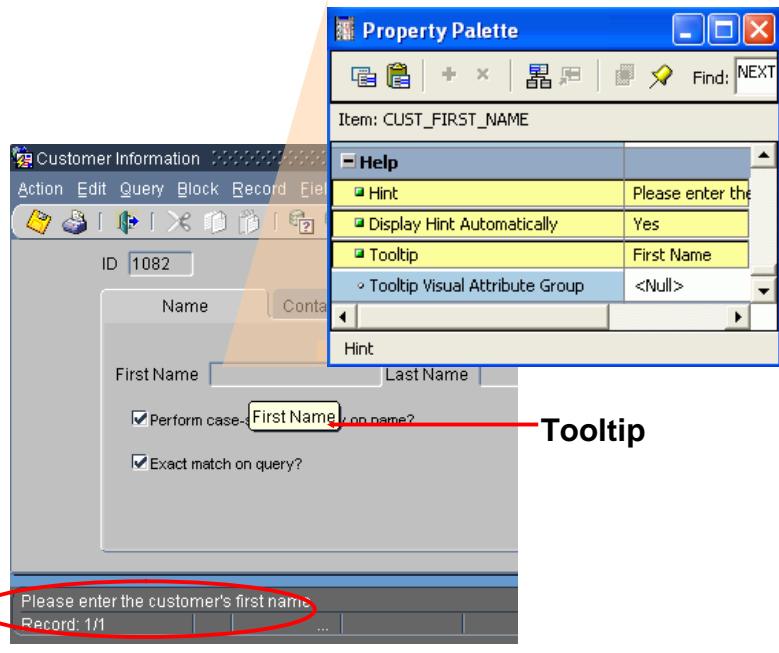
Setting the Multi-Line property to Yes enables a text item to store multiple lines of text, but it does not automatically make the item large enough to display multiple lines. You can set the Width, Height, Font Size, and Maximum Length properties to ensure that the desired number of lines and characters are displayed.

The graphic in the slide illustrates that even if the height and width of the text item enable you to enter a lot of text, the maximum length limits the length of the value that can be stored in the text item. When a value that is too long for the text item is retrieved from the database, truncation occurs.

Wrap Style: For multi-line text items, specifies how text is displayed when a line of text exceeds the width of a text item or editor window

Note: Setting right or center justification for scrollable text items may result in values being hidden from the user.

Displaying Helpful Messages: Help Properties



7 - 23

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Displaying Helpful Messages: Help Properties

You can use the Help group properties to provide context-sensitive help to users:

Help Property	Function
Hint	Writes item-specific Help text that is displayed on the message line at run time. The Help text is available when input focus is on the item.
Display Hint Automatically	Determines whether the hint for the item is displayed automatically. If set to No, the hint displays only when the operator clicks Help or selects the Help command on the default menu.
Tooltip	Help text that should appear in a small box beneath the item when the cursor moves over the item. The item does not need to have input focus for the tooltip to appear.
Tooltip Visual Attribute Group	Specifies Visual Attribute to use for the tooltip

The screenshots show the hint appearing at the bottom of the form when a user navigates to its text item, while the tool tip appears in the vicinity of its text item when the cursor moves over the text item at run time.

Summary

In this lesson, you should have learned that:

- Text items are interface objects that usually correspond to database columns
- You can create a text item with:
 - The Text Item tool in the Layout Editor
 - The Create icon in the Object Navigator
 - The Data Block Wizard



Summary

This lesson showed you how to create and modify a text item that Forms Builder creates for each column flagged for inclusion in a data block.

Summary

- You can modify a text item in its Property Palette:
 - General, Records, and Physical properties control the appearance of the text item
 - Data properties control the length, data type, format, and other aspects of the data
 - Navigation properties control how to navigate to and from a text item
 - Database properties specify the relationship between the text item and its corresponding database column
 - Functional properties control how the text item functions
 - Help properties specify the display of helpful messages

ORACLE®

7 - 25

Copyright © 2009, Oracle. All rights reserved.

Summary (continued)

In particular, text items have properties that enable you to do the following:

- Modify their appearance.
- Control the data stored in the item.
- Alter navigational behavior.
- Enhance the relationship with the database.
- Add functionality.
- Include Help information.

Practice 7: Overview

This practice covers the following topics:

- Deleting text items
- Modifying text item properties
- Creating text items



Practice 7: Overview

In this practice session, you will create text items, alter the behavior and the appearance of text items, and delete text items.

- Delete the Region_Id item in the Customers form.
- Using the Property Palette, change the properties of several text items in the CUSTOMERS data block to change their run-time appearance. Save and run the form after the changes are applied.
- In the Orders form, create new text items to hold the customer name and sales rep name values in the ORDERS block, and set the suggested properties. Change additional text item properties in the ORDERS, ORDER_ITEMS, and INVENTORIES data blocks to change their run-time appearance and behavior. Save and run the form after the changes are applied.

Creating LOVs and Editors

8

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

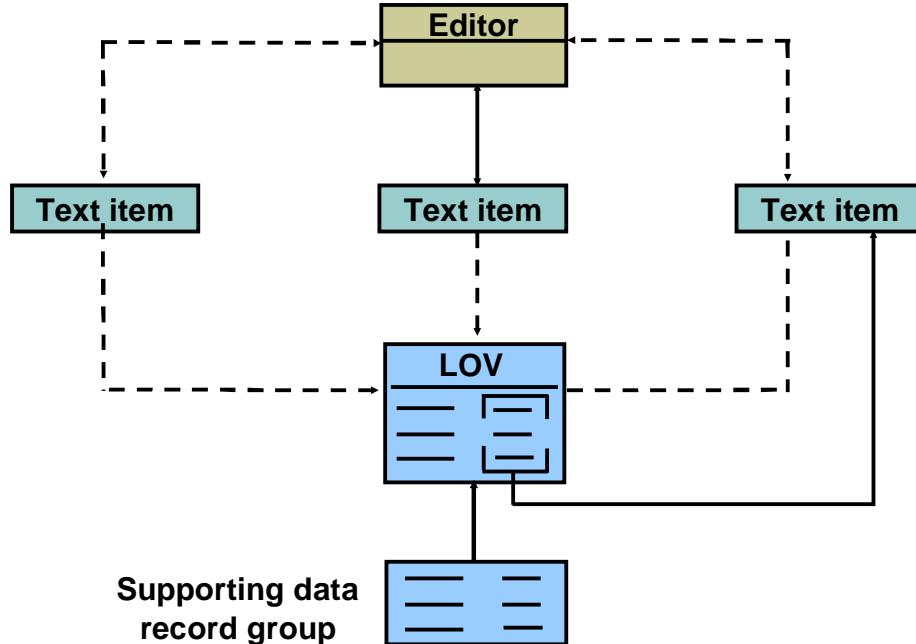
- Describe lists of values (LOVs) and editors
- Design, create, and associate LOVs with text items in a form module
- Create editors and associate them with text items in a form module



Lesson Aim

With Oracle Forms Builder, you can enhance your application with lists of available values and text editors to supplement the text item object. In this lesson, you learn how to create LOVs and text editors, and to associate them with items in your application.

Overview of LOVs and Editors



Overview of LOV and Editors

LOVs and editors are objects in a form module that you can associate with text items to enhance input. Each opens its own window when activated at run time.

LOVs enable users to choose a value from a static or dynamic list, while editors provide a larger area for text entry with search and replace capability. Both are defined at the form level, which means that you can use them to support text items in any block of the form module.

The slide graphics depict three text items on a form. The first and second items use the same editor, while the second and third items use the same LOV. The LOV takes its values from a supporting record group. In this lesson, you learn how to create these objects and attach them to text items.

What Are LOVs?

- List of values for text items
- Dynamic or static list
- Independent of single text items
- Flexible and efficient

ORACLE®

8 - 4

Copyright © 2009, Oracle. All rights reserved.

What Are LOVs?

An LOV is a scrollable pop-up window that enables a user to pick the value of an item from a multicolumn dynamic list. The user can reduce the lines displayed in the list by simple automatic reduction techniques, or by search strings.

Each line in an LOV can present several field values, with column headings above. You can design your LOV to retrieve some or all of the field values from the line chosen by the user and place them into form items.

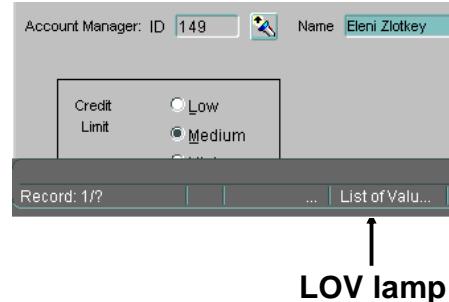
LOVs have the following qualities:

- **Dynamic:** The list entries can change to reflect changes in the source data.
- **Independent:** The designer can invoke an LOV from any text item, or from outside a text item if called programmatically.
- **Flexible:** You can use the same LOV to support several items, if appropriate (for example, product_ID, product_name).
- **Efficient:** You can design LOVs to reuse data already loaded into the form, instead of accessing the database for every call. This is useful where data is relatively static.

Using an LOV at Run Time

At run time:

- Status bar message (lamp)
- List to select value for data entry
- Automatic reduction
- Search



Using an LOV at Run Time

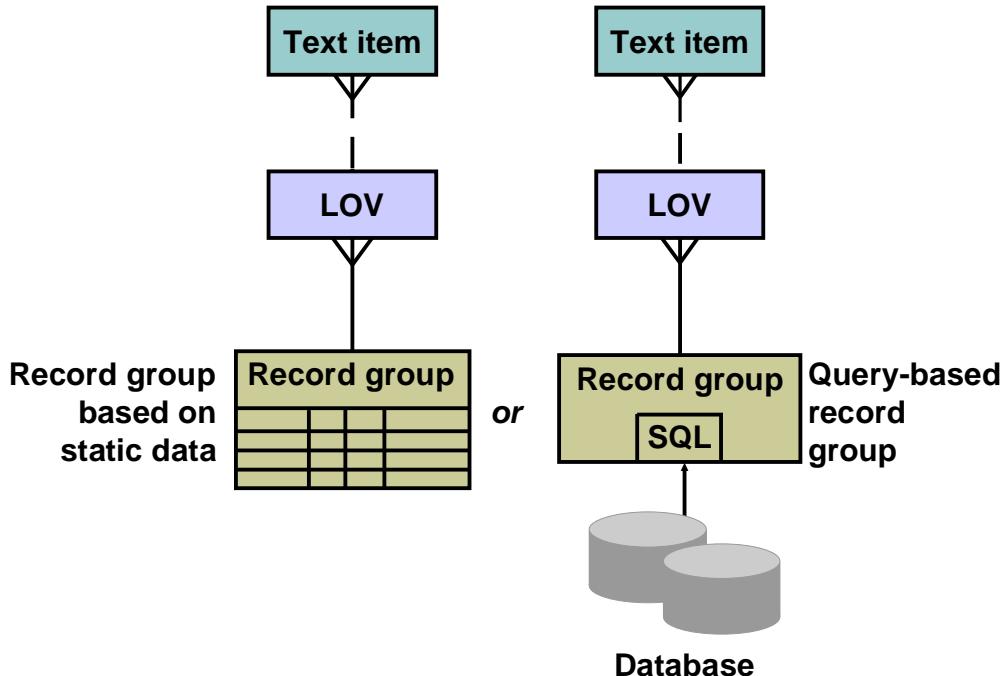
When a text item has an LOV attached, the List of Values lamp displays on the status line while the cursor is on the item, as shown in the screenshot.

To use the LOV:

1. Either click List of Values, or select Edit > Display List to invoke the LOV.
2. Select an entry in the displayed list. You can enter characters to automatically reduce the list, or enter a search string in the Find field.
3. Click OK to retrieve the line value.

Note: Automatic reduction works by comparing the search string entered with the values displayed in the first column of the LOV. If you start your search criteria with a % symbol, Forms performs a search on all LOV columns.

LOV-Related Objects



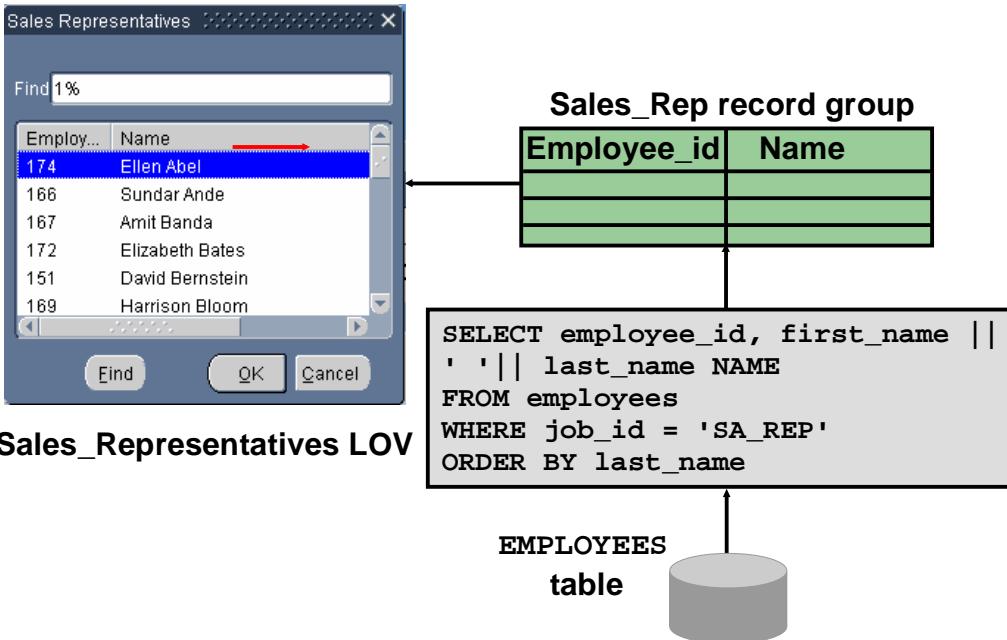
LOV-Related Objects

When you build an LOV, consider the following objects:

- **Record group:** A Forms Builder object that is used to store the array of values that are presented by an LOV (The record group can be created first or as part of the LOV creation process if based on a query.)
- **LOV:** The list itself, which presents one or more column values from the supporting record group in the LOV window (It enables the user to select values, and then write values back to specified items or variables.)
- **Text items:** The main text item that you attach to an LOV is usually one that the LOV returns a value to. You can call the LOV from this item to provide possible values for it. A single LOV can return values to several items. You can attach the LOV to any text item from which the same list of values needs to be viewed, whether or not it will receive a value.

The slide shows two types of LOVs. The left side of the slide shows a text item with an attached LOV that is based on a static record group. The right side of the slide shows a text item with an attached LOV that uses a SQL query to define the record group.

LOVs and Record Groups



8 - 7

Copyright © 2009, Oracle. All rights reserved.

LOVs and Record Groups

A record group is a column-and-row structure stored within Forms Runtime memory and is similar to the structure of a database table. It holds records that can be reused by other text items, therefore reducing repeated access to external data.

Record groups can be designed to contain static values. Alternatively, they can be populated programmatically at run time or, most commonly, populated by a SQL query. In this lesson, you use record groups to support LOVs.

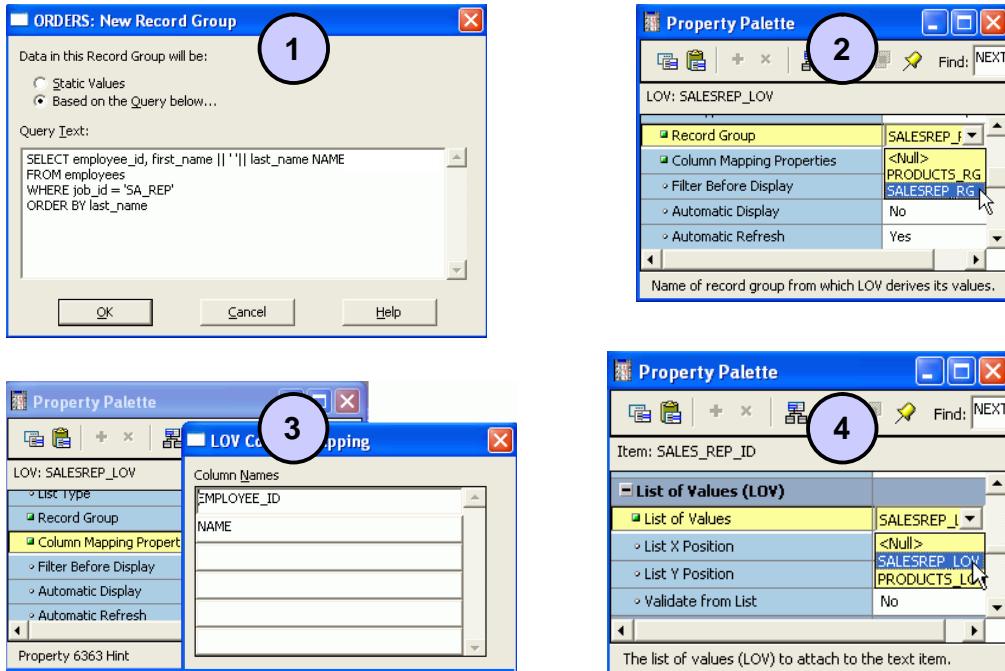
Record groups can provide the following:

- Data that is presented by LOVs
- Data for dynamic list items

Note: Because LOVs and record groups are separate objects, you can create multiple LOVs based on the same record group.

The screenshot shows an LOV for sales representatives that displays employee ID and name. The graphics in the slide show that the data for this LOV comes from a record group that is based on a SQL query, so is therefore dynamically retrieved from the database.

Creating an LOV Manually



8 - 8

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

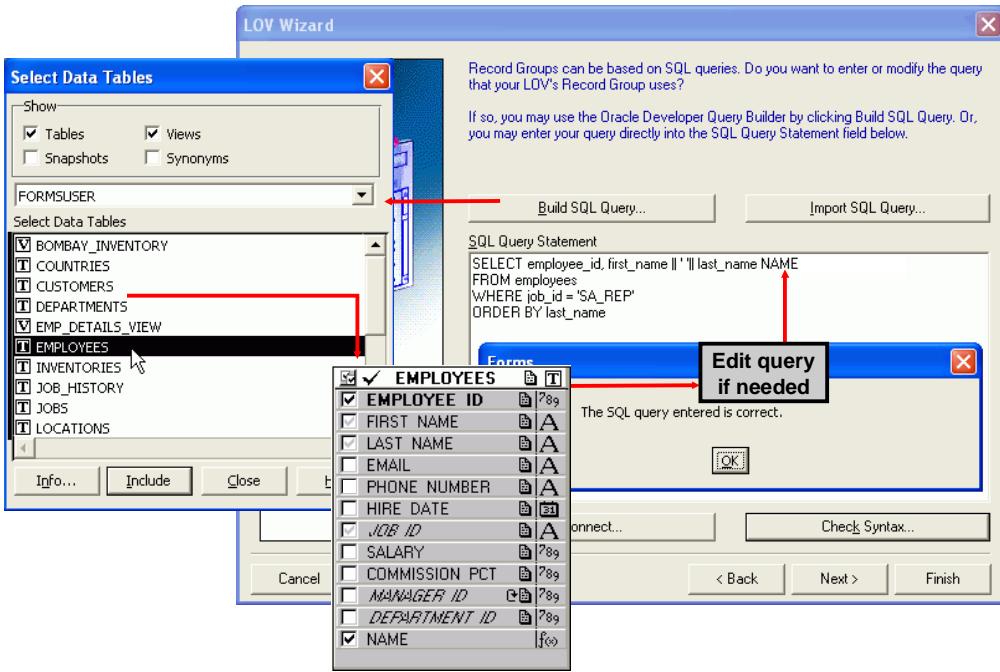
Creating an LOV Manually

Because Forms Builder has an LOV Wizard for you to use in creating LOVs and their associated record groups, you may never need to create an LOV manually. However, knowing how to do so helps you understand how to set the properties of the record group, the LOV, and the item to which it is attached, even if using the wizard.

The steps to create an LOV manually are the following:

1. Create the record group. You will need to enter the query on which the record group is based.
2. Create the LOV and set its Record Group property to the appropriate record group.
3. Set the LOV Column Mapping Property. You must enter the columns and their headings, and then select a return item for each item that you want to populate from the LOV.
4. Assign the LOV to any text items from which you want the LOV to be available.

Creating an LOV with the LOV Wizard: SQL Query Page



8 - 9

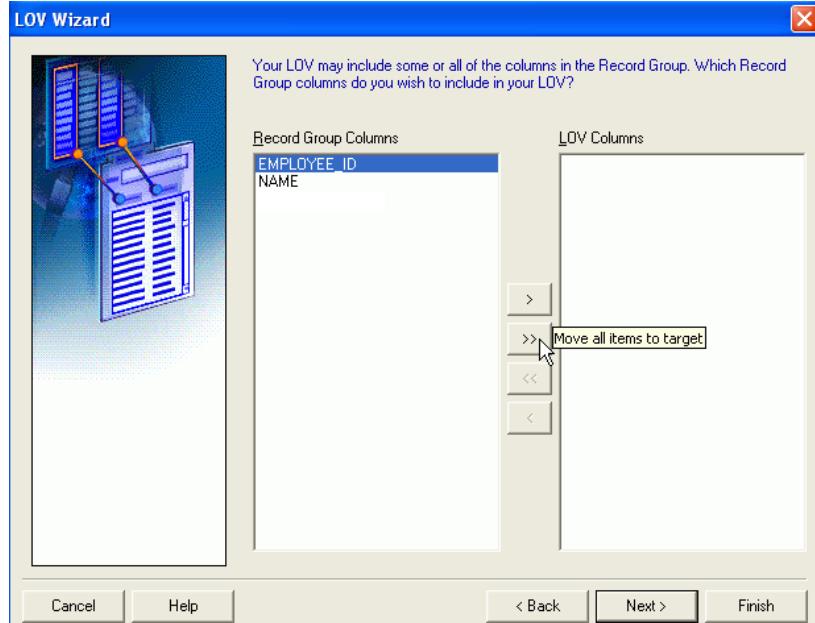
Copyright © 2009, Oracle. All rights reserved.

Creating an LOV with the LOV Wizard: SQL Query Page

It is easy to make a mistake or to forget one of the manual steps. This can be avoided by using the LOV Wizard, which guides you through the process. To create an LOV with the wizard, perform the following steps:

1. Launch the LOV Wizard.
The Welcome page appears. Click Next. The LOV Source page appears.
2. Specify the LOV source on the LOV Source page. Choose an existing record group or create a new one based on a query. The default option is New Record Group based on a query. Click Next to select the default. The SQL Query page appears.
3. On the SQL Query page, specify the query that is used for the record group. You cannot include a column of a complex object data type. Use one of the following three options for constructing the query:
 - Click Build SQL Query to use Query Builder, as shown in the slide. The EMPLOYEES table is used and the columns that are selected are the EMPLOYEE_ID column and a pseudocolumn called NAME, consisting of FIRST_NAME concatenated with LAST_NAME.
 - Click Import SQL Query to import the query from a file.
 - Enter the SQL syntax in the SQL Query Statement field to enter the query directly. Then click Check Syntax; a message about the validity of the query is displayed, as shown in the slide.

Creating an LOV with the LOV Wizard: Column Selection Page



ORACLE®

8 - 10

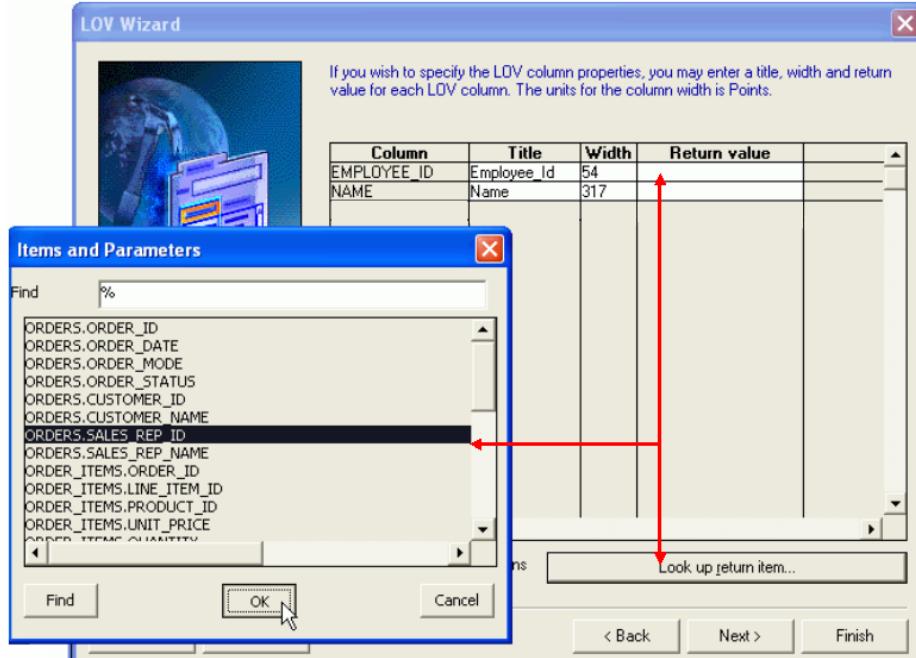
Copyright © 2009, Oracle. All rights reserved.

Creating an LOV with the LOV Wizard: Column Selection Page

4. On the Column Selection page, select the record group columns that you want to include in the LOV and shuttle them to the LOV Columns list, as shown in the slide. Click Next. The Column Properties page appears.

The slide shows that the wizard presents two record group columns that are available for inclusion in the LOV: EMPLOYEE_ID and NAME.

Creating an LOV with the LOV Wizard: Column Properties Page

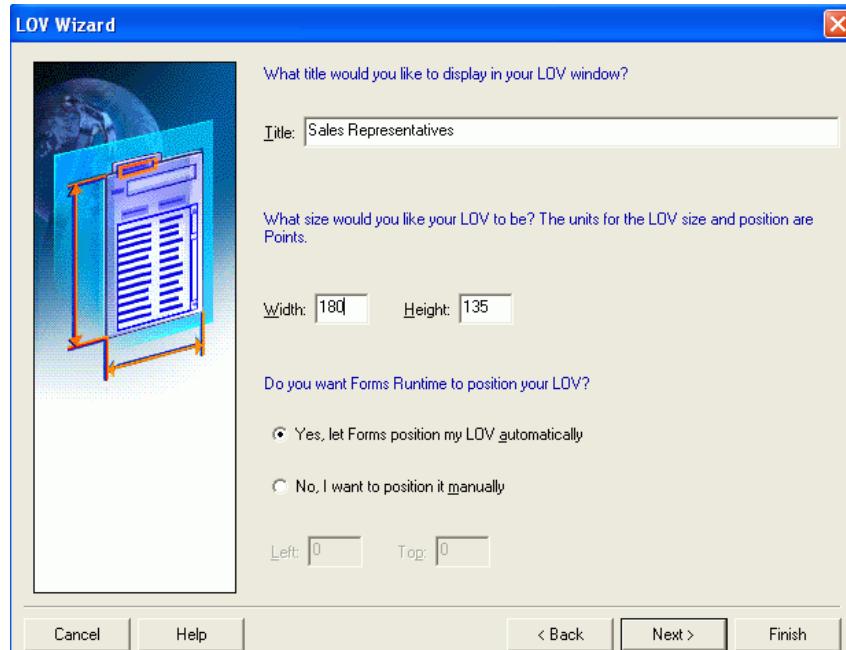


Creating an LOV with the LOV Wizard: Column Properties Page

5. On the Column Properties page, specify the Title, Width, and Return value for each LOV column. Note that Return value is optional and there is a button that invokes a separate window from which you can choose an item. Click Next. The LOV Display page is displayed.

The screenshots show how to set a Return value for Employee_Id. The “Look up return item” button was clicked to invoke a separate window, and the ORDERS . SALES _REP _ID item is selected. This means that, when the user selects an employee from the LOV, that employee’s ID populates the Sales_Rep_Id text item in the ORDERS block.

Creating an LOV with the LOV Wizard: LOV Display Page



8 - 12

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

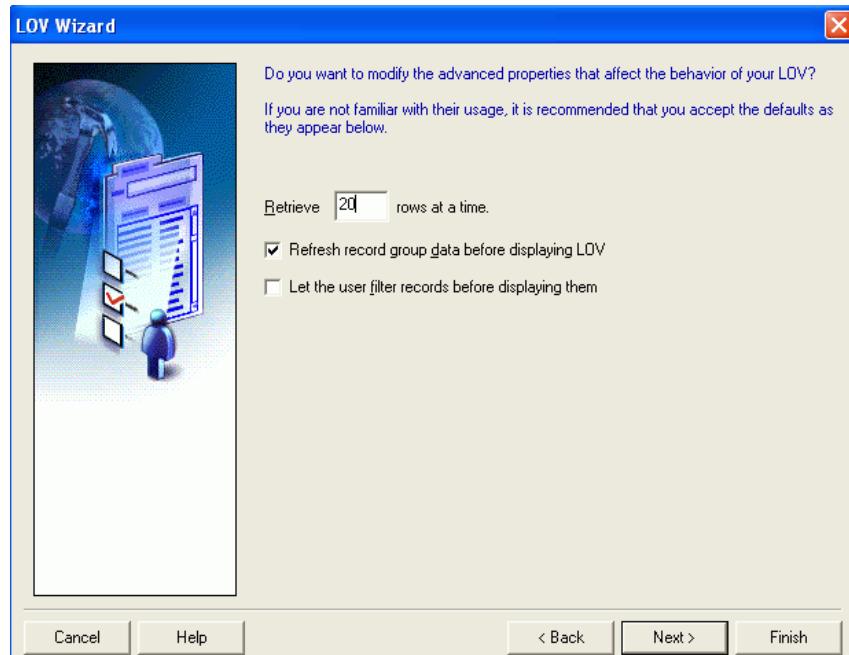
Creating an LOV with the LOV Wizard: LOV Display Page

6. On the LOV Display page, specify the title, the width, and the height of the LOV window. You can choose to display it at a set position that you manually define, or let Forms position it automatically. Click Next. The Advanced Options page appears.

In the slide, you see the screenshot of the LOV Display page of the wizard with the following selections:

- **Title:** Sales Representatives
- **Width:** 180
- **Height:** 135
- Automatic positioning option selected

Creating an LOV with the LOV Wizard: Advanced Properties Page



ORACLE®

8 - 13

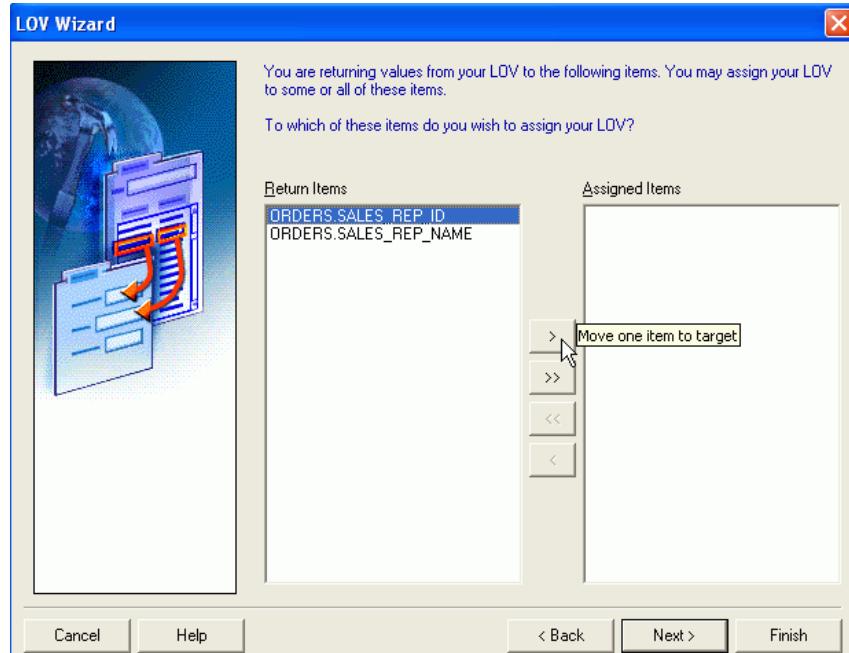
Copyright © 2009, Oracle. All rights reserved.

Creating an LOV with the LOV Wizard: Advanced Properties Page

7. On the Advanced Properties page, set the additional advanced properties for the LOV. Specify:
 - The number of records at a time to be fetched from the database; the example in the slide shows 20
 - If the LOV records should be queried each time the LOV is invoked (this check box is selected in the slide example)
 - If the user should be presented with a dialog box to add criteria before the LOV is displayed (this check box is not selected in the slide example)

Click Next. The “Assign to Item” page appears.

Creating an LOV with the LOV Wizard: “Assign to Item” Page



8 - 14

Copyright © 2009, Oracle. All rights reserved.

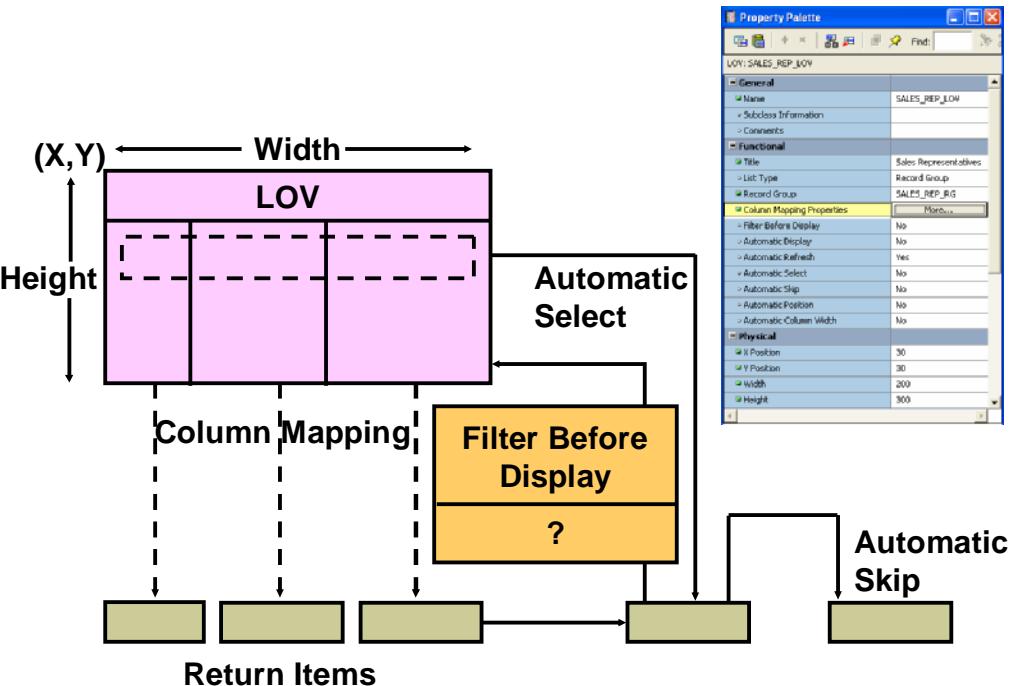
ORACLE®

Creating an LOV with the LOV Wizard: “Assign to Item” Page

8. On the “Assign to Item” page, select the items to which your LOV should be attached; the screenshot shows ORDERS . SALES _REP _ID selected in the available Return Items and that the user is about to shuttle that selection to the Assigned Items column. At run time, the LOV is available from the Sales_Rep_Id item so that operators may use it while input focus is in that item.
Click Next. The Finish page appears.
9. On the Finish page, click Finish to complete the LOV creation process.

Note: The LOV Wizard is reentrant, so you can use it to modify the LOV after it is created. In the Object Navigator, select the LOV to be modified, and then select Tools > LOV Wizard from the menu.

Setting LOV Properties

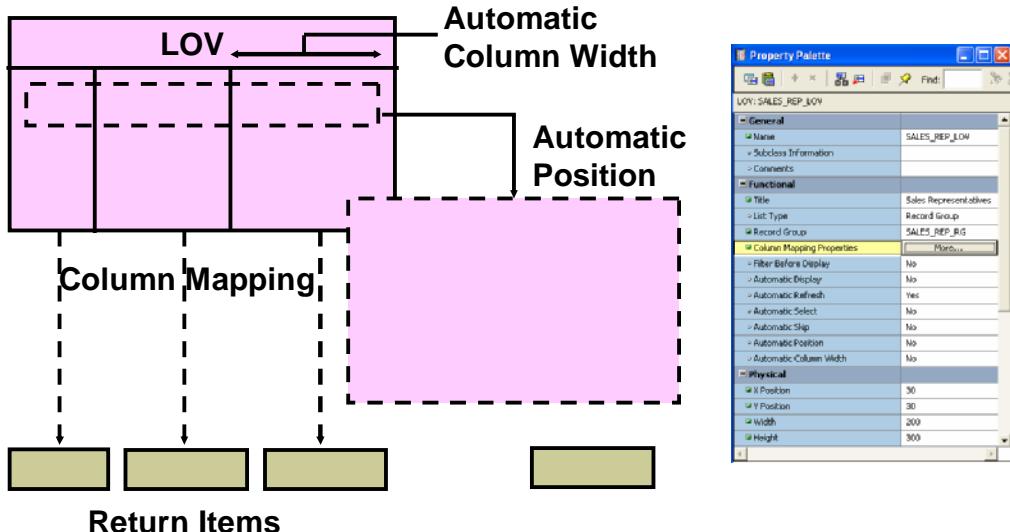


Setting LOV Properties

After you create an LOV, you can open its Property Palette to modify or further define its properties. Some of these properties are the following:

- **X and Y Position:** Specify screen coordinates for the LOV window in the form coordinate units
- **Width and Height:** Define size of the LOV window in the current form coordinate units; can be resized by the user
- **Column Mapping:** Click the button labeled More to open the LOV Column Mapping window; the slide graphic shows mapping LOV columns to return items on the form.
- **Filter Before Display:** Determines whether the user should be prompted with a dialog box that enables the user to enter a search value before the LOV is invoked; the value will be used as additional restriction on the first column in the query. The graphic in the slide shows the filter applied prior to displaying the LOV.
- **Automatic Display:** Controls whether the LOV should be invoked automatically when form operator enters an item to which the LOV is attached
- **Automatic Select:** Specifies if the LOV should close and return values automatically when reduced to single entry, as illustrated in the slide graphic
- **Automatic Skip:** Determines whether the cursor skips to the next navigable item when the operator selects a value from the LOV to populate the text item, as illustrated in the slide graphic

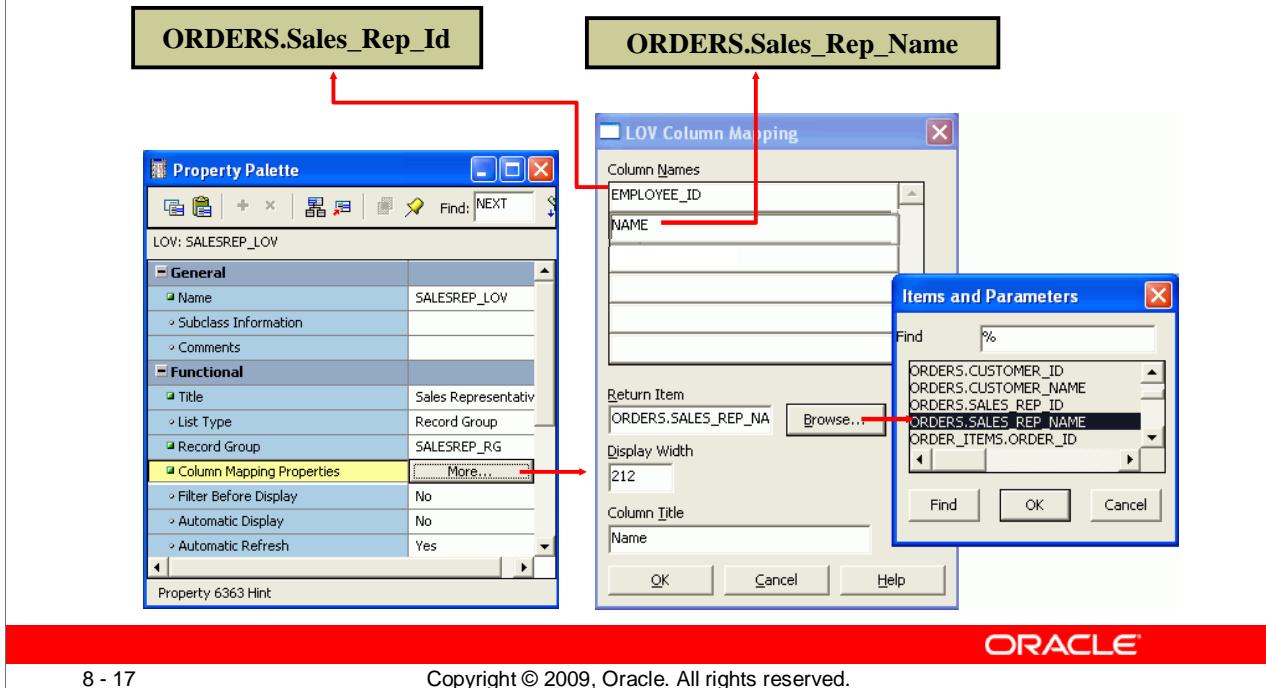
Setting LOV Properties



Setting LOV Properties (continued)

- **Automatic Refresh:** If Yes, the record group reexecutes the query every time the LOV is invoked; if No, the query fires only the first time when the LOV is invoked in the session.
Note: You can base more than one LOV on the same record group. When this is the case and you set Automatic Refresh to No, Forms Builder does not reexecute the LOV query after any of the associated LOVs are invoked.
- **Automatic Position:** Determines whether Forms automatically positions the LOV near the item from which it was invoked; the slide graphic illustrates this setting by showing the LOV moved close to the text item that invokes it.
- **Automatic Column Width:** Determines if Forms automatically sets the width of each column so that the entire title is displayed if the title is longer than the column display width, as illustrated in the graphic

Setting Column Mapping Properties



8 - 17

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Setting Column Mapping Properties

When you click More for Column Mapping Properties, the LOV Column Mapping dialog box opens, with the following elements:

- **Column Names:** Lets you select an LOV column for mapping or defining a column
- **Return Item:** Specifies the name of the form item or variable to which Forms should assign the column value. If null, the column value is not returned from the LOV. If you want to return a value, specify one of the following:
 - <block_name>.<item_name>
 - GLOBAL.<variable_name>
 - PARAMETER.<parameter_name>
- **Display Width:** Width of column display in LOV; value of zero causes the column to be hidden, but value is available for return
- **Column Title:** Heading for column in the LOV window

To set a column mapping, first select the column from the Column Names list, and then set the other mapping values, as required. The slide illustrates mapping the NAME column from the LOV to the ORDERS.Sales_Rep_Name item on the form.

Note: You can modify the record group query from its own properties list. The record group columns and LOV columns must remain compatible.

Associating an LOV with a Text Item

Set List of Values (LOV) properties of the text item.



Property Palette	
List of Values (LOV)	
List of Values	SALESREP_LOV
List X Position	0
List Y Position	0
Validate from List	No

Associating an LOV with a Text Item

To enable the user to invoke an LOV from a text item, you must specify the LOV name in the Property Palette of the text item.

1. Select the text item in the Object Navigator from which the LOV is to be accessible.
2. In the item Property Palette, set the List of Values (LOV) property to the required LOV.

The slide shows setting the List of Values (LOV) property for the ORDERS.Sales_Rep_Id text item to SALESREP_LOV. This means that the List of Values lamp is displayed when the user navigates to this text item, indicating that the LOV is available through the List of Values key or menu command.

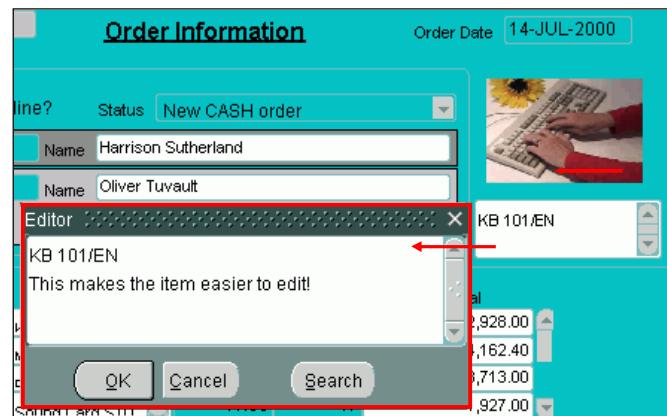
Other List of Values properties for a text item include:

- List X/Y Position: The X and Y coordinates where you want the LOV to display for this particular text item; if set to 0, use the properties that were set for the LOV
- Validate from List: Restrict the valid values for the text item to those that are contained in the LOV

What Are Editors?

Editors:

- Override default editor
- Used for special requirements such as larger editing window, position, color, and title



ORACLE®

8 - 19

Copyright © 2009, Oracle. All rights reserved.

What Are Editors?

With a text editor the user can view multiple lines of a text item simultaneously, search and replace text in it, and generally modify the value of an item from this separate window.

Every text item has the default editor available, but you can design your own replacement editor for those items that have special requirements such as larger editing window, position, color, and title.

By overriding the default editor for a text item, you can provide a larger editing window for items with potentially large textual values.

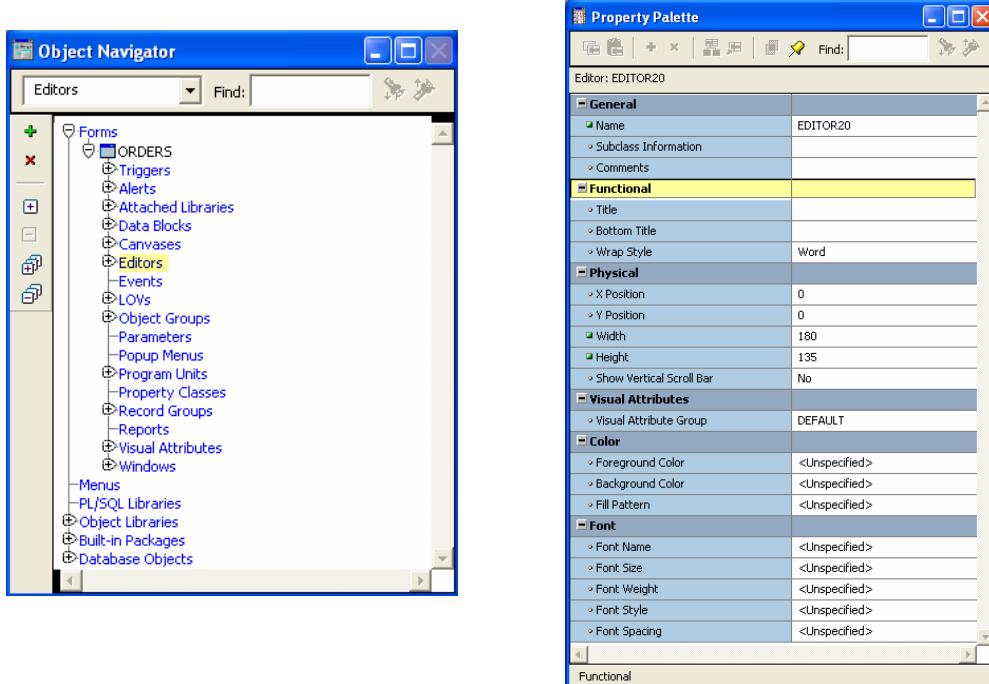
How to Use an Editor at Run Time

With the cursor on the text item to be edited, follow these steps:

1. Click Edit, or select Edit > Edit to invoke the attached editor.
2. Edit the text in the Editor window. Forms Builder editors provide a Search button that invokes an additional search-and-replace dialog box for manipulating text.
3. Click OK to write your changes back to the text item.

The screenshot shows the default editor that is invoked from a multi-line text item. The editor gives the user more space for editing the text item, and also offers the search-and-replace functionality.

Defining an Editor



Defining an Editor

If the user needs to use an editor on text values, the default Forms Builder editor is usually sufficient for most items. However, you can design your own customized editor as an object in a form module, and then attach it to the text items that need it.

How to Create a Customized Editor

1. Select the Editors node in the Object Navigator, and then click Create, as illustrated in the screenshot at the left of the slide. A new editor object is displayed in the list.
2. Select the new editor in the Object Navigator, and then access its Property Palette, where you can set its name and other properties. The Property Palette for an editor is shown at the right of the slide, with the property categories of General, Functional, Physical, Visual Attributes, Color, and Font.

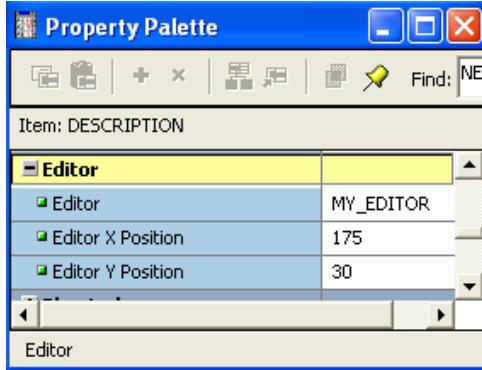
Setting Editor Properties

The following properties show the individual customization that is possible by creating your own editor:

- **Title/Bottom Title:** Displays at the top or bottom of the editor window
- **Width/Height:** Control size of editor window and, therefore, its editing area
- **X/Y Position:** Screen position for editor; can be overridden by a text item property
- **Wrap Style:** How text wraps in the window: None, Character, or Word
- **Show Vertical Scrollbar:** Specify Yes to add vertical scroll bar to the editor window

Associating an Editor with a Text Item

- Associate one of two types of editors with a text item.
- Set text item's Editor property to one of the following:
 - Null (default Forms Builder editor)
 - Editor name (customized editor)



ORACLE®

8 - 21

Copyright © 2009, Oracle. All rights reserved.

Associating an Editor with a Text Item

To associate an editor with a text item, you must specify the editor in the Property Palette of the text item.

Select the text item in the Object Navigator from which the editor is to be accessible.

In the item Property Palette, set the Editor property to one of the following settings:

- **Null:** The text item uses the default Forms Builder editor.
- **Editor Name:** The text item uses the named editor that you have created and customized in this module.

The screenshot shows the Property Palette for the Description text item. It shows that the Editor property is set to MY_EDITOR. This specifies that when the user invokes an editor from the Description text item at run time, the MY_EDITOR custom editor is invoked.

Summary

In this lesson, you should have learned that:

- An LOV is a scrollable pop-up window that enables a user to pick the value of an item from a multicolumn dynamic list
- The easiest way to design, create, and associate LOVs with text items is to use the LOV Wizard
- An editor is a separate window that enables the user to view multiple lines of a text item simultaneously, search and replace text in it, and modify the text
- You create editors in the Object Navigator and associate them with text items in the item's Property Palette

ORACLE®

8 - 22

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you learned that lists of values (LOVs) and text editors can be used to support text items. Both LOVs and editors are objects in a form module that open their own window when activated at run time and are used to support text items in any block of the form module.

- LOVs and editors can be shared across text items.
- The steps to implement an LOV are the following:
 1. Create a new LOV (and record group).
 2. Define column mapping for return items.
 3. Attach the LOV to text items, as required.
- The LOV Wizard performs these steps automatically.
- Text items can use the default editor or a user-named editor.

Practice 8: Overview

This practice covers the following topics:

- Creating an LOV and attaching the LOV to a text item
- Creating an editor and attaching it to a text item



Practice 8: Overview

In this practice session, you will create three LOVs and an editor.

- Using the LOV Wizard, create an LOV in the Orders form to display product numbers and their descriptions. Attach the LOV to the Product_Id item in the ORDER_ITEMS data block.
- Using the LOV Wizard, create an LOV in the Orders form to display sales representatives' IDs and names. Attach the LOV to the Sales_Rep_Id item in the ORDERS data block. Save and run the form.
- Using the LOV Wizard, create an LOV in the Customers form to display sales representatives' numbers and their names. Attach the LOV to the Acct_Mgr_Id item in the CUSTOMERS data block. Save and run the form.
- In the Customers form, create an editor for the Phone_Numbers item.

Creating Additional Input Items

9

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Identify the item types that allow input
- Create a check box
- Create a list item
- Create a radio group



Lesson Aim

In addition to text items, Oracle Forms Builder provides a variety of other item types. These can be divided into two groups: those that accept input and those that do not. This lesson covers input items and how they are used.

What Are Input Items?

- Item types that accept user input include:
 - Check boxes
 - List items
 - Radio groups
- Input items enable insert, update, delete, and query.

ORACLE®

9 - 3

Copyright © 2009, Oracle. All rights reserved.

What Are Input Items?

Input item is a generic term for Forms Builder item types that accept user input.

These item types include the following:

- Check box
- List item
- Radio group

What Can You Do with Input Items?

When you create input items, they already have some initial functionality. Through input items, you can interact with the database in the following ways:

- Insert values.
- Update existing values.
- Delete existing values.
- Query existing values.

Note: You can add functionality to input items with triggers and PL/SQL program units.

What Are Check Boxes?

- Two-state interface object:
 - Checked
 - Unchecked
- Not limited to two values



What Are Check Boxes?

A *check box* is a two-state interface object that indicates whether a certain value is ON or OFF. The display state of a check box is always either checked (selected) or unchecked (deselected). The screenshots show a check box in both its selected and deselected states.

You can use check boxes to enhance the user interface by converting existing items that have two possible states. Although a check box is limited to two states, it is not limited to just two values. You specify the value to represent checked, the value to represent unchecked, and how other values are processed.

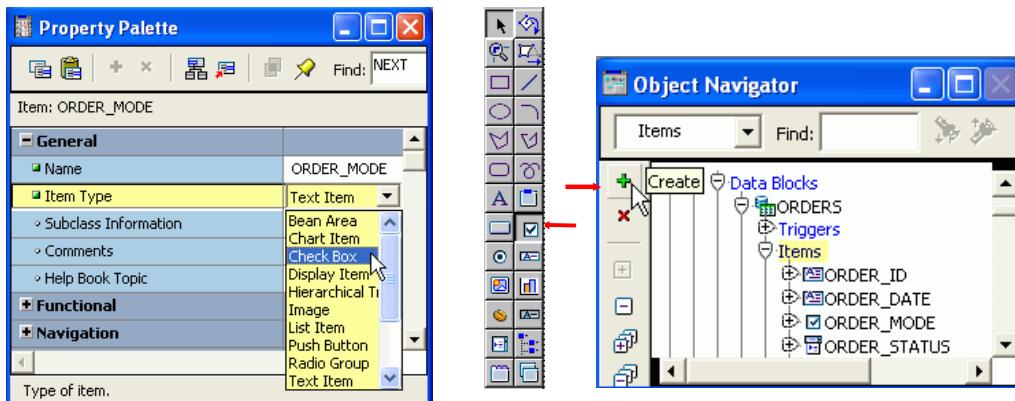
Using a Check Box at Run Time

You can do the following at run time:

- Set check box values either by user input, by means of the Initial Value property, or programmatically.
- In Enter Query mode:
 - Query checked values by clicking one or more times until item is selected
 - Query unchecked values by clicking one or more times until item is deselected
 - Ignore check box values by not selecting or deselecting the initial displayed value

Creating a Check Box

- Convert an existing item.
- Use the Check Box tool in the Layout Editor.
- Use the Create icon in the Object Navigator, then convert the text item to a check box.



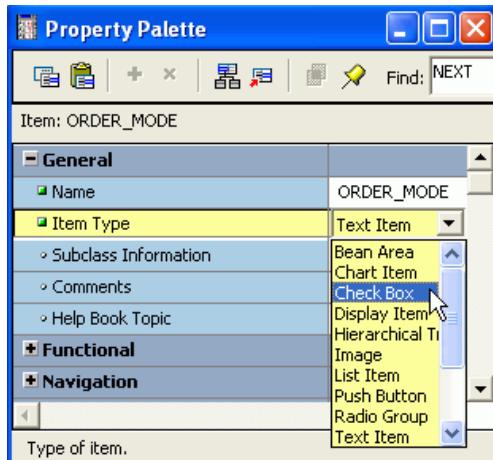
Creating a Check Box

A check box can be created by one of the following methods:

- Converting an existing item—the screenshot at the left shows that the Item Type drop-down list for a text item includes the Check Box item type that you can select.
- Using the Check Box tool in the Layout Editor—the screenshot in the middle of the slide shows the Check Box tool.
- Using the Create icon in the Object Navigator (creates a text item, which you can convert to a check box)—the screenshot at the right of the slide shows creating a text item in the Object Navigator.

Converting an Existing Item to a Check Box

Convert text item
to check box.



ORACLE®

9 - 6

Copyright © 2009, Oracle. All rights reserved.

Converting an Existing Item to a Check Box

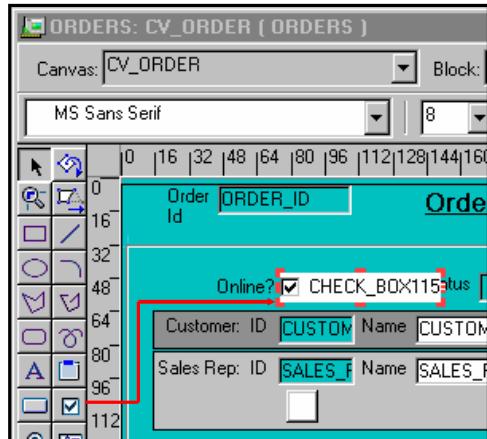
You can convert an existing item to a check box by changing the Item Type property to Check Box in the Property Palette and setting other relevant properties. Perform the following steps:

1. Invoke the Property Palette for the item that you want to convert.
2. Set the Item Type property to Check Box.
3. Specify a check box label.
4. Specify values for the checked and the unchecked states.
5. Set the “Check Box Mapping of Other Values” property.
6. Specify an initial value for the check box item.

Note: The check box label that you specify is displayed to the right of the check box element at run time. If the complete label name is not displayed, adjust it in the Layout Editor. If the item already has a prompt, delete it in the item Property Palette.

Creating a Check Box in the Layout Editor

**Use the Check Box tool
in the Layout Editor.**



ORACLE®

9 - 7

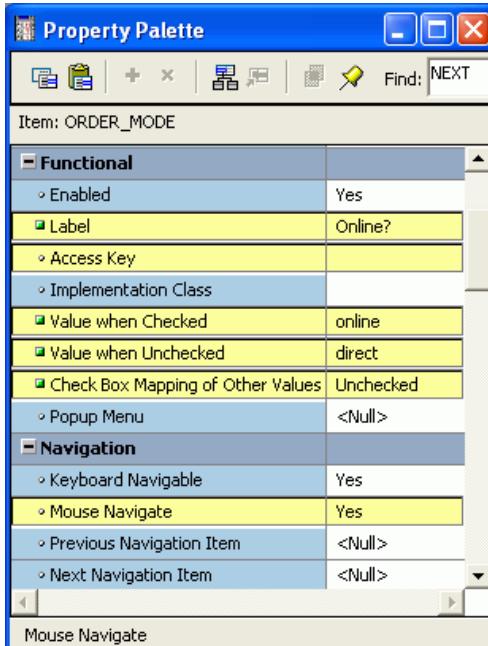
Copyright © 2009, Oracle. All rights reserved.

Creating a Check Box in the Layout Editor

You can also create a check box by using the Check Box tool in the Layout Editor.

1. Invoke the Layout Editor.
2. Set the canvas and block to those on which you want the check box item to be displayed.
3. Click the Check Box tool.
4. Click the canvas in the position where you want the check box to be displayed. The screenshot shows the Layout Editor right after this step has been performed.
5. Double-click the check box to invoke its Property Palette.
6. Set the properties as required.

Setting Check Box Properties



- Data Type
- Label
- Access Key
- “Value when Checked”
- “Value when Unchecked”
- “Check Box Mapping of Other Values”
- Mouse Navigate

ORACLE®

9 - 8

Copyright © 2009, Oracle. All rights reserved.

Setting Check Box Properties

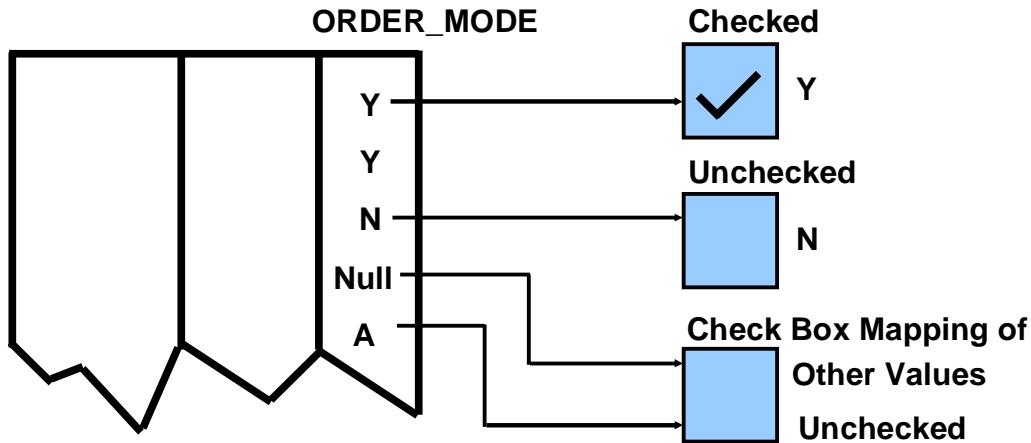
The following properties may be set to affect the appearance and behavior of check boxes:

- **Data Type:** Must be compatible with values specified in the Value properties
- **Label:** Text label displayed next to check box item (independent of the check box value)
- **Access Key:** Which combination of keys may be used to navigate to this item and select or deselect it
- **Initial Value:** Initial value of the item for new record, determining whether check box is initially selected or deselected
- **“Value when Checked”:** Value to represent selected state of the check box
- **“Value when Unchecked”:** Value to represent deselected state of the check box
- **“Check Box Mapping of Other Values”:** How other values are to be processed (Not Allowed, Checked, or Unchecked)
- **Mouse Navigate:** Whether Forms navigates to and moves input focus to the item when the user activates the item with the mouse (default is Yes)

The screenshot shows the Property Palette for a check box for an input item that can have values of either online or direct. The following property values are set:

- **Label:** Online?
- **“Value when Checked”:** online
- **“Value when Unchecked”:** direct
- **“Check Box Mapping of Other Values”:** Unchecked
- **Mouse Navigate:** Yes

Check Box Mapping of Other Values



Check Box Mapping of Other Values

Dealing with Other Values

If your base-table column accepts other values, your check box should account for them. You can assign other values to either the checked or unchecked states by using the Check Box Mapping of Other Values property. Alternatively, you can choose not to accept other values with the Not Allowed setting.

Note: If you choose not to accept other values and they exist in the base-table column, Forms ignores the entire record during query processing.

Dealing with Null Values

If your base-table column accepts null values, you can account for them by one of the following methods:

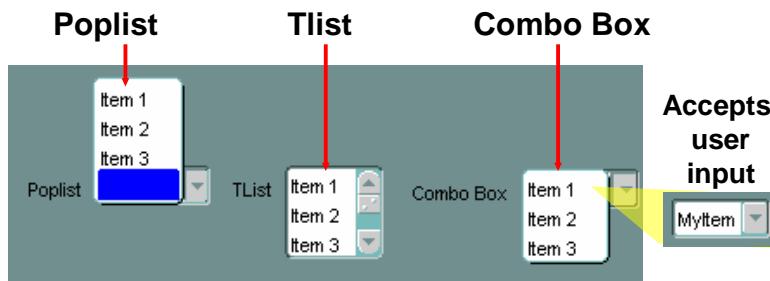
- Set the “Check Box Mapping of Other Values” property.
- Set the checked or unchecked state to represent null (leave the value blank).

The example in the slide shows a database table with a column named ORDER_MODE that has values of Y, N, and A, and also some null values. When Checked is set to Y for the check box, Unchecked is set to N, Checkbox Mapping of Other Values is set to Unchecked, all records can be displayed in the form block, but only the Y records show the check box to be selected.

What Are List Items?

List Items:

- Set of mutually exclusive choices, each representing a different value
- Three list styles available:



- Space-saving alternative to a radio group
- Smaller-scale alternative to an LOV

ORACLE®

9 - 10

Copyright © 2009, Oracle. All rights reserved.

What Are List Items?

A *list item* is an interface object that displays a predefined set of choices, each corresponding to a specific data value. You use the list item at run time to select a single value. List choices or elements are mutually exclusive; only one can be selected at a time.

List Item Styles

- **Poplist:** It appears as a field with a button attached to the right; click the poplist or its button to display all its list elements.
- **Tlist:** It appears as a rectangular box that displays the list elements; when the display area is not big enough to display all the list elements, a scroll bar is automatically attached to the right to view the remaining list elements, or you can use the up and down arrow keys to navigate the list.
- **Combo Box:** It appears as a field with a drop-down list; click the button to display all the combo box list elements. As illustrated on the slide, the combo box is the only list item type that accepts user input of values not on the list.

What Are List Items? (continued)

Note: The poplist and combo box take up less space, but end users must open them to see the list elements. A tlist remains “open,” and end users can see multiple values at a time. Use the attached scroll bar to see more values if the tlist is not big enough to display all the list elements.

Uses and Benefits of List Items

- Enable display of a defined set of choices.
- Display a set of choices without using a vast area of canvas.
- Provide an alternative to radio groups.
- Provide a Windows-style list of values.

Setting the Value for a List Item

The value for a list item can be set in any of the following ways:

- User selection
- User input (combo box style only)
- A default value
- Programmatic control

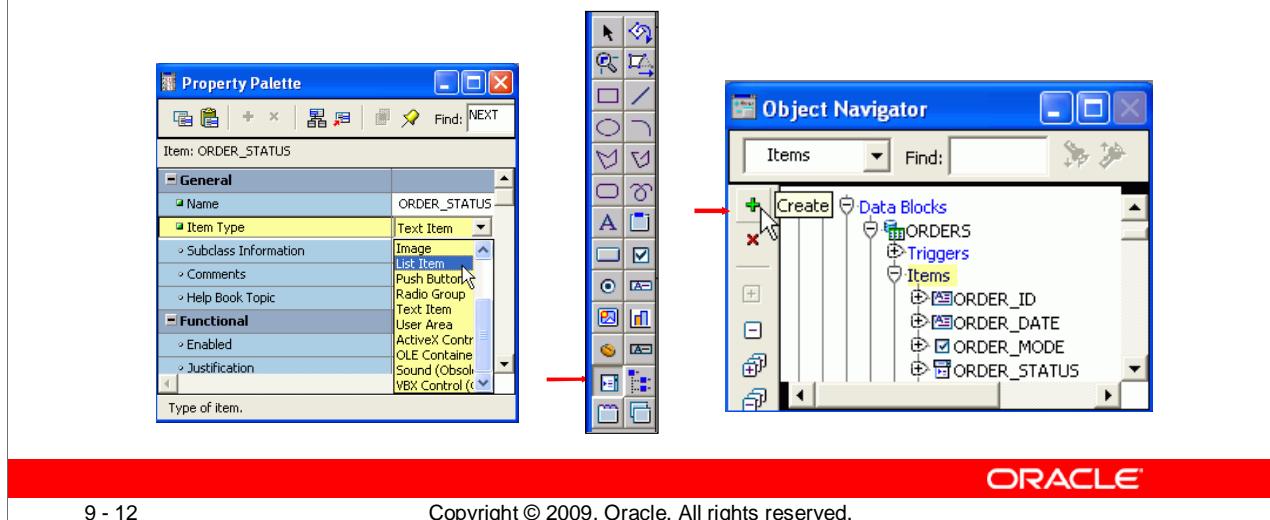
What are some of the differences between a list item and an LOV?

List items:

- Are generally used for a small number of elements
- Do not have a Find button
- Cannot be attached to other items
- Usually have choices that are not based on a SELECT statement (although they can be if the list elements are created programmatically)

Creating a List Item

- Convert an existing item.
- Use the List Item tool in the Layout Editor.
- Use the Create icon in the Object Navigator, then convert the text item to a list item.

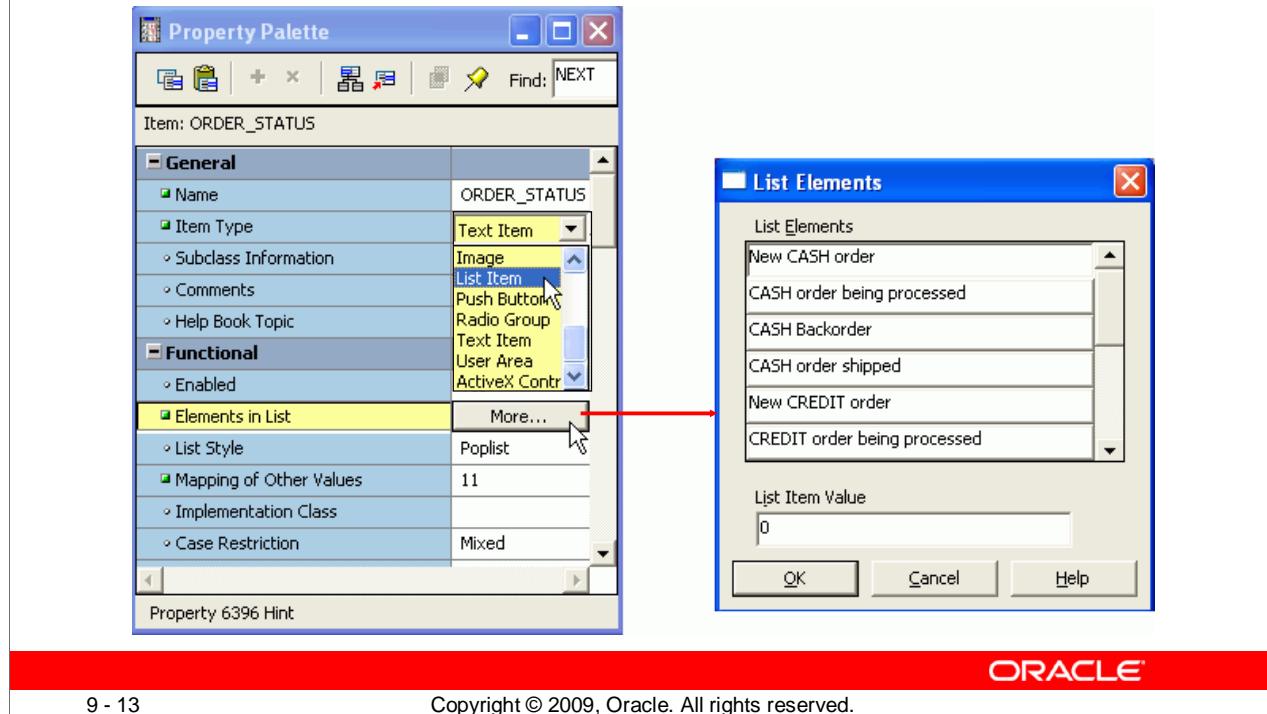


Creating a List Item

A list item can be created by:

- Converting an existing item; the screenshot at the left of the slide shows that the Item Type drop-down list for a text item includes the List Item item type that you can select.
- Using the List Item tool in the Layout Editor; the screenshot at the middle of the slide shows the List Item tool.
- Using the Create icon in the Object Navigator (creates a text item that you can convert to a list item.); the screenshot at the right of the slide shows creating a text item in the Object Navigator.

Converting an Existing Item to a List Item



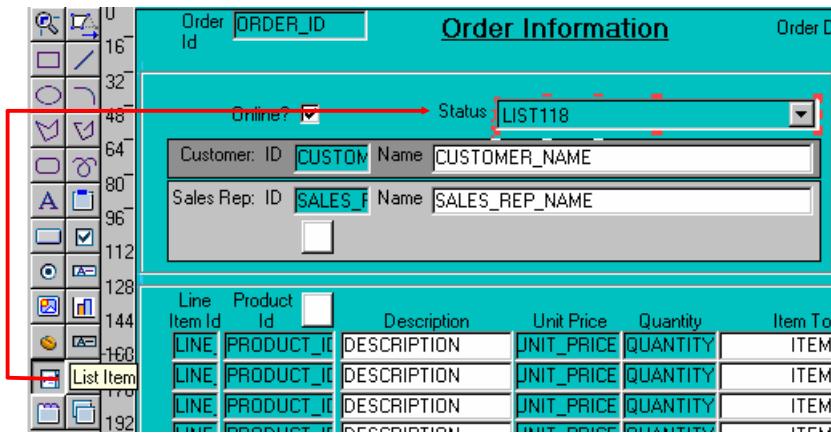
Converting an Existing Item to a List Item

You can convert an existing item to a list item by changing its Item Type property to List Item and setting the relevant properties.

1. Invoke the Property Palette for the item that you want to convert.
2. Set the Item Type property to List Item.
3. Select the “Elements in List” property and click More. The List Elements dialog box appears.
4. Enter the element that you want to display in your list item in the List Elements column; the example shows possible display values for the Order_Status item, such as “New CASH order” and “CASH order being processed”.
5. Enter the value for the currently selected list element in the List Item Value field. The example in the slide shows that if the user selects the displayed value of “New CASH order”, the value 0 is entered in the List Item Value field.
6. Create additional list elements and values by repeating steps 5 and 6.
7. Click OK to accept and close the List Elements dialog box.
8. Set the Other Values property to do one of the following:
 - Reject values other than those predefined as list values.
 - Accept and default all other values to one of the predefined list element values.
9. Enter an initial value for the list item.

Creating a List Item in the Layout Editor

Use the List Item tool
in the Layout Editor.



ORACLE®

Creating a List Item in the Layout Editor

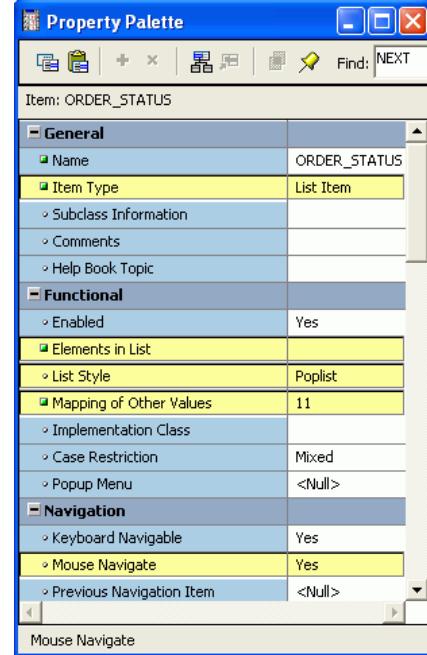
You can also create a list item by using the List Item tool in the Layout Editor.

1. Invoke the Layout Editor.
2. Set the canvas and block to those on which you want the list item to be displayed.
3. Select the List Item tool.
4. Click the canvas in the position where you want the list item to be displayed. The screenshot shows the Layout Editor just after this step has been performed.
5. Double-click the list item to invoke its Property Palette.
6. Set the properties as required.

Note: To obtain a list of available functions when defining list elements, press Ctrl + K while the input focus is in the List Elements window. The Keys window may appear behind the List Elements window; if so, just move the List Elements window so that you can see the Keys window.

Setting List Item Properties

- “Elements in List”:
 - List elements
 - List item value
- List Style
- “Mapping of Other Values”
- Mouse Navigate



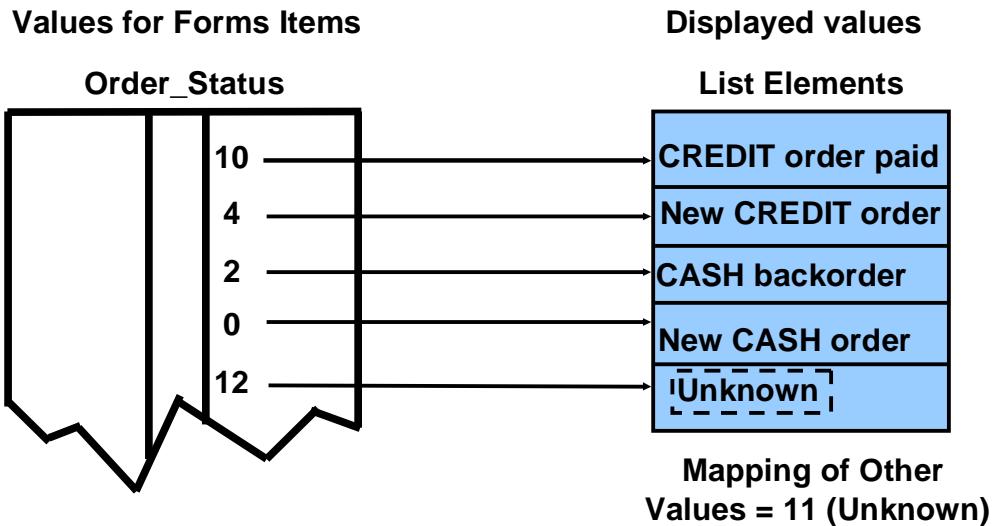
ORACLE®

Setting List Item Properties

- “Elements in List”: Clicking More opens the List Elements dialog box, where you specify:
 - List Elements: List elements that display at run time
 - List Item Value: Actual value that corresponds to the list element
- List Style: Display style of list item (Poplist, Tlist, or Combo Box)
- “Mapping of Other Values”: How other values are processed
- Mouse Navigate: Whether Forms navigates to the item and moves input focus to it when the user clicks the item

The screenshot shows the Property Palette for the Order_Status list item with List Style set to Poplist, “Mapping of Other Values” set to 11 (which in this application corresponds to a status of “Unknown”), and Mouse Navigate set to Yes.

List Item Mapping of Other Values



List Item Mapping of Other Values

NULL Values in a List Item

If the base-table column for a list item accepts NULL values, Forms Builder creates a pseudochoice in the list to represent the null.

All three list styles display a blank field if a query returns a NULL value. If the Data Required property is set to No:

- Poplist displays a blank element for a NULL value
- The user can omit a selection for TList or can click Clear Field to deselect all list elements. This sets the list item to NULL.
- Combo Box does not display a blank element. The end user must delete the default value if the default value is not NULL.

Handling Other Values in a List Item

If the base-table column for a list item accepts values other than those associated with your list elements, you must specify how you want to handle the values. Do this in one of the following ways:

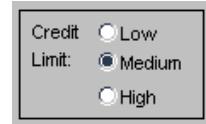
- Ignore other values by leaving the “Mapping of Other Values” property blank.
- Associate the other values with one of the existing list elements (by naming either the list element or its associated value) in the “Mapping of Other Values” property.

List Item Mapping of Other Values (continued)

In the example in the slide, order status codes were defined to include 0 through 11. The table, however, contains a record with the status code of 12. For the list item in the form, the “Mapping of Other Values” property is set to 11 (meaning status is unknown.) The form then displays the record with the status code of 12, but displays its status as unknown.

What Are Radio Groups?

- Set of mutually exclusive radio buttons, each representing a value
- Use:
 - To display two or more static choices
 - As an alternative to a list item
 - As an alternative to a check box



ORACLE®

9 - 18

Copyright © 2009, Oracle. All rights reserved.

What Are Radio Groups?

A *radio group* is an item where a set of radio buttons represents the possible values for the item. These values and hence their corresponding radio buttons are mutually exclusive.

Uses and Benefits of Radio Groups

Radio groups provide:

- A choice between two or more static values; the screenshot shows a radio group for Credit Limit with choices of Low, Medium, or High.
- An alternative to list items that have only a few choices
- A choice between two alternatives, where choice is not On/Off or Yes/No (for example, Landscape or Portrait print format)

Note: Consider list items instead of radio groups if there are more than four or five choices.

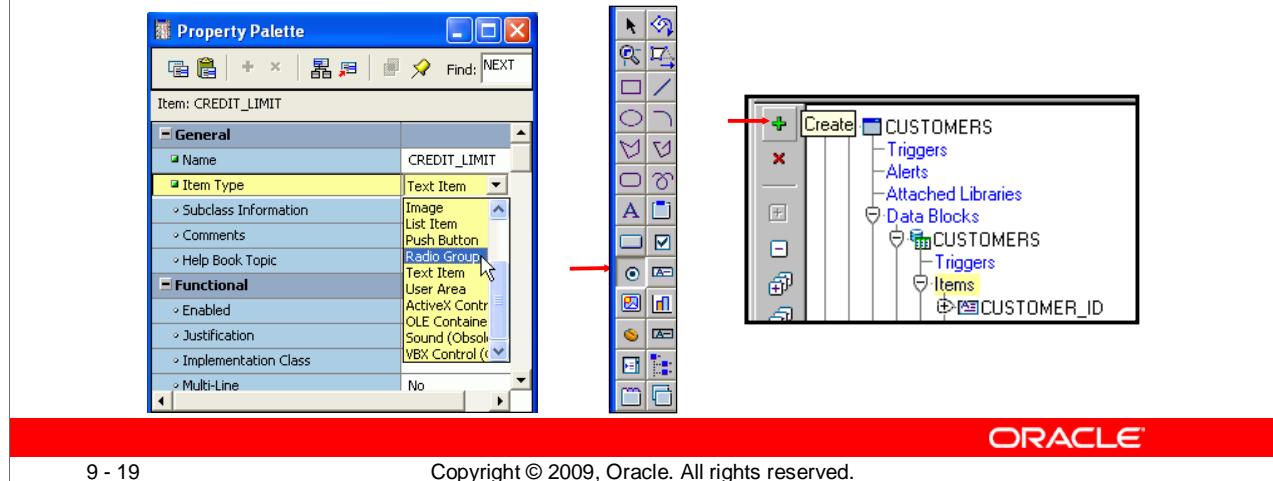
Using a Radio Group at Run Time

You can do the following at run time:

- Set radio group values:
 - By user input
 - By means of the Initial Value property
 - Programmatically
- Query individual radio button values.

Creating a Radio Group

- Convert an existing item.
- Create a new radio button in the Layout Editor, which implicitly creates a radio group also.
- Use the Create icon in the Object Navigator, then convert the text item to a radio group.

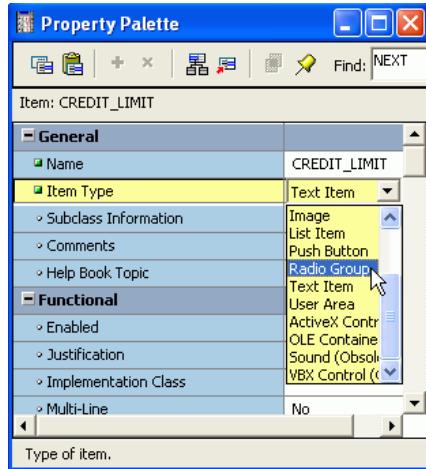


Creating a Radio Group

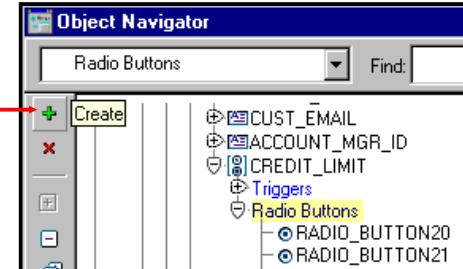
A radio group can be created by:

- Converting an existing item to a radio group; the screenshot at the left of the slide shows that the Item Type pop-up list for a text item includes the Radio Group item type that you can select.
- Creating a new radio button in the Layout Editor (automatically creates a radio group if none exists); the screenshot in the middle of the slide shows the Radio Button tool.
- Using the Create icon in the Object Navigator (this creates a text item that you can convert to a radio group); the screenshot at the right shows creating a text item in the Object Navigator.

Converting an Existing Item to a Radio Group



Change Item Type and
set other properties.



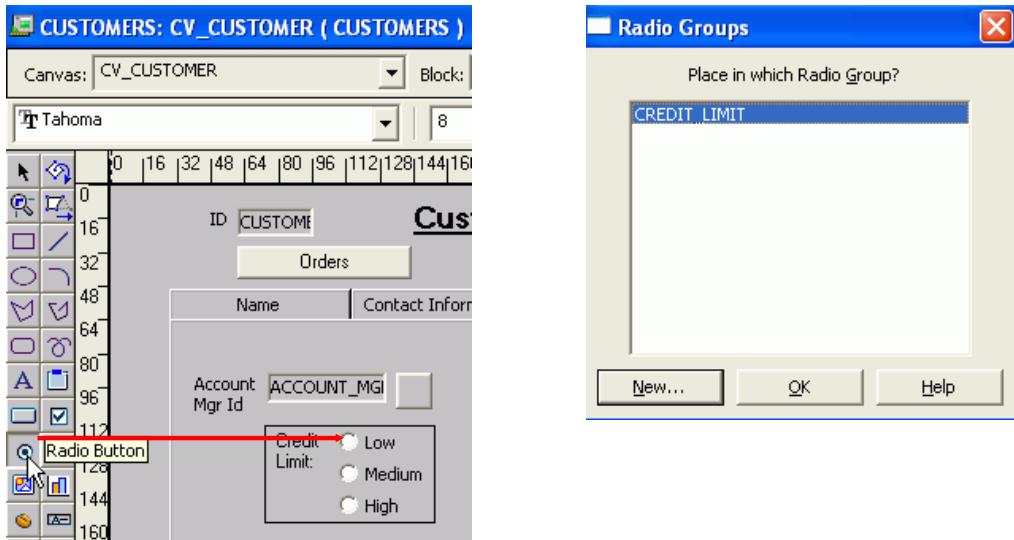
Create radio buttons for
the radio group.

Converting an Existing Item to a Radio Group

You can convert an existing item to a radio group by changing the item type and setting the properties for a radio group.

1. Invoke the Property Palette for the item that you want to convert.
2. Set the Item Type property to Radio Group.
3. Set the Canvas property to the Canvas on which you want the radio buttons to appear.
4. Set the “Mapping of Other Values” property to specify how the Radio Group should handle any other values.
5. Set the Initial Value property, as required. This should be the name of a radio button.
6. Expand the item node in the Object Navigator.
The Radio Buttons node appears.
7. Select the Radio Buttons node and click the Create icon. A radio button displays in the Object Navigator and the Property Palette takes on its context.
8. Enter a name, value, and a label for the radio button.
9. Specify the display properties of the radio button.
10. Create additional radio buttons by repeating steps 6 through 9.

Creating a Radio Group in the Layout Editor



ORACLE®

9 - 21

Copyright © 2009, Oracle. All rights reserved.

Creating a Radio Group in the Layout Editor

You can also create a radio group by using the Radio Button tool in the Layout Editor.

1. Invoke the Layout Editor.
 2. Set the canvas and block to those on which you want the radio group to be displayed.
 3. Click the Radio Button tool.
 4. Click the canvas at the desired location for the radio group, as shown in the screenshot at the left of the slide.
- If you already have a radio group in the current block, the Radio Groups dialog box appears and you must decide whether the new radio button should appear in the existing group or a new one. If you select New, the new radio group is created implicitly. The screenshot at the right of the slide shows the new radio button being created in the existing Credit_Limit radio group.

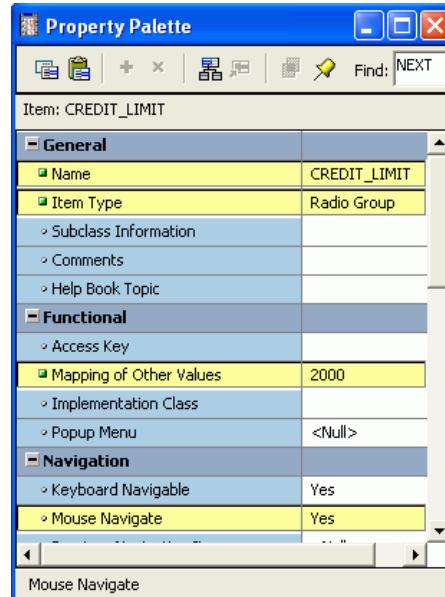
5. Double-click the radio button to invoke its Property Palette.
6. Set the radio button properties as required.

Note: The canvas property for a radio group is set in the Property Palette of the radio group. The individual radio buttons do not have a canvas property.

Setting Radio Group Properties

Radio group properties:

- “Mapping of Other Values”
- Data Type
- Initial Value
- Mouse Navigate



Setting Radio Group Properties

You should set the following properties for radio groups:

- **Data Type:** Must be compatible with “Mapping of Other Values” for the Radio Group, and Radio Button Value for each Radio Button in the group
- **“Mapping of Other Values”:** How values other than those specified are processed
- **Mouse Navigate:** Whether Forms navigates to the item when the operator clicks the item

You also must specify a valid Initial Value, except under either of the following conditions:

- The radio group accepts other values.
- The value associated with one of the radio buttons is NULL.

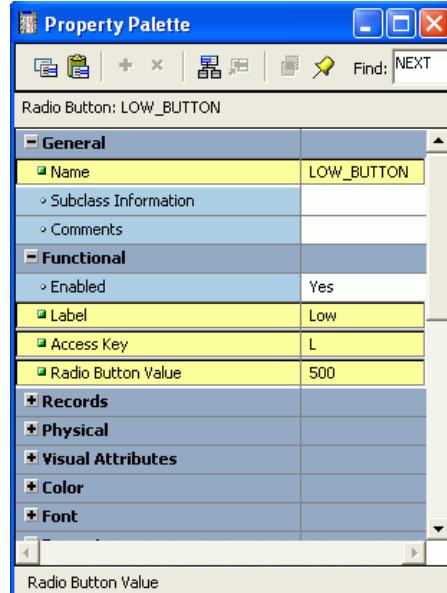
The screenshot shows setting the following property values for the Credit_Limit radio group:

- **“Mapping of Other Values”:** 2000
- **Mouse Navigate:** Yes

Setting Radio Button Properties

Radio button properties:

- Label
- Access Key
- Radio Button Value



ORACLE®

Setting Radio Button Properties

You should set the following properties for radio buttons:

- **Label:** Text that appears adjacent to the radio button (independent of the button value)
- **Access Key:** Which combination of keys can be used to navigate to and manipulate this radio button
- **Radio Button Value:** The value for this item if this radio button is selected

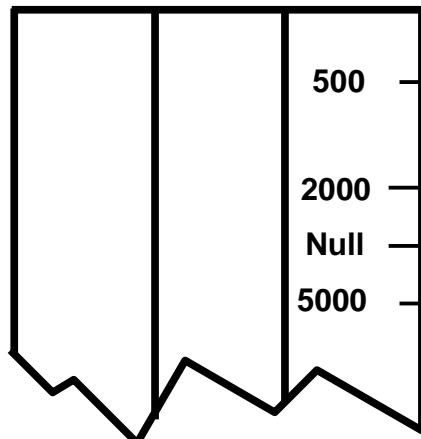
The screenshot shows setting the following property values for the Low_Button radio button:

- **Label:** Low
- **Access Key:** L
- **Radio Button Value:** 500

Radio Group Mapping of Other Values

Values for Forms Items

CREDIT_LIMIT



Displayed Values

List Elements

- LOW_BUTTON Low
 - MEDIUM_BUTTON Medium
 - HIGH_BUTTON High
- Mapping of Other Values
2000

ORACLE®

Radio Group Mapping of Other Values

Handling Other Values in a Radio Group

If the base-table column for a radio group accepts values other than those associated with your radio buttons, you must use one of the following methods to specify how you want to handle the values:

- Ignore other values by leaving the radio group's "Mapping of Other Values" property blank.
- Associate the other values with one of the existing radio buttons by naming the associated value of the button in the "Mapping of Other Values" property.

Note: Ignoring other values results in the entire row being ignored during query processing.

NULL Values in a Radio Group

A radio group can treat NULL as a valid value. You should account for the NULL case, if your base-table column allows null values. You can do this in one of the following ways:

- Use the "Mapping of Other Values" property to implicitly force NULL to a radio button.
- Assign the NULL to its own radio button by leaving the Radio Button Value property blank.

The example in the slide shows that when "Mapping of Other Values" is set to 2000, null values appear with the radio button labeled Medium selected.

Summary

In this lesson, you should have learned that:

- Check boxes, list items, and radio groups are the item types that allow input
- You create these items by:
 - Changing the item type of an existing item
 - Using the appropriate tool in the Layout Editor
- You can use a check box for items that have only two possible states
- You can use a list item to enable users to pick from a list of mutually exclusive choices
- You can use a radio group for two or three mutually exclusive alternatives

ORACLE®

9 - 25

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you learned how to create items that accept direct user input. Use these items to enhance the user interface:

- **Check boxes:** To convert items that have two possible states
- **List items (Poplists, Tlists, and Combo Boxes):** To convert items that have mutually exclusive choices
- **Radio groups:** To convert items with two or three alternatives that are mutually exclusive

Practice 9: Overview

This practice covers the following topics:

- Converting a text item into a list item
- Converting a text item into a check box item
- Converting a text item into a radio group
- Adding radio buttons to the radio group



Practice 9: Overview

In this practice, you convert existing text items into other input item types. You create a list item, a check box, and a radio group.

- In the Orders form, convert the Order_Status item into a list item. Save and run the form.
- In the Orders form, convert the Order_Mode item into a check box item.
- In the Customers form, convert the Credit_Limit item into a radio group. Add three radio buttons in the radio group. Save and run the form.

10

Creating Noninput Items

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Identify item types that do not allow input
- Create a display item
- Create an image item
- Create a button
- Create a calculated item
- Create a hierarchical tree item
- Create a bean area item



Lesson Aim

Some Oracle Forms item types do not accept user input (noninput items), but they do provide an effective means of accessing data and initiating actions. This lesson describes how to create and use noninput items.

Noninput Items: Overview

Item types that do not accept direct user input include:

- Display items
- Image items
- Push buttons
- Calculated items
- Hierarchical tree items
- Bean area items



Noninput Items: Overview

“Noninput items” is a generic term for item types that do not accept direct user input. However, you can set the value of some noninput items by programmatic control. Noninput items can be divided into two groups—those that can display data and those that cannot.

- Noninput items that can display data:
 - Display items
 - Image items
 - Calculated items
 - Hierarchical tree items
- Noninput items that cannot display data:
 - Push buttons

Bean area items can fall into either of these categories, depending on the functionality of the implemented JavaBean.

Use noninput items to enhance your application by displaying additional data, often from a nonbase table.

What Are Display Items?

Display items:

- Are similar to text items
- Cannot:
 - Be edited
 - Be queried
 - Be navigated to
 - Accept user input
- Can display:
 - Non-base-table information
 - Derived values



What Are Display Items?

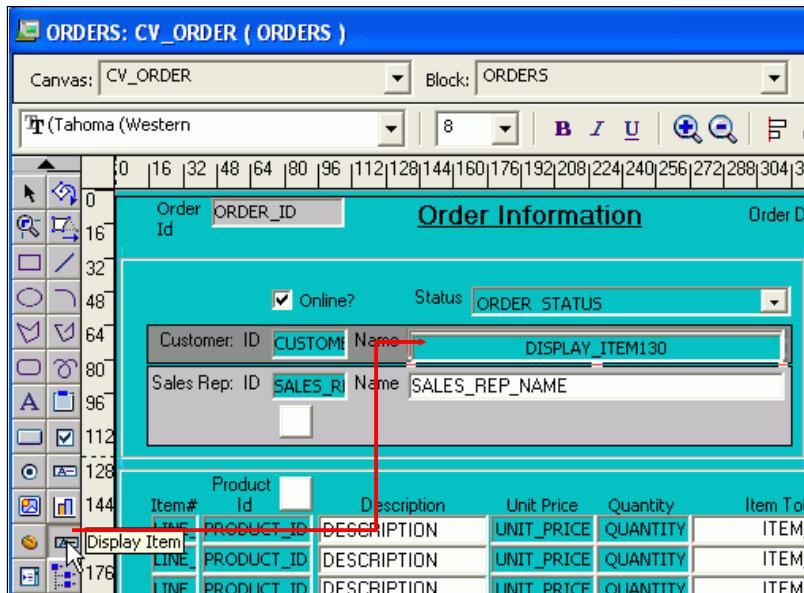
A display item is similar to a text item, except that it cannot be edited or navigated to at run time. A display item is a read-only text box whose value must be fetched or assigned programmatically.

You can use display items to display:

- Additional, non-base-table information
- Derived data values

The screenshot shows two display items, the customer name and the sales representative name, that were created by using the Display Item tool in the Layout Editor.

Creating a Display Item



10 - 5

Copyright © 2009, Oracle. All rights reserved.

Creating a Display Item

A display item can be created by using:

- The Layout Editor
- The Create icon in the Object Navigator (creates a text item that you can convert to a display item)
- The Item Type property to convert an existing item into a display item

To create a display item in the Layout Editor, perform the following steps:

1. Invoke the Layout Editor.
2. Display the desired canvas and select the correct data block.
3. Click the Display Item tool.
4. Click the canvas at the position where you want to place the display item. The screenshot shows the Layout Editor after this step is performed.
5. Double-click the new display item to display its Property Palette.
6. Change the Name from DISPLAY_ITEMXX to the desired name.
7. Specify the other properties as needed. You can set the required item properties in the Property Palette:
 - Set Database Item to No for a nonbase table display item.
 - You can assign a format mask for a single-line display item by setting its Format Mask property.

Image Items

- Use image items to display images.
- Images imported from a file system are stored in any supported file type.
- Images imported from a database are stored in a LONG RAW column or a BLOB column.



The screenshot shows an Oracle Forms application window titled "Order Information". The window displays an order with Order Id 2355 and Order Date 26-JUN-1998. It includes fields for Online? (checked), Status (CREDIT order billed), Customer ID (104) and Name (Harrison Sutherland), and Sales Rep ID (KB 101/ES). A product line item table shows Item Id 2289, Description KB 101/ES, Unit Price 46, Quantity 200, and Item Total 9,200.00. An image of a person typing on a keyboard is displayed in a red-bordered image item. The Oracle logo is at the bottom right.

Image Items

Using Images

You can use images as graphic objects within a form module. A graphic image is displayed automatically and cannot be manipulated at run time. It can be imported from the database or the file system.

Alternatively, you can display images within image items.

What Is an Image Item?

An image item is a special interface control that can store and display vector or scanned images. Just as text items store and display VARCHAR2, number, or date values, image items store and display images.

Like other items that store values, image items can be either data block items or control items.

The screenshot shows an image item that displays a product image for a selected product in an order. The Layout Editor tool for creating an image item is also shown.

Image Items (continued)

Displaying Image Items

You can populate an image item by using one of the following methods:

- Fetching from a LONG RAW or BLOB database column
- Using a trigger and a built-in to populate the image item programmatically; you learn about triggers in later lessons.

Storing Images

You can store images in either the database or the file system.

When you insert images into the database by means of a save (commit), they are automatically compressed using Oracle image compression.

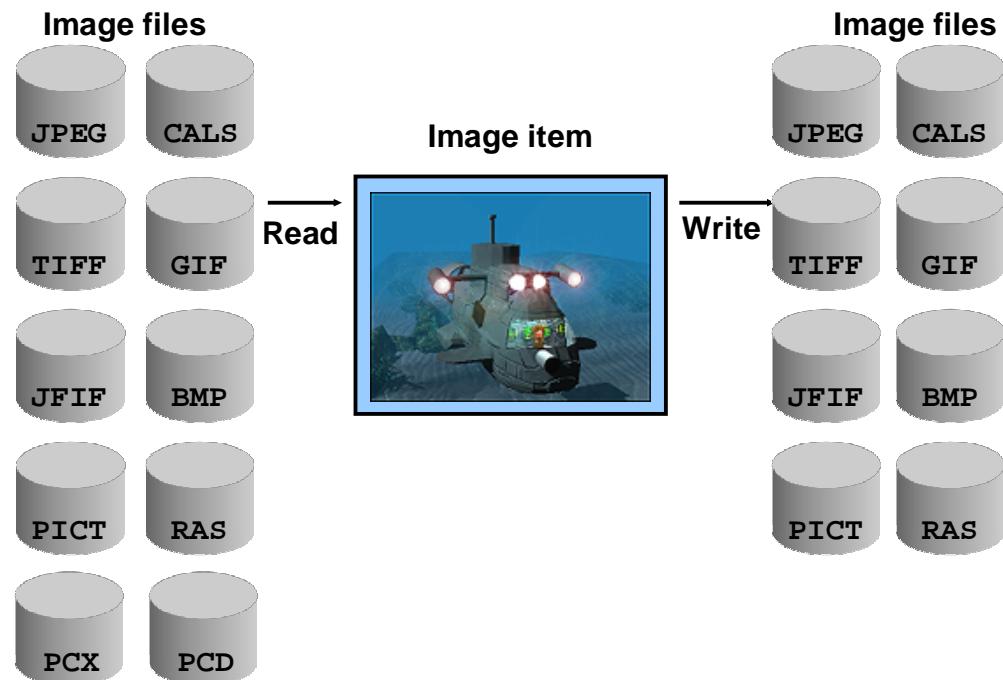
Where Image Is Stored	Description
Database	LONG RAW column compressed image that can be up to 2 gigabytes or BLOB column that can be up to four gigabytes
File	Any supported file format

Note: To conserve application server memory when displaying large image items, reduce the number of records that are buffered by manipulating the “Number of Records Buffered” data block property.

Technical Note

You can also populate an image item with a BFILE, but you need to use DBMS_LOB to do so.

Image File Formats



ORACLE®

10 - 8

Copyright © 2009, Oracle. All rights reserved.

Image File Formats

Forms Builder supports the following image formats:

File Suffix	Description	Image Items
BMP	MS Windows and OS/2 BitMap Picture	Read/Write
CALS	CALS type raster	Read/Write
GIF	Graphics interchange format	Read/Write
JFIF	JPEG file interchange format	Read/Write
TIFF (single page)	Tag image file format	Read/Write
JPEG	Joint Photographic Experts Group	Read/Write
PICT	Macintosh Quickdraw Picture	Read/Write
RAS	Sun Raster	Read/Write
PCX	PC Paintbrush Bitmap Graphic	Read only
PCD	Photo-CD Image	Read only

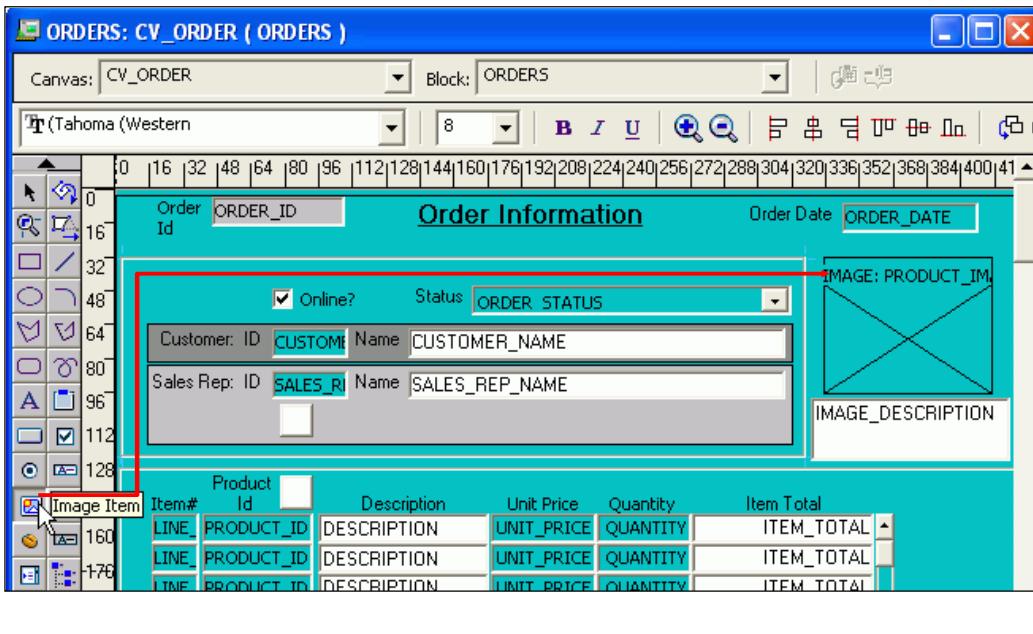
To reduce network traffic, limit the number of image items and background images that must be downloaded from the application server. You can deploy them as Java Archive (JAR) files to reduce network traffic, or you can download them in an alternative manner.

Image File Formats (continued)

For example, to display a company logo in your application, you can include the image in the HTML page that downloads at application startup rather than retrieving the image from the database or the file system. The HTML page can be cached.

The graphics in the slide convey the fact that most image types can be both read into an image item from the database and written from an image item to the database; only .pcd and .pcx image types are read-only.

Creating an Image Item



10 - 10

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Creating an Image Item

There are three ways to create an image item:

- By using the Image Item tool in the Layout Editor (as described in the next section)
- By using the Create icon in the Object Navigator (creates a text item that you can convert to an image item)
- By converting an existing item into an image item

Steps to Create an Image Item from the Layout Editor

1. Invoke the Layout Editor.
2. Set the canvas and block to those where you want the item to display.
3. Click the Image Item tool.
4. Click the canvas at the position where you want the image item to display. The screenshot in the slide shows the Layout Editor after this step has been performed.
5. Double-click the image item. The Property Palette appears.
6. Change the name from IMAGEXX to the desired name.
7. Specify the other properties as needed.

Note: Remember to set the Database Item property to No for an image item whose value is not stored in the base table.

Setting Image-Specific Item Properties

- Image Format
- Image Depth
- Display Quality
- Sizing Style
- Show Horizontal Scroll Bar
- Show Vertical Scroll Bar

Functional	
Enabled	Yes
Image Format	TIFF
Image Depth	Original
Display Quality	High
Sizing Style	Crop
Popup Menu	<Null>
Navigation	
Data	
Records	
Database	
Physical	
Visible	Yes
Canvas	CANVAS2
Tab Page	<Null>
X Position	68
Y Position	68
Width	72
Height	72
Bevel	Lowered
Show Horizontal Scroll Bar	No
Show Vertical Scroll Bar	No

ORACLE®

10 - 11

Copyright © 2009, Oracle. All rights reserved.

Setting Image-Specific Item Properties

The screenshot shows the Property Palette of an image item with the default property values.

Set the following properties to affect the appearance and behavior of the image item:

- **Image Format:** Format in which image is stored in the database (default is TIFF)
- **Image Depth:** Depth setting for image being read from or written to a file on the file system (Original, Monochrome, Gray, LUT , or RGB) (default is Original)
- **Display Quality:** Resolution used to display the image item; controls trade off between quality and performance (High, Medium, or Low) (default is High)
- **Sizing Style:** Determines how much of the image is displayed when the sized image does not match the size of the item (Crop cuts off edges of the image so that it fits in item display area; Adjust scales the image so that it fits within the item display area.) (default is Crop)
- **Show Horizontal/Vertical Scroll Bar:** Displays scroll bars to enable scrolling of the image that does not fit into the item display area (default is No)

Note: Image items do not have a Data Type property. If you set an image item Database Item property to Yes, Forms Builder understands that the data type is LONG RAW.

What Are Push Buttons?

Push buttons:

- Cannot display or represent data
- Are used to initiate an action
- Display as:
 - Text button 
 - Icon 

ORACLE®

10 - 12

Copyright © 2009, Oracle. All rights reserved.

What Are Push Buttons?

A *push button* is an interface object that you click to initiate an action. A push button is usually displayed as a rectangle with a descriptive label inside. Push buttons cannot store or display values.

You can enhance your form module by adding push buttons to provide quick and direct access to the most needed operations.

Push Button Styles

Forms Builder supports two push button styles:

- **Text button:** Displayed with a text label on the push button (a button with the Exit label in the screenshot)
- **Icon:** Displayed with a bitmapped graphic on the push button and often used in toolbars (a button with an icon of an exit door in the screenshot in the slide)

Push Button Actions

Use buttons to:

- Move input focus
- Display a list of values (LOV)
- Invoke an editor
- Invoke another window
- Commit data
- Issue a query
- Perform calculations

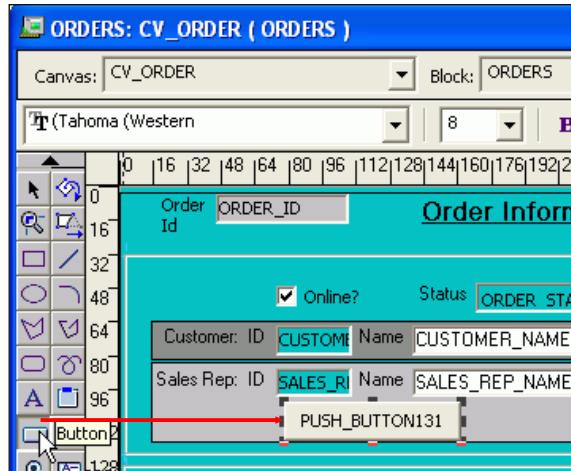


Push Button Actions

You use push buttons to enable users to perform some type of action. Clicking the button invokes code that is contained in a When-Button-Pressed trigger, as described in the lesson titled “Producing Triggers.” There is no default action performed by a button; you must code the action that you want a button to perform.

Note: Push buttons do not accept input focus on some window managers. On these platforms, the Keyboard Navigable property has no effect, and users can interact with the items only by using a mouse. Clicking a push button does not move the input focus on these platforms. The input focus remains in the item that was active before the user clicked the push button.

Creating a Push Button



ORACLE®

10 - 14

Copyright © 2009, Oracle. All rights reserved.

Creating a Push Button

A push button can be created by using:

- The Push Button tool in the Layout Editor
- The Create icon in the Object Navigator (creates a text item that you can convert to a push button)

How to Create a Push Button from the Layout Editor

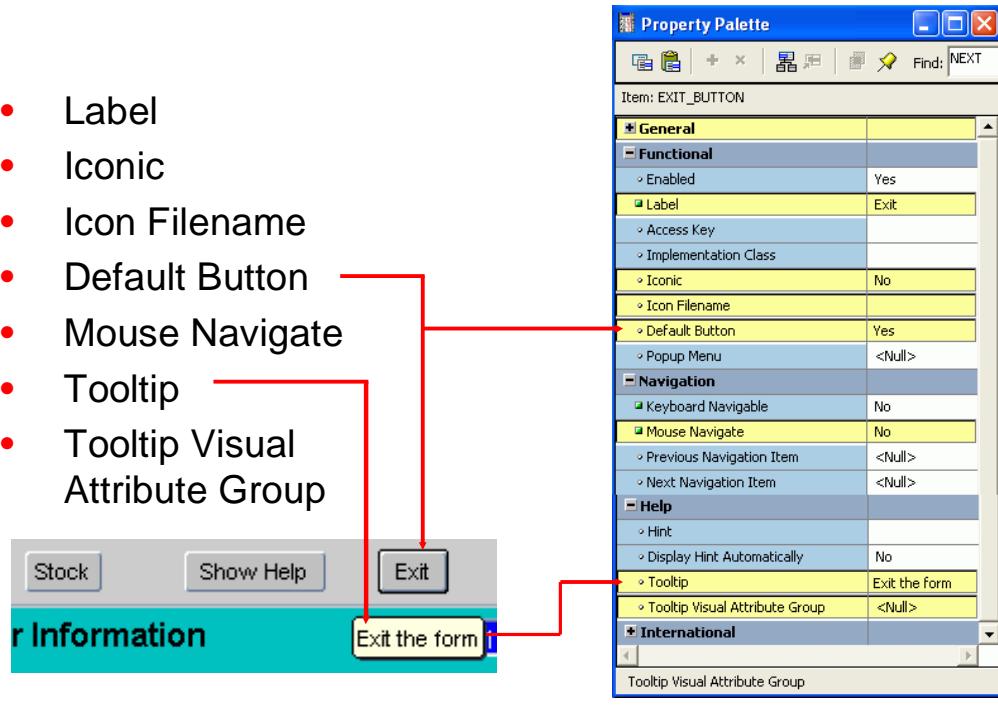
To create a push button from the Layout Editor, perform the following steps:

1. Invoke the Layout Editor.
2. Set the canvas and block to those where you want the push button to display.
3. Click the Push Button tool.
4. Click the canvas at the position where you want the push button to display. The screenshot shows the Layout Editor immediately after this step has been performed.
5. Double-click the push button. The Property Palette appears.
6. Change the name from PUSH_BUTTONXX to the required name.
7. Specify the other properties as required.

Note: Icon image files for buttons must be in GIF or JPEG format for run time, but can be .ico files to display at design time.

Setting Push Button Properties

- Label
- Iconic
- Icon Filename
- Default Button
- Mouse Navigate
- Tooltip
- Tooltip Visual Attribute Group



10 - 15

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Setting Push Button Properties

The slide shows the Property Palette of a button and how it appears at run time. Some of the properties to set for a button include:

- **Label:** Text label that appears on the button at run time; the example shows an Exit label
- **Iconic:** Indicates whether the button displays an icon instead of a label; example is no icon
- **Icon Filename:** Name of the file that contains the icon resource (file name only without the extension, such as list, not list.gif)
- **Default Button:** Indicates whether this is the default push button for the block, which can be selected implicitly by pressing [Enter] without the need to navigate or use the mouse (The default push button may be bordered or highlighted in a unique fashion to distinguish it from other push buttons. The screenshot of the run-time form, at the bottom left of the slide, shows the Exit button with a distinct border, indicating that it is the default button.)
- **Mouse Navigate:** Indicates whether Forms navigates to the item when you click it
- **Tooltip:** Help text that should appear in a tool tip beneath the button when the cursor moves over it; screenshots in the slide show a tool tip of "Exit the form"
- **Tooltip Visual Attribute Group:** Named visual attribute to apply to the tool tip at run time; the example in the slide shows no visual attribute applied

What Are Calculated Items?

Calculated items:

- Accept item values that are based on calculations
- Are read-only
- Can be expressed as:
 - Formula:
 - A calculated item value is the result of a horizontal calculation.
 - It involves bind variables.
 - Summary:
 - A calculated item value is a vertical calculation.
 - A summary is performed on values of a single item over all rows in a block.

ORACLE®

10 - 16

Copyright © 2009, Oracle. All rights reserved.

What Are Calculated Items?

With a calculated item, you can declaratively base item values on calculations involving one or more variable values. For example, you can use a calculated item to display a running total of employees' total compensation.

Any item that can store a value can be used as a calculated item by setting certain property values. However, calculated items are read-only, so you should generally use display items as calculated items.

Calculation Modes

Calculations can be expressed as a formula or as a summary of all items in a block. Forms Builder supports the following calculation modes:

- **Formula:** The calculated item value is the result of a horizontal calculation involving one or more bind variables, such as form items, global variables, and parameters. For example, the screenshot on the next slide shows the settings for Item_Total.
- **Summary:** The calculated item value is a vertical calculation involving the values of a single item over all rows within a single block. For example, the screenshot on the next slide also shows the settings for Order_Total.

Setting Item Properties for the Calculated Item

- Formula:
 - Calculation Mode
 - Formula
- Summary:
 - Calculation Mode
 - Summary Function
 - Summarized Block
 - Summarized Item

The screenshot illustrates the configuration of calculated items. At the top, two dialog boxes are shown: 'Order Total' and 'Item Total'. The 'Order Total' dialog has 'Calculation Mode' set to 'Formula' with the formula ':order_items.quantity * :order_items.unit_price'. The 'Item Total' dialog has 'Calculation Mode' set to 'Summary' with 'Summary Function' set to 'Sum'. Below these dialogs is a data grid representing an order. The grid has columns: Line Item Id, Product Id, Description, Unit Price, Quantity, and Item Total. The data rows are:

Line Item Id	Product Id	Description	Unit Price	Quantity	Item Total	
1	3106	KB 101/EN	48	61	2,928.00	
2	3114	MB - S900/650+	96.8	43	4,162.40	
3	3123	PS 220V /D	79	47	3,713.00	
4	3129	Sound Card STD	41	4	1,927.00	
					Order Total	46,257.00

Setting Item Properties for the Calculated Item

Unlike the other items covered so far in this lesson, there is no Item Type property value called Calculated Item. The calculation-specific properties of an item make it a calculated item. Text items and display items both support calculated items.

A calculated item can be created by:

- Setting the calculation-specific properties of any existing item that can store a value
- Creating a new item in the Layout Editor and setting its calculation-specific properties
- Using the Create icon to create a new item and setting calculation-specific properties

The following properties are specific to calculated items:

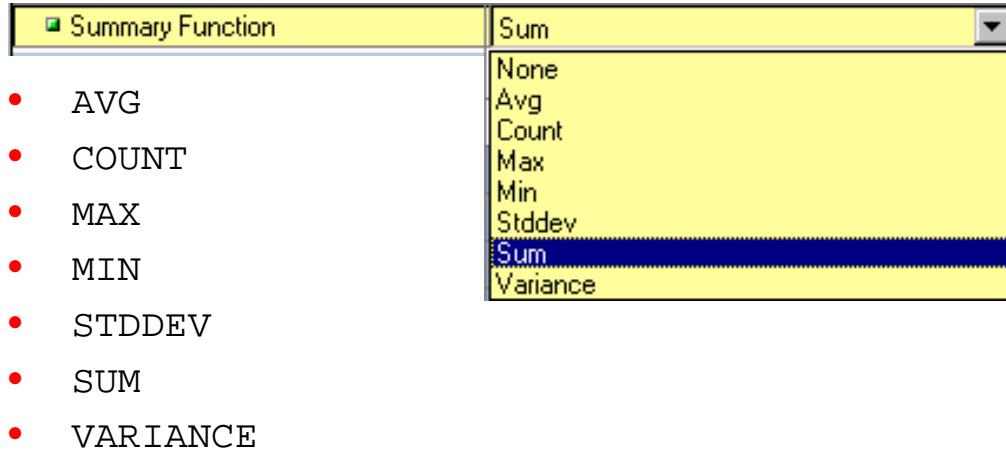
- **Calculation Mode:** The method of computing the calculated item values (None, Formula, or Summary)
- **Formula:** A single PL/SQL expression that determines the calculated value of a formula calculated item; can compute a value or call a subprogram
- **Summary Function:** The type of summary function (discussed on the following page) to be performed on the summary calculated item
- **Summarized Block:** The block over which all rows are summarized in order to assign a value to the summary calculated item; if not specified, current block is assumed
- **Summarized Item:** The item whose value is summarized in order to assign a value to the calculated item; required if the Calculation Mode is Summary

Setting Item Properties for the Calculated Item (continued)

The screenshots in the slide show the calculation properties in Property Palettes of two calculated items, Item_Total (a formula) and Order Total (a summary):

Property	Setting for Item Total	Setting for Order Total
Calculation Mode	Formula	
Formula	:order_items.quantity * :order_items.unit_price	
Summary Function		Sum
Summarized Block		<Null>
Summarized Item		ITEM_TOTAL

Using Summary Functions



ORACLE®

10 - 19

Copyright © 2009, Oracle. All rights reserved.

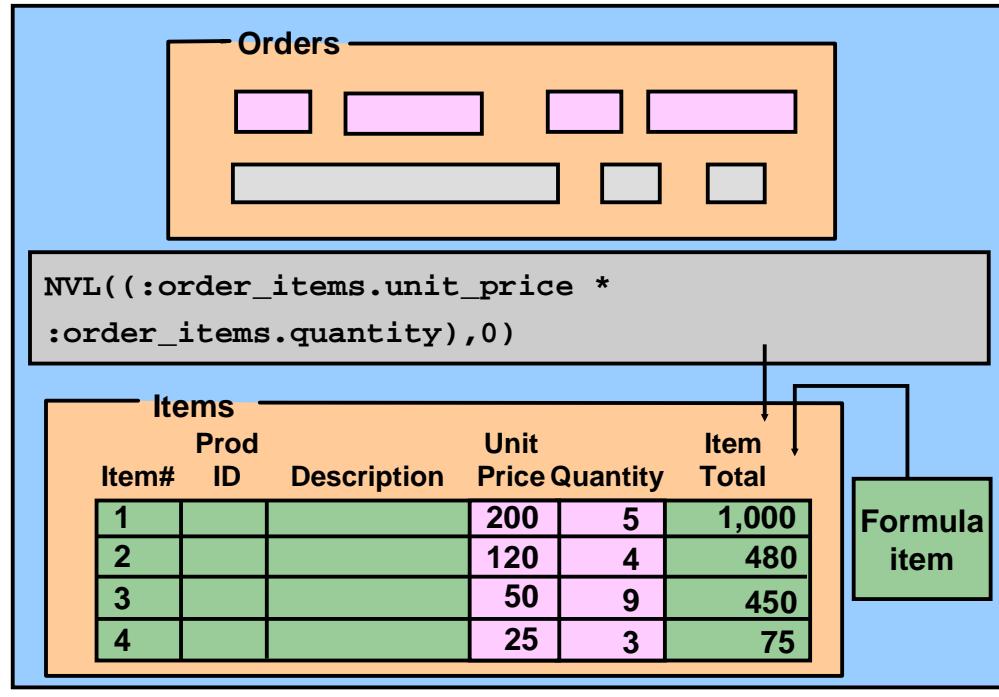
Using Summary Functions

You can use the following standard SQL aggregate functions for summary items:

- **AVG:** The average value (arithmetic mean) of the summarized item over all records in the block
- **COUNT:** Count of all non-NULL instances of the summarized item over all records in the block
- **MAX/MIN:** Maximum/minimum value of the summarized item over all records in the block
- **STDDEV:** The standard deviation of the summarized item's values over all records in the block
- **SUM:** Sum of all values of the summarized item over all records in the block
- **VARIANCE:** The variance (square of the standard deviation) of the summarized item's values over all records in the block

The screenshot shows that the Property Palette displays a drop-down list for the Summary Function property from which you can select one of these functions.

Creating a Calculated Item Based on a Formula



Creating a Calculated Item Based on a Formula

To create a calculated item based on a formula, perform the following steps:

1. Create a new item in the Object Navigator or the Layout Editor.
2. Open the Property Palette of the item.
3. Set the Calculation Mode property to Formula.
4. Click More for the Formula property and enter the PL/SQL expression to define the formula.

The slide graphic depicts setting a formula for the Item Total. Notice that Forms item names in the formula are preceded with a colon.

Note: A formula item cannot be a database item because its value is computed by Forms, not queried from a database column.

Rules for Calculated Item Formulas

Create calculated item formulas according to the following rules:

- A formula item must not invoke restricted built-ins.
- A formula item cannot execute any data manipulation language (DML) statements.
- Do not terminate a PL/SQL expression with a semicolon.
- Do not enter a complete PL/SQL assignment expression.



Rules for Calculated Item Formulas

When writing formulas for calculated items, observe the following rules:

- The formula (and any user-written subprogram that calls it) must not invoke any restricted built-ins.
- The formula (and any user-written subprogram that calls it) cannot execute any DML statements.
- Do not terminate the PL/SQL expression with a semicolon.
- If the PL/SQL expression involves an assignment, do not enter the complete PL/SQL statement. Forms Builder assigns the actual assignment code internally.

Example

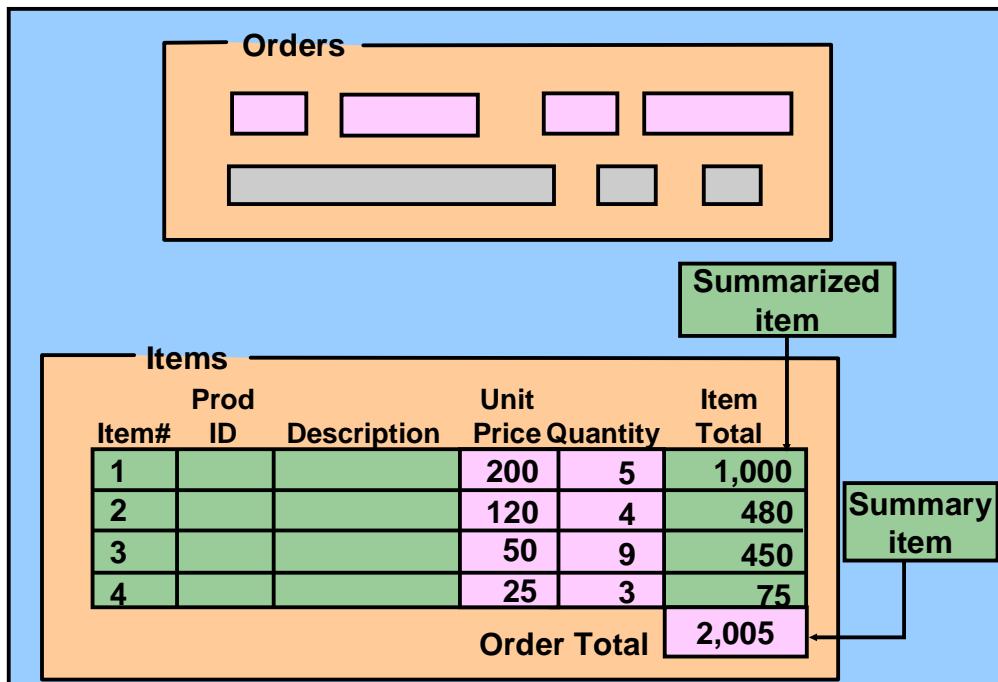
Suppose you have a formula item called Gross_Comp in the EMPLOYEES block of a form. If you set the Formula property to:

```
NVL(:employees.salary,0) * NVL(:employees.commission_pct,0)
```

Forms Builder internally converts this expression to a complete statement:

```
:employees.gross_comp := (NVL(:employees.salary,0) *  
NVL(:employees.commission_pct,0));
```

Creating a Calculated Item Based on a Summary



Creating a Calculated Item Based on a Summary

To create a calculated item based on a summary, perform the following steps:

1. Create a new item in the Object Navigator or the Layout Editor.
2. Open the Property Palette of the item.
3. Set the Calculation Mode property to Summary.
4. Select a function from the Summary Function pop-up list.
5. From the Summarized Block pop-up list, select a block over which all rows will be summarized.
6. From the Summarized Item pop-up list, select an item to be summarized.

The slide graphic depicts setting a summary for the Order_Total. The summarized item is Item_Total. In other words, the order total is the sum of all of the item totals for this order.

Note: A *summary item* is the calculated item to which you assign a value.

A *summarized item* is the item whose values are summarized and then assigned to the summary item.

Rules for Summary Items

- Summary items must reside in either:
 - The same block as the summarized item
 - Or
 - A control block with the Single Record property set to Yes
- Summarized item must reside in either:
 - A control block
 - Or
 - A data block with either the Query All Records property or the Precompute Summaries property set to Yes
- Data Type of summary item must be Number, unless using MAX or MIN.

ORACLE®

10 - 23

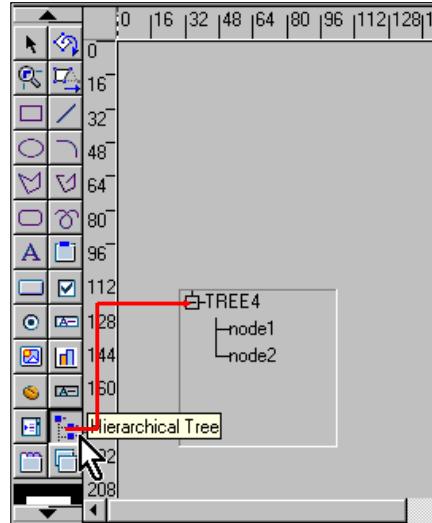
Copyright © 2009, Oracle. All rights reserved.

Rules for Summary Items

When creating calculated items based on a summary, observe the following rules:

- The summary item must reside in the same block as the summarized item, or in a control block whose Single Record property is set to Yes.
- The summarized item must reside in a control block, or in a data block whose Query All Records property or the Precompute Summaries property is set to Yes.
Note: This ensures that records fetched in the block and the summarized value are consistent. Otherwise, another user may possibly update a record that has not been fetched yet.
- Set the Data Type property for a summary item to Number, unless the summary function is Max or Min, in which case the data type must mirror its associated summarized item. For example, a calculated item that displays the most recent (maximum) date in the HIRE_DATE column must have a data type of Date.
- If the summarized item values are based on a formula, the summarized item must reside in a block whose Query All Records property is set to Yes.

Creating a Hierarchical Tree Item



ORACLE®

Creating a Hierarchical Tree Item

A hierarchical tree is an item that displays data in the form of a standard navigator.

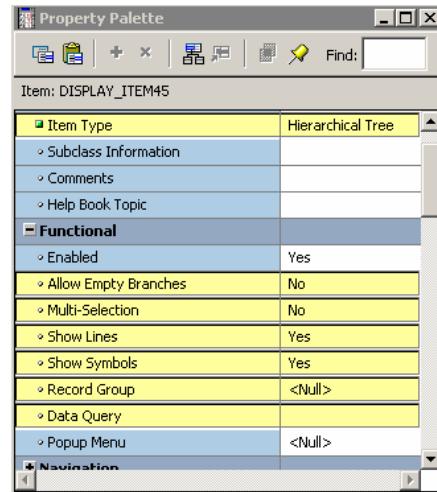
How to Create a Hierarchical Tree Item

To create a hierarchical tree item, do one of the following:

- **In the Layout Editor:**
 - Click the Hierarchical Tree icon.
 - Click and drag on the canvas to create the hierarchical tree object. The screenshot shows the Layout Editor immediately after this step has been performed.
 - Set other hierarchical tree-related properties as required.
- **In the Object Navigator:**
 - Create a new item by using the Create icon.
 - Open the item's Property Palette and set the Item Type property to Hierarchical Tree.
 - Set other hierarchical tree-related properties as required.

Setting Hierarchical Tree Item Properties

- Allow Empty Branches
- Multi-Selection
- Show Lines
- Show Symbols
- Record Group
- Data Query



Setting Hierarchical Tree Item Properties

Hierarchical Tree properties include the following (as you see in the screenshot of the Property Palette showing the default property values for a hierarchical tree):

- **Item Type:** Must be set to Hierarchical Tree
- **Allow Empty Branches:** Indicates whether branch nodes may exist with no children (if set to the default value of No, branch nodes with no children will be converted to leaf nodes; if set to Yes, an empty branch will be displayed as a collapsed node)
- **Multi Selection:** Indicates whether multiple nodes may be selected at one time (default No)
- **Show Lines:** Indicates whether a hierarchical tree displays lines leading up to each node (default Yes)
- **Show Symbols:** Indicates whether a hierarchical tree should display the + or – symbol in front of each branch node (default Yes)
- **Record Group:** Name of the record group from which the hierarchical tree derives its values (default <Null>)
- **Data Query:** Specifies the query-based data source (default blank)

Several built-ins are available to manipulate hierarchical trees. For more information, see the lesson titled “Adding Functionality to Items,” which discusses how to implement a hierarchical tree in a form.

Note: A hierarchical tree must be the only item in the data block.

What Are Bean Area Items?

The bean area item enables you to:

- Add a JavaBean to a form
- Extend Forms functionality
- Interact with the client machine
- Reduce network traffic



ORACLE®

10 - 26

Copyright © 2009, Oracle. All rights reserved.

What Are Bean Area Items?

A JavaBean is a component written in Java that can plug in to any applet or Java application. The bean area item enables you to extend Forms functionality by adding a JavaBean to your form. The slide graphic, a cup of coffee surrounded by beans, symbolizes JavaBeans.

With JavaBeans, you can interact with the client machine, performing such functions as:

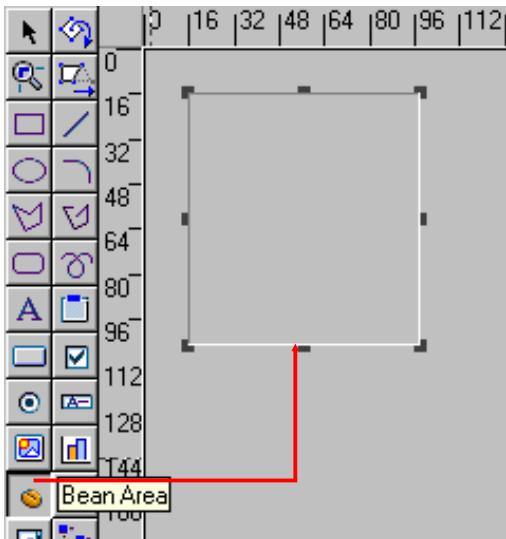
- Obtaining information about the client machine
- Uploading client files to the server machine
- Modifying the user interface on the client
- Checking the spelling of a text item
- Displaying a progress bar, clock, calendar, or color picker with which the operator may be able to interact and select values

Some of this functionality, such as the calendar, is possible using Forms native functionality. However, using a JavaBean enables client interaction without generating network traffic.

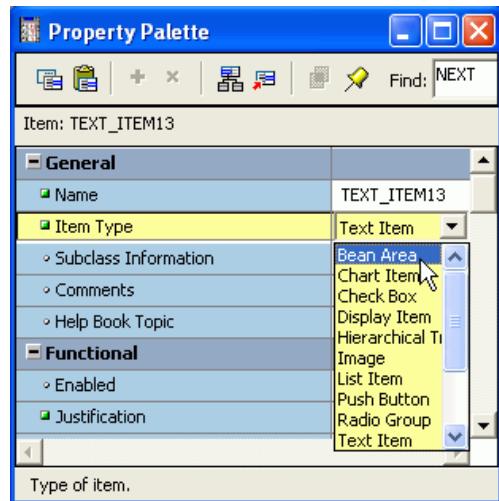
Although JavaBeans can be used to input data, as in the case of the Calendar JavaBean, the bean area item itself does not accept user input.

Creating a Bean Area Item

Create a bean area
in the Layout Editor.



Convert an existing item
to a bean area.

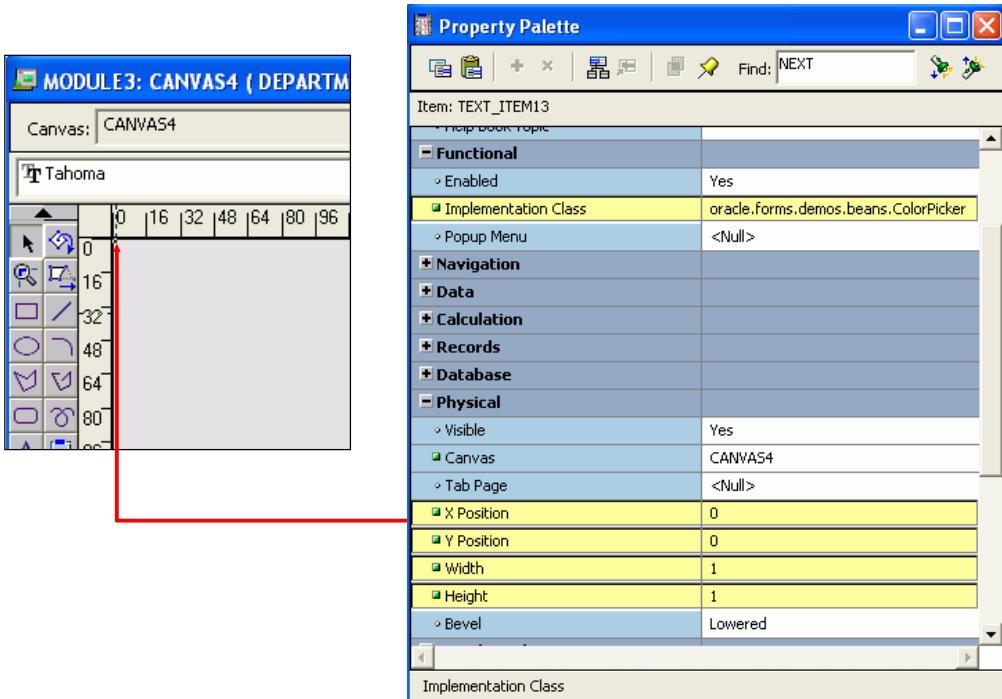


Creating a Bean Area

You create a bean area using the Bean Area tool in the Layout Editor. Click the tool and drag out an area on the canvas. At first the area will look like an empty rectangle, as you see in the screenshot at the left of the slide.

You can also create a bean area from any existing item by changing its item type to Bean Area in the Property Palette. The screenshot at the right of the slide shows that Bean Area is one of the selections from the Item Type drop-down list in the Property Palette for an item.

Setting Bean Area Item Properties



Setting Bean Area Item Properties

The most important bean area property is Implementation Class. This property is used to specify the fully qualified name of the Java class that the bean area item should instantiate.

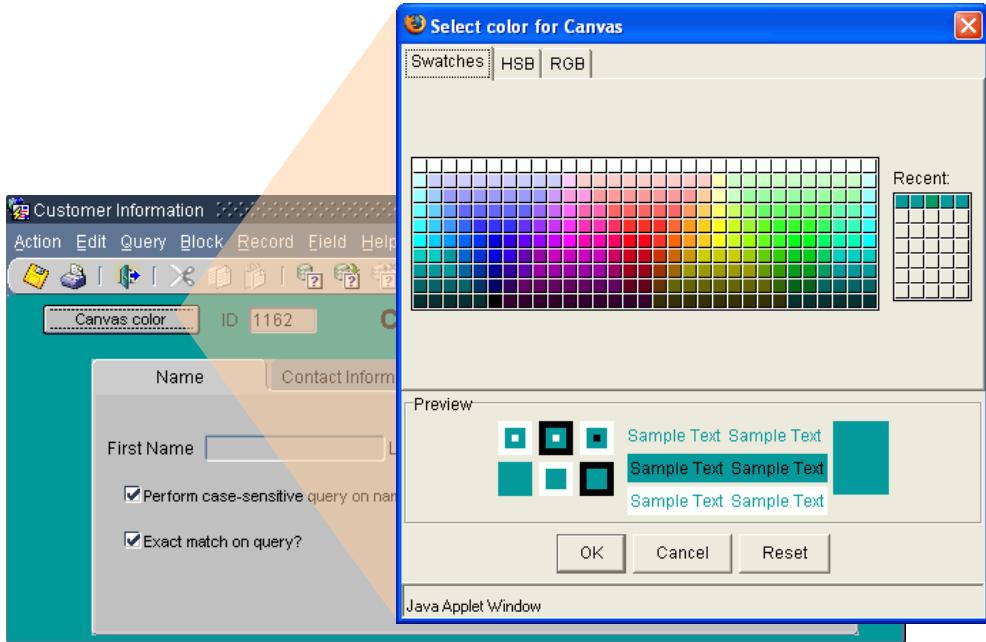
It is not necessary to set the Implementation Class at design time. Instead, you can programmatically register the bean at run time. This is discussed in the lesson titled "Adding Functionality to Items."

If the JavaBean has a visible component, Forms Builder displays the JavaBean within the bean area item in the Layout Editor after the Implementation Class is set. You can set the size and position of the bean area visually in the Layout Editor, or by changing the X/Y Position, Width, and Height properties.

Some JavaBeans have no visible component. To avoid the distraction of a large empty rectangle, you can set the properties so that the bean area is displayed as a tiny dot in the corner of the canvas. This is depicted by the screenshots in the slide, where Width and Height are set to 1 and the bean area shows up in the Layout Editor as a dot.

Alternatively, you can set its Visible property to No, which ensures that the bean area does not display at run time, although you still see it in the Layout Editor.

Interacting with the JavaBean at Run Time



ORACLE®

Interacting with the JavaBean at Run Time

You may programmatically define functionality for the JavaBean to perform. At run time the user may interact with the JavaBean to initiate this functionality.

In the example in the slide, there is a button that invokes the ColorPicker JavaBean, which displays to the user a color picker window. When the operator selects a color from the color picker, it is used to programmatically set the background color of the canvas.

Programming responses to user interaction is discussed in the lesson titled “Adding Functionality to Items.”

Summary

In this lesson, you should have learned that:

- The following item types do not allow input:
 - Display items (to show non-base-table information)
 - Image items (to display an image)
 - Push buttons (to initiate action)
 - Calculated items (to display the results of a calculation)
 - Hierarchical tree items (to display related data in a hierarchical fashion)
 - Bean area items (to execute client-side Java code)
- You create noninput items by:
 - Changing the type of an existing item and setting certain properties
 - Using the appropriate tool in the Layout Editor

ORACLE®

10 - 30

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you should have learned that:

- Display items display non-base-table information or derived values
- Image items store and display vector or scanned bitmapped images
- Push Buttons initiate an action
- Calculated items base item values on calculations. Calculations can be expressed in one of the following modes:
 - Formula
 - Summary
- Hierarchical trees display information in an Object Navigator style display
- Bean Area items enable you to integrate Java components into your application
- You create the above item types by:
 - Using the appropriate tool in the Layout Editor
 - Clicking Create in the Object Navigator (this creates a text item that you can convert to a different item type)
 - Changing the item type and/or setting appropriate properties on existing items

Practice 10: Overview

This practice covers the following topics:

- Creating display items
- Creating an image item
- Creating buttons with icons
- Creating calculated items:
 - Formula
 - Summary
- Creating a bean area item



Practice 10: Overview

In this practice, you add several items in the Customers and Orders forms: display items, image item, push buttons, calculated items, and a bean area.

- In the Orders form:
 - Create two display items in the ORDER_ITEMS block.
 - Create an image item in the CONTROL block.
 - Create a button that displays an icon in the CONTROL block.
 - Base the Item_Total item in the ORDER_ITEMS block on a formula.
 - Create a control item in the CONTROL block. Base this item value on a summary that displays the total value of an order.
- In the Customers form:
 - Create an iconic button in the CONTROL block.
 - Create a bean area.
- Save and run the Orders and Customers forms.

The calculated items function immediately because their functionality is defined declaratively. However, the other item types that you create in this practice do not function until programmatic logic is added in later lessons.

Creating Windows and Content Canvases



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe the relationship between windows and content canvases
- Create windows and content canvases
- Display a form module in multiple windows
- Display a form module on multiple layouts

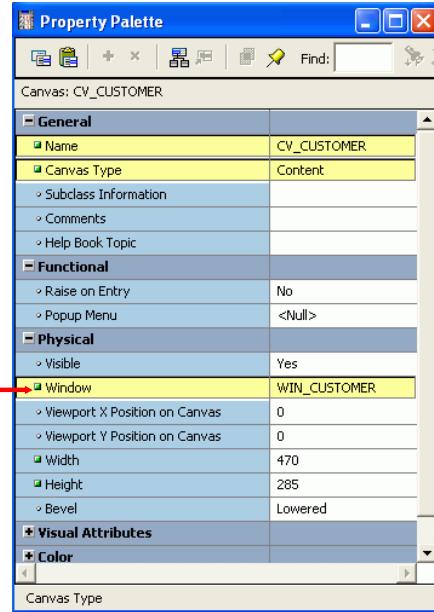


Lesson Aim

With Oracle Forms Builder, you can take advantage of the graphical user interface (GUI) environment by displaying a form module across several canvases and in multiple windows. This lesson familiarizes you with the window object and the default canvas type, the content canvas.

What Are Windows, Canvases, and Viewports?

- Window: Container for Forms Builder visual objects
- Canvas: Surface on which you “paint” visual objects;
to see a canvas and its objects, display the canvas in a window.
- Viewport: Part of the canvas visible in the window



11 - 3

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

What Are Windows, Canvases, and Viewports?

With Forms Builder, you can display an application in multiple windows by using its display objects—windows and canvases.

Window

A window is a container for all visual objects that make up a Forms application. It is similar to an empty picture frame. The window manager provides the controls for the window that enable such functionality as scrolling, moving, and resizing. You can minimize a window. A single form may include several windows.

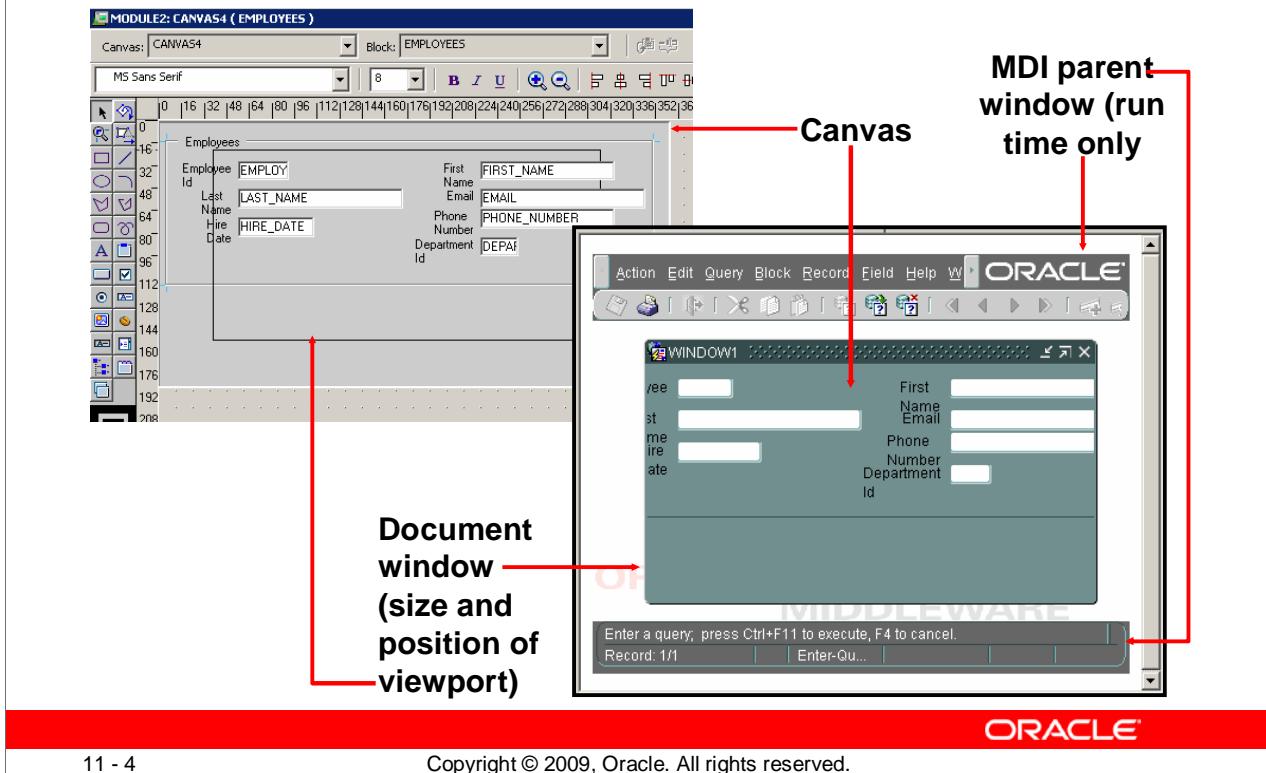
Canvas

A canvas is a surface inside a window container on which you place visual objects, such as interface items and graphics. It is similar to the canvas on which a picture is painted. To see a canvas and its contents at run time, you must display it in a window. A canvas always displays in the window to which it is assigned.

Viewport

A viewport is an attribute of a canvas. It is effectively the visible portion of, or view onto, the canvas. When you resize a viewport, the window in which the canvas is displayed is resized.

Comparing Design Time with Run Time



Comparing Design Time with Run Time

The slide shows windows and canvases at design time and at run time.

The viewport at design time has been moved and resized so that the items on the canvas are only partially enclosed in the viewport. At run time, the slide shows that the document window is initially the same size and shows the same area as the design-time viewport, although the user can usually resize this window. The canvas in this example is larger than the viewport. The canvas must completely contain all of its items, or you are unable to run the form.

There is an additional window, the MDI (Multiple Document Interface) window, that at run time displays the menu and toolbar, if any. It also displays the message and status areas, in addition to any document windows that may be open.

Using a Content Canvas

- A content canvas is the “base” canvas.
- Its view occupies the entire window.
- It is the default canvas type.
- Each window should have at least one content canvas.

ORACLE®

11 - 5

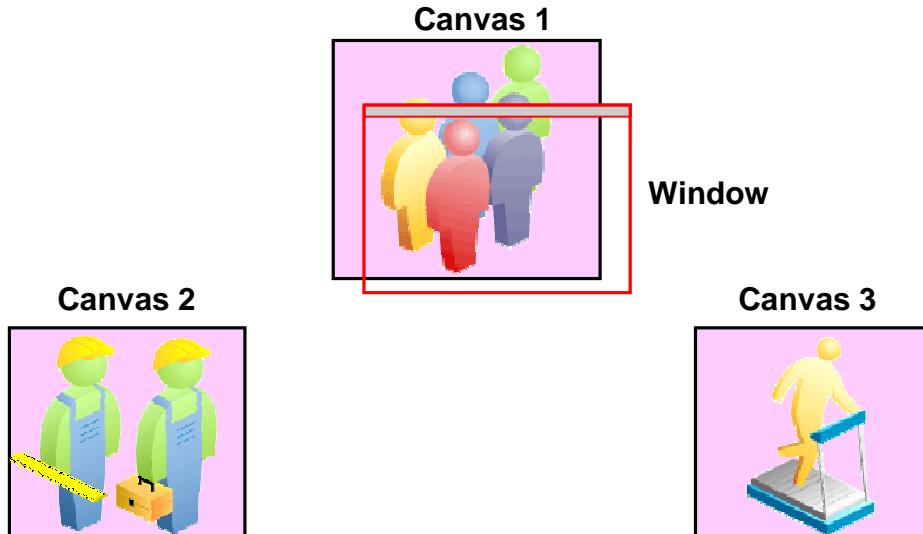
Copyright © 2009, Oracle. All rights reserved.

Using a Content Canvas

Forms Builder offers different types of canvases. A content canvas is the base canvas that occupies the entire content pane of the window in which it is displayed. The content canvas is the default canvas type. Most canvases are content canvases.

Note: Each item in a form must refer to no more than one canvas. An item displays on the canvas to which it is assigned, through its Canvas property. If the Canvas property for an item is left unspecified, that item is said to be a Null-canvas item and will not display at run time.

Relationship Between Windows and Content Canvases



Relationship Between Windows and Content Canvases

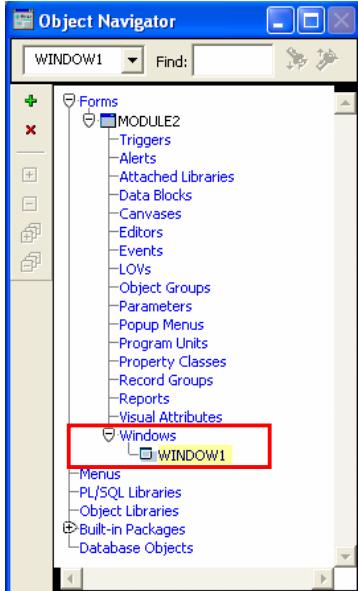
You must create at least one content canvas for each window in your application. When you run a form, only one content canvas can be displayed in a window at a time, even though more than one content canvas can be assigned to the same window at design time.

At run time, a content canvas always completely fills its window. As the user resizes the window, Forms resizes the canvas automatically. If the window is too small to show all items on the canvas, Forms automatically scrolls the canvas to bring the current item into view.

The slide shows three canvases and one window. All three canvases are assigned to the one window. The first canvas is displayed in the window. The other canvases are not currently displayed; however, if a user navigates to an item on one of these canvases, it appears in the window and the first canvas is no longer visible.

Note: You can assign multiple content canvases to a window; however, only one content canvas can be displayed in a window at a time.

Using the Default Window



WINDOW1 is created by default with each new form module.

- It is modeless.
- You can delete, rename, or change its attributes.

ORACLE®

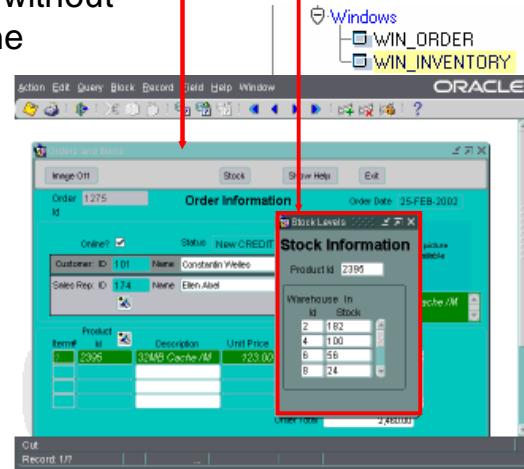
Using the Default Window

When you create a new form module, Forms Builder creates a new window implicitly. Thus, each new form module has one predefined window, which is called WINDOW1. You can delete or rename WINDOW1, or change its attributes.

The screenshot in the slide shows that the default window is the only object that is part of a new form.

Displaying a Form Module in Multiple Windows

- Use additional windows to:
 - Display two or more content canvases at once and switch between them without replacing the initial one
 - Modularize form contents
 - Take advantage of the window manager
- Two types of windows:
 - Modal
 - Modeless



ORACLE®

Displaying a Form Module in Multiple Windows

Creating additional windows provides the ability to do the following:

- Display two or more content canvases at once
- Switch between canvases without replacing the initial one
- Modularize the form contents into multiple layouts
- Take advantage of window manager functionality, such as allowing the user to resize or close a window

Types of Windows

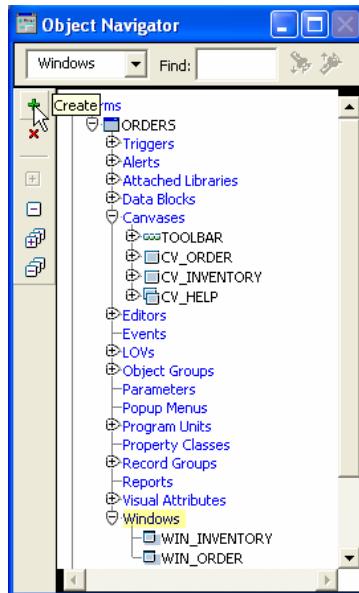
You can create two types of windows:

- A *modal window* is a restricted window that the user must respond to before moving the input focus to another window. Modal windows must be dismissed before control can be returned to a modeless window. They become active as soon as they are displayed, and require that you define a means of exit or dismissal.
- A *modeless window* is an unrestricted window that the user can exit freely. Modeless windows, which are the default window type, can be displayed at the same time as multiple other modeless windows. They are not necessarily active when displayed.

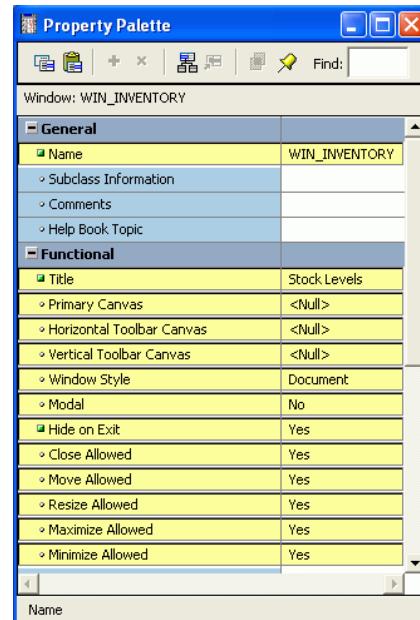
The screenshot shows a form's main window displaying orders, with an additional modeless window displaying inventory levels of a selected product.

Creating a New Window

Object Navigator: Click Create with the Windows node selected.



Property Palette: Set properties.



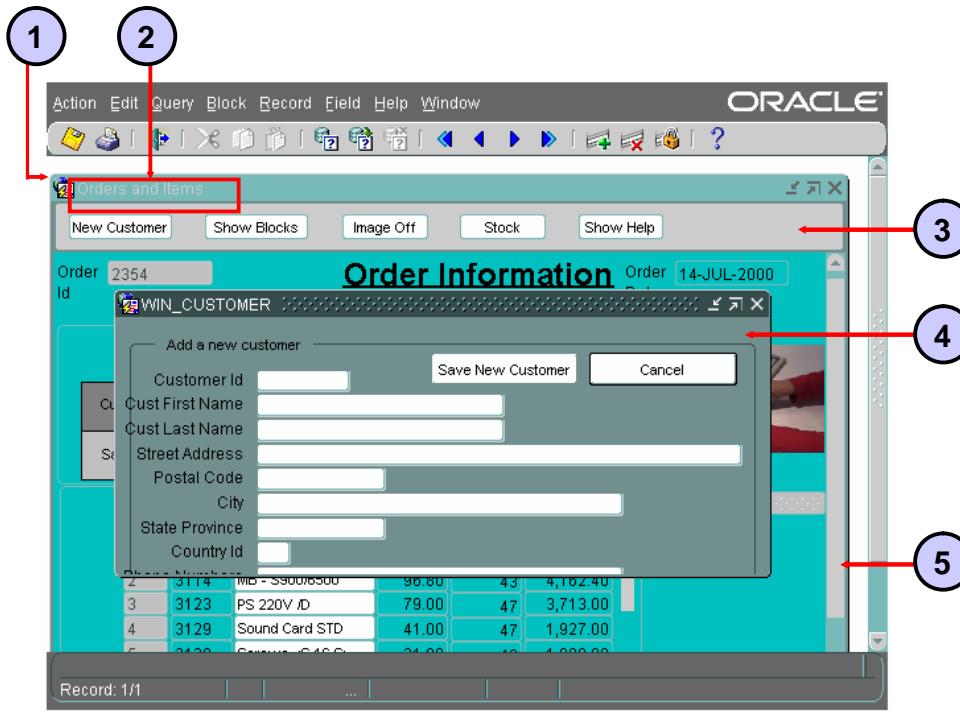
Creating a New Window

To create a new window, perform the following steps:

1. Select the Windows node in the Object Navigator.
2. Click Create. A new window entry displays, with a default name of WINDOWXX.
3. If the Property Palette is not already displayed, double-click the window icon to the left of the new window entry.
4. Set the window properties according to your requirements. The screenshot at the right of the slide shows the Property Palette of a window.

Note: For your new window to display, you must specify its name in the Window property of at least one canvas. To display a console to end users, set the form-level property Console Window to the window in which you want to display the console. To hide the console, set the property to <Null>.

Setting Window Properties



11 - 10

Copyright © 2009, Oracle. All rights reserved.

Setting Window Properties

You set properties for windows to determine their behavior and appearance. The following are some of the properties, which correspond to the numbered areas in the slide screenshot:

1. **X/Y Position:** Specifies the location of the window within the containing window
2. **Title:** The title to be displayed; if not specified, uses the name indicated by the window Name property
3. **Horizontal/Vertical Toolbar Canvas:** Specifies the toolbar canvas to be displayed horizontally across the top or vertically to the left of the window; selected from all horizontal/vertical toolbar canvases assigned to this window
4. **Modal:** Specifies if the window is modal, requiring the user to dismiss the window before any other user interaction can continue, such as the Add Customer window in the slide, where users must dismiss the window to continue in the Orders form
5. **Show Horizontal/Vertical Scroll Bar:** Specifies whether a horizontal or vertical scroll bar should display in the window
6. **Hide on Exit:** Indicates whether Forms hides the window automatically when the end user navigates to an item in another window (not shown in the slide)

For a description of other properties that affect the behavior of windows, click the property in the Property Palette and press F1.

Using GUI Hints

- GUI hints are recommendations to the window manager about appearance and functionality of a window.
- If the window manager supports a specific GUI hint and its property is set to Yes, it will be used.
- Functional properties for GUI hints:
 - Close Allowed
 - Move Allowed
 - Resize Allowed
 - Maximize Allowed
 - Minimize Allowed
 - Inherit Menu

ORACLE®

11 - 11

Copyright © 2009, Oracle. All rights reserved.

Using GUI Hints

There are certain properties under the Functional group that enable you to make recommendations about window appearance and functionality:

- **Close Allowed:** Enables the mechanism for closing the window (Forms responds to a user's attempts to close the window by firing a When-Window-Closed trigger.)
- **Move/Resize Allowed:** Specifies whether a user can move and resize the window
- **Maximize/Minimize Allowed:** Specifies whether a user can zoom or iconify the window
- **Inherit Menu:** Specifies whether the window displays the current form menu

Displaying a Form Module on Multiple Layouts

PROPERTIES:

Canvas

CV_ORDER

Window:

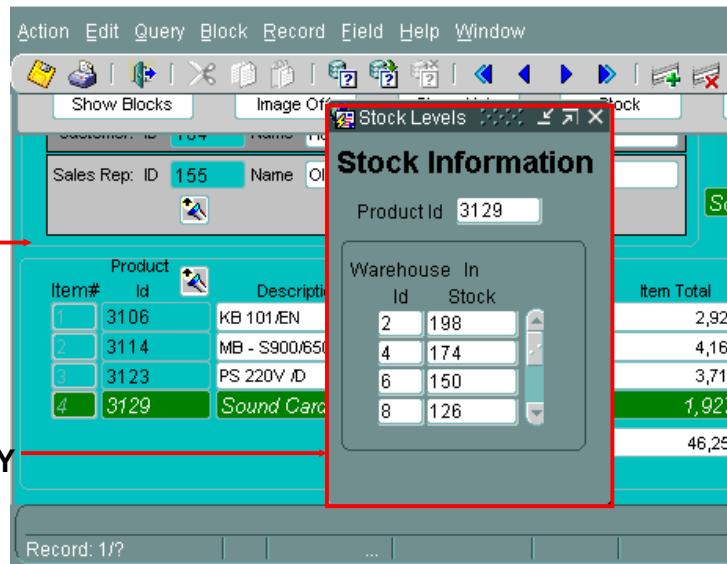
WIN_ORDERS

Canvas

CV_INVENTORY

Window:

WIN_INVENTORY



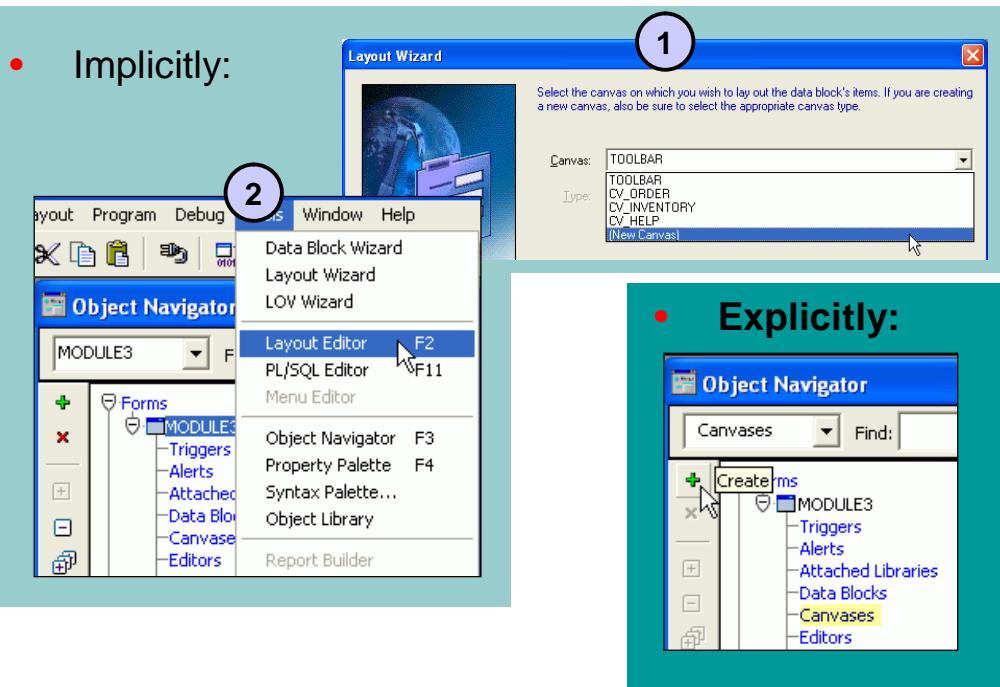
ORACLE®

Displaying a Form Module on Multiple Layouts

You can have more than one content canvas in your Forms application. However, remember that only one content canvas can be displayed in a window at one time. To display more than one content canvas at the same time, you can assign each content canvas to a different window.

Now you can display the form module on multiple layouts or surfaces. The screenshot shows two content canvases that are displayed simultaneously, each in a separate window. One window (WIN_ORDERS) displays the canvas containing orders and order items (CV_ORDER). The other window (WIN_INVENTORY) displays the canvas showing inventory levels of a selected product (CV_INVENTORY.).

Creating a New Content Canvas



11 - 13

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Creating a New Content Canvas

The creation of a content canvas can be implicit or explicit.

Implicitly Creating a Content Canvas

There are two ways of implicitly creating a new content canvas:

1. **Layout Wizard:** When you use the Layout Wizard to arrange data block items on a canvas, the wizard enables you to select a new canvas on its Canvas page. In this case, the wizard creates a new canvas with a default name of CANVASXX. (See the top-right screen shot in the slide.)
2. **Layout Editor:** When there are no canvases in a form module and you invoke the Layout Editor, as shown in the screenshot at the left of the slide, Forms Builder creates a default canvas on which you can place items.

Explicitly Creating a Content Canvas

You can create a new content canvas explicitly by using the Create icon in the Object Navigator, as shown in the screenshot at the bottom-right of the slide.

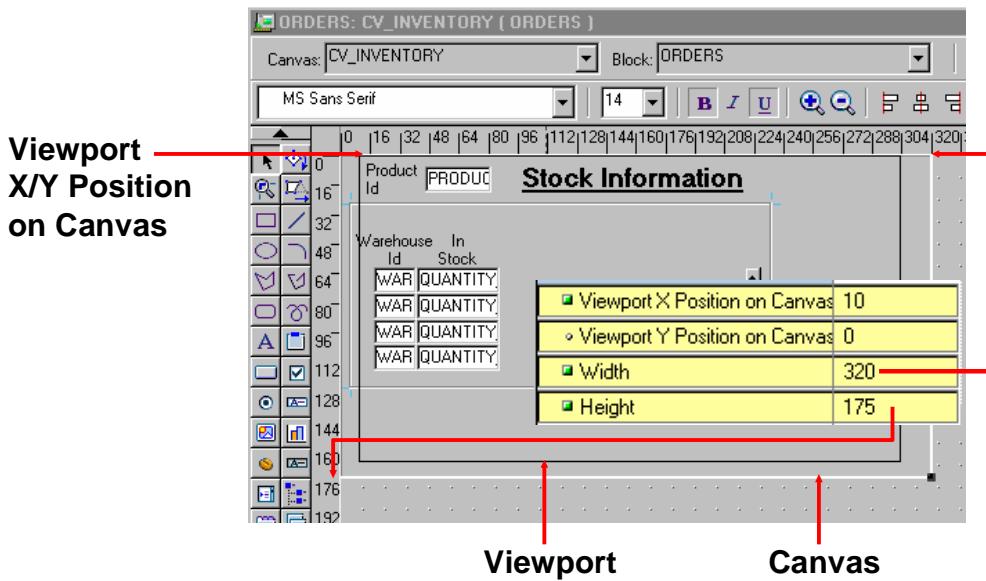
Creating a New Content Canvas (continued)

To explicitly create a new content canvas, perform the following steps:

1. Click the Canvases node in the Object Navigator.
2. Click the Create icon. A new canvas entry appears with a default name of CANVASXX.
3. If the Property Palette is not already displayed, select the new canvas entry. Then select Tools > Property Palette.
4. Set the canvas properties according to your requirements.

Note: Double-clicking the icon for a canvas in the Object Navigator invokes the Layout Editor instead of the Property Palette.

Setting Content Canvas Properties



Setting Content Canvas Properties

You can set properties to determine how the canvas is to be displayed. Several properties in the Physical group are shown in the slide: Width, Height, and Viewport X/Y Position on Canvas. For a canvas to display at run time, its Window property must also be specified and its Visible property must be set to Yes.

In the General group, you can choose the Canvas Type. (This lesson explains the content canvas. For information about other canvas types, refer to the lesson titled “Working with Other Canvas Types.”)

In the Functional group, you can set “Raise on Entry” to affect whether the canvas is always brought to the front of the window when the user navigates to an item on the canvas. You use this property when the canvas is displayed in the same window with other canvases. Forms always ensures that the current item is visible. Even if “Raise on Entry” is set to No, Forms will bring the canvas to the front of the window if the user navigates to an item on the canvas that is hidden behind other canvases.

Performance Tip: To reduce the time that is needed to display the initial screen, keep the number of items initially displayed to a minimum. You can hide elements, such as canvases, that are not immediately required. To do this, set the canvas “Raise on Entry” property to Yes, and set Visible to No.

Summary

In this lesson, you should have learned that:

- Windows can display multiple content canvases, but can display only one canvas at a time
- Content canvases are displayed only in the window to which they are assigned
- You must assign at least one content canvas to each window in your application
- You create windows in the Object Navigator; one is created by default with each new module
- You create canvases in the Object Navigator, by using the Layout Wizard, or by invoking the Layout Editor in a module without a canvas
- You can display multiple layouts by assigning canvases to different windows

ORACLE®

11 - 16

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you should have learned:

- About the relationship between windows and content canvases
- How to create a new window
- How to create a new content canvas
- How to display a form module on multiple layouts by displaying each canvas in a separate window

Practice 11: Overview

This practice covers the following topics:

- Changing a window size, position, name, and title
- Creating a new window
- Displaying data block contents in the new window



Practice 11: Overview

In this practice, you customize windows in your form modules. You resize the windows to make them more suitable for presenting canvas contents. You also create a new window to display the contents of the INVENTORIES block.

- Change the size and position of the window in the Customers form. Change its name and title. Save and run the form.
- Modify the name and title of the window in the Orders form.
- Create a new window in the Orders form. Make sure the contents of the INVENTORIES block display in this window. Save and run the form.

12

Working with Other Canvas Types

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe the different types of canvases and their relationships to each other
- Identify the appropriate canvas type for different scenarios
- Create an overlay effect by using stacked canvases
- Create a toolbar
- Create a tabbed interface

ORACLE®

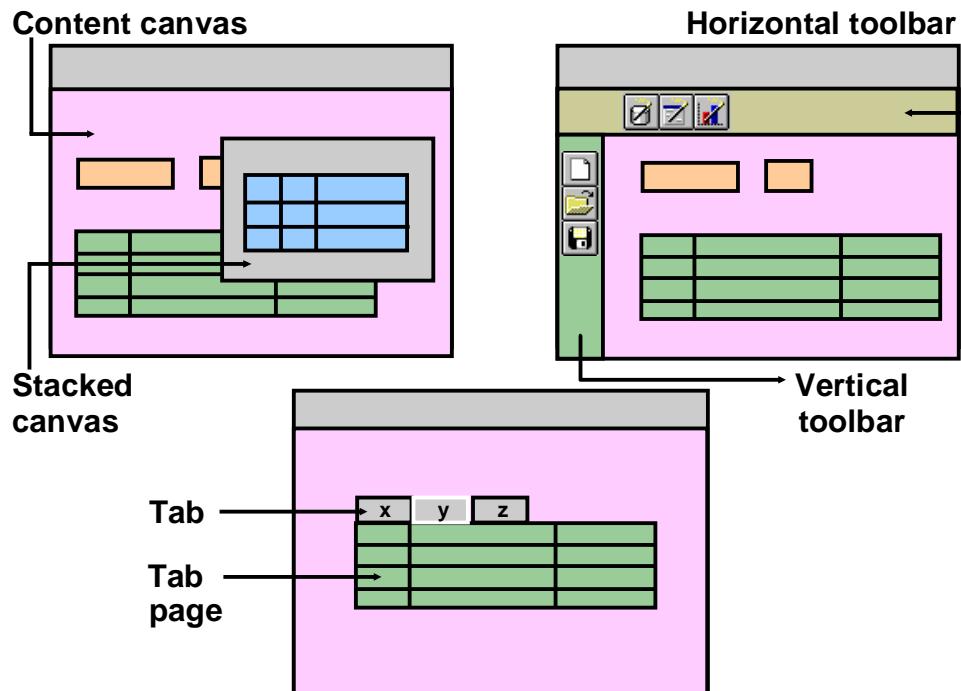
12 - 2

Copyright © 2009, Oracle. All rights reserved.

Lesson Aim

In addition to the content canvas, Oracle Forms Builder enables you to create three other canvas types. This lesson introduces you to the stacked canvas, which is ideal for creating overlays in your application. It also presents the toolbar canvas and the tabbed canvas, both of which enable you to provide a user-friendly GUI application.

Overview of Canvas Types



ORACLE®

Overview of Canvas Types

In addition to the content canvas, Forms Builder provides three other types of canvases:

- **Stacked canvas:** The slide shows a smaller stacked canvas appearing on top of the content canvas, obscuring part of the content canvas.
- **Toolbar canvas:** The slide shows a vertical toolbar to the left of the canvas and a horizontal toolbar at the top of the canvas; toolbars do not obscure any of the content canvas.
- **Tab canvas:** The slide shows that the user can select different tabs to display different information on the canvas.

When you create a canvas, you specify its type by setting the Canvas Type property. The type determines how the canvas is displayed in the window to which it is assigned.

What Is a Stacked Canvas?

- The stacked canvas is displayed on top of a content canvas.
- It shares a window with a content canvas.
- Its size is:
 - Usually smaller than the content canvas in the same window
 - Determined by viewport size
- It is created in:
 - Layout Editor (usually easier)
 - Object Navigator

ORACLE®

12 - 4

Copyright © 2009, Oracle. All rights reserved.

What Is a Stacked Canvas?

A *stacked canvas* is displayed on top of, or stacked on, the content canvas that is assigned to a window. It shares a window with a content canvas and any number of other stacked canvases. Stacked canvases are usually smaller than the window in which they display.

You can create a stacked canvas in either of the following:

- Layout Editor (usually easier because you can set properties visually)
- Object Navigator

Determining the Size of a Stacked Canvas

Stacked canvases are typically smaller than the content canvas in the same window. You can determine the stacked canvas dimensions by setting the Width and Height properties. You can determine the view dimensions of the stacked canvas by setting the Viewport Width and Viewport Height properties.

Typical Usage of a Stacked Canvas

If a data block contains more items than the window can display, Forms scrolls the window to display items that were initially not visible. This can cause important items, such as primary key values, to scroll out of view. By placing important items on a content canvas, and then placing the items that can be scrolled out of sight on a stacked canvas, the stacked canvas becomes the scrolling region, rather than the window itself.

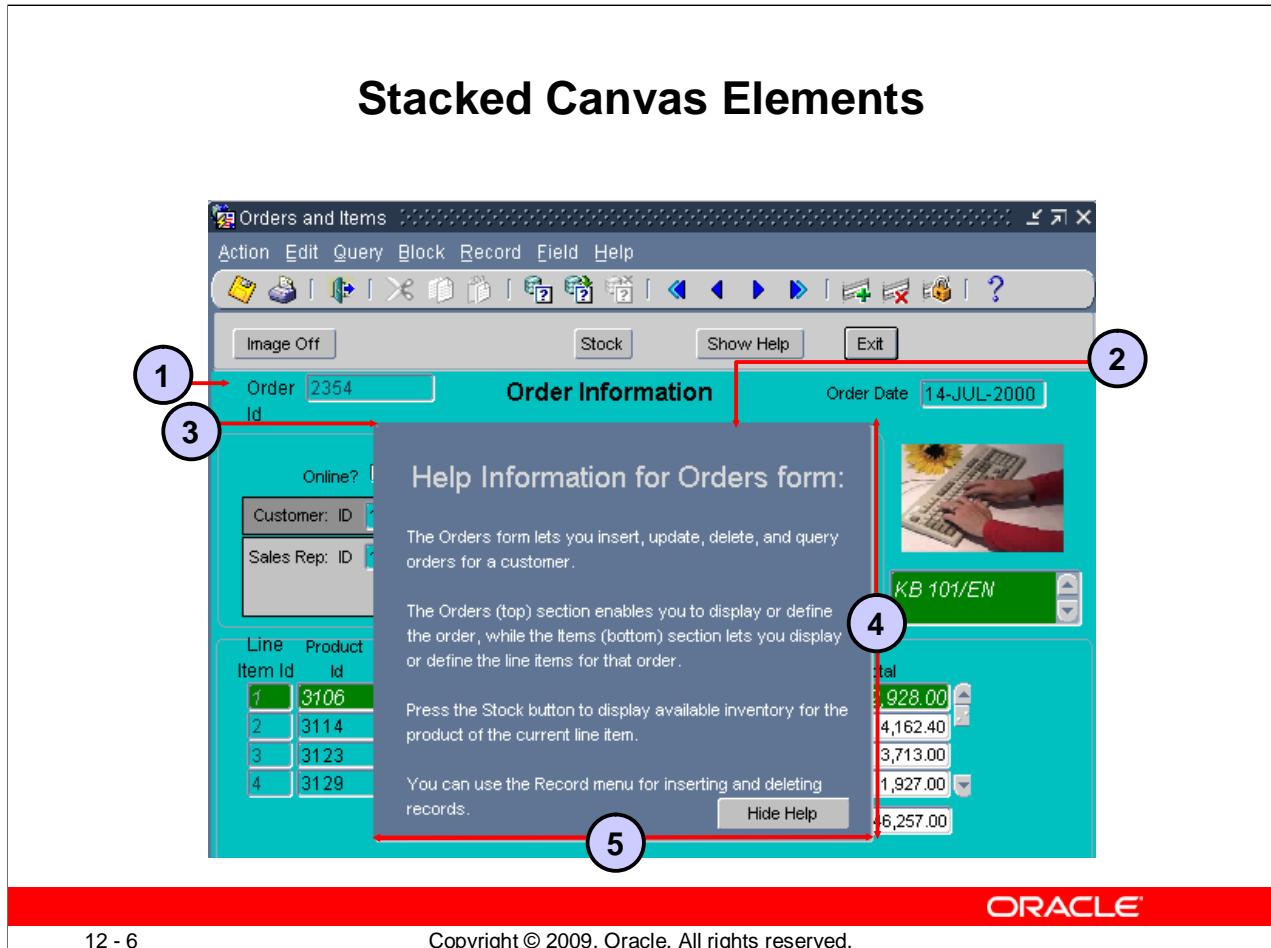
What Is a Stacked Canvas? (continued)

Typical Usage of a Stacked Canvas

With stacked canvases, you can achieve the following:

- Scrolling views
- Creating an overlay effect within a single window
- Displaying headers with constant information, such as company name
- Creating a cascading or a revealing effect within a single window
- Displaying additional information
- Displaying information conditionally
- Displaying context-sensitive help
- Hiding information

Stacked Canvas Elements

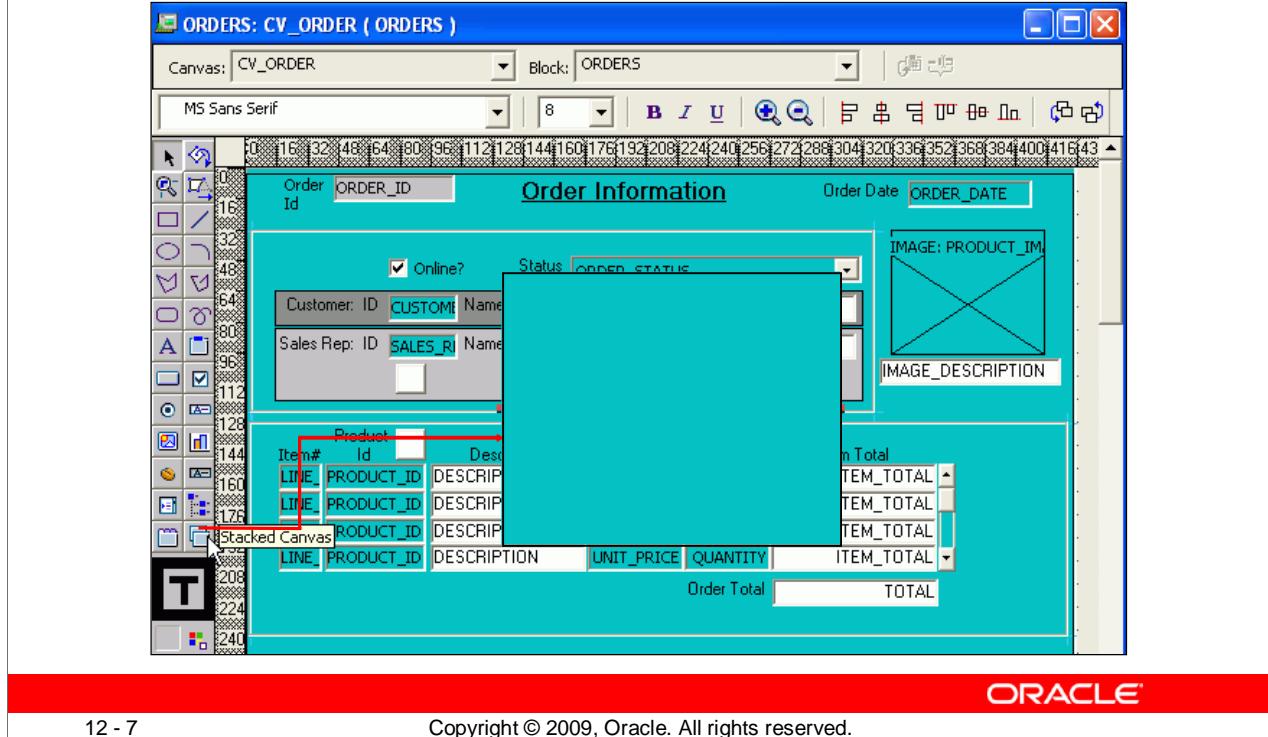


Stacked Canvas Elements

The stacked canvas is displayed on the content canvas. You can see the following elements in the screenshot in the slide:

1. Content canvas
2. Stacked canvas
3. Viewport X/Y position of the stacked canvas
4. Viewport height of the stacked canvas
5. Viewport width of the stacked canvas

Creating a Stacked Canvas



Creating a Stacked Canvas

How to Create a Stacked Canvas in the Layout Editor

It is usually easier to use the Layout Editor to create a stacked canvas because you can visually set its properties. To create a stacked canvas in the Layout Editor, perform the following steps:

1. In the Object Navigator, double-click the object icon for the content canvas on which you want to create a stacked canvas.
The Layout Editor is displayed.
 2. Click the Stacked Canvas tool in the toolbar.
 3. Click and drag in the canvas where you want to position the stacked canvas. The slide graphic pictures the Layout Editor immediately after this step has been done. At this point, the stacked canvas appears as a blank window on top of and obscuring part of the content canvas.
 4. Open the Property Palette of the stacked canvas. Set the canvas properties according to your requirements (described later in this lesson).

Creating a Stacked Canvas (continued)

How to Create a Stacked Canvas in the Object Navigator

To create a stacked canvas in the Object Navigator, perform the following steps:

1. Select the Canvases node in the Object Navigator.
2. Click the Create icon.
A new canvas entry displays with a default name of CANVASXX.
3. If the Property Palette is not already displayed, select the new canvas entry and then select Tools > Property Palette.
4. Set the Canvas Type property to Stacked. Additionally, set the properties that are described later in this lesson according to your requirements.
5. Ensure that the stacked canvas is below the content canvas in the Object Navigator.

Note: To convert an existing content canvas to a stacked canvas, change its Canvas Type property value from Content to Stacked.

For the stacked canvas to display properly at run time, ensure that its position in the stacking order places it in front of the content canvas assigned to the same window. The stacking order of canvases is defined by the sequence in which they appear in the Object Navigator.

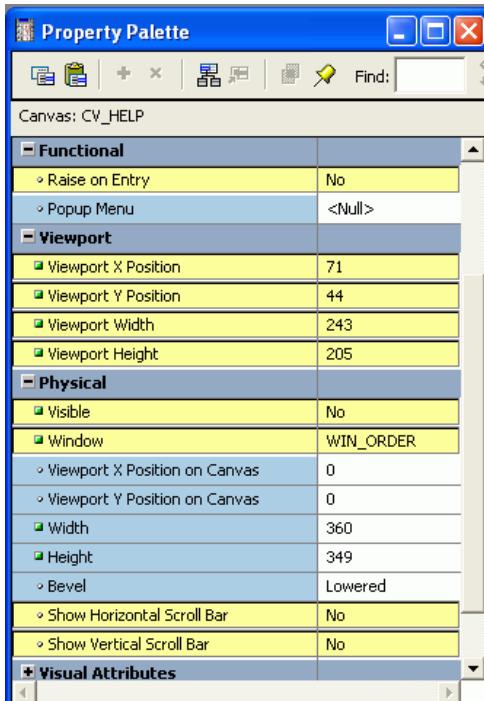
Displaying Stacked Canvases in the Layout Editor

You can display a stacked canvas as it sits over the content canvas in the Layout Editor. You can check the display position of stacked canvases by doing the following:

- Select View > Stacked Views in the Layout Editor. The Stacked/Tab Canvases dialog box is displayed, with a list of all the stacked canvases assigned to the same window as the current content canvas.
- Select the stacked canvases you want to display in the Layout Editor.

Note: Press Ctrl and click to clear a stacked canvas that was previously selected.

Setting Stacked Canvas Properties



ORACLE®

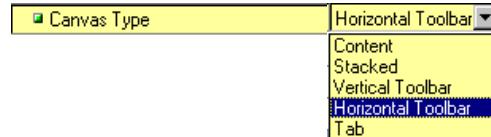
Setting Stacked Canvas Properties

There are several properties that you can set to affect the appearance and behavior of stacked canvases. The screenshot shows the Property Palette for a stacked canvas:

- **"Raise on Entry":** Controls how Forms displays the canvas when the user or the application navigates to an item on the canvas
- **Viewport X/Y Position:** Specifies the point at which the upper-left corner of the stacked canvas is located in relation to the content canvas
- **Viewport Width/Height:** Determines the size of the stacked canvas displayed at run time
- **Visible:** Indicates whether the stacked canvas is initially displayed or visible; set to Yes or No to show or hide the stacked canvas
- **Window:** Specifies the window in which the canvas will be displayed
- **Show Horizontal/Vertical Scroll Bar:** Specifies whether scrollbars display with the canvas

What Is a Toolbar Canvas?

- Special type of canvas for tool items
- Two types:
 - Vertical toolbar
 - Horizontal toolbar
- Provide:
 - Standard look and feel
 - Alternative to menu or function key operation



ORACLE®

12 - 10

Copyright © 2009, Oracle. All rights reserved.

What Is a Toolbar Canvas?

A *toolbar canvas* is a special type of canvas that you can create to hold buttons and other frequently used GUI elements.

Toolbar Types

- **Vertical toolbar:** Use a vertical toolbar to position all your tool items to the left of your window.
- **Horizontal toolbar:** Use a horizontal toolbar to position all your tool items and controls across the top of your window under the window's menu bar.

The screenshot in the slide shows that each of these toolbar types is available in the drop-down list for canvas types in the Property Palette for a canvas.

Uses and Benefits of Toolbars

Toolbar canvases offer the following advantages:

- Provide a standard look and feel across canvases displayed in the same window.
- Decrease form module maintenance time.
- Increase application usability.
- Create applications similar to others used in the same environment.
- Provide an alternative to menu or function key-driven applications.

Creating a Toolbar Canvas

1. Create new or modify the existing canvas.
 - a. Click Create in the Object Navigator.
 - b. Change Canvas Type.
 - c. Set other properties as needed.
2. Add functionality.
3. Resize the canvas (not the view).
4. Assign to window and/or form.

ORACLE®

12 - 11

Copyright © 2009, Oracle. All rights reserved.

Creating a Toolbar Canvas

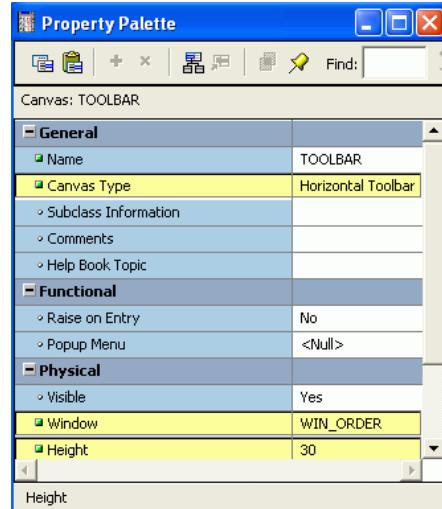
To create a Toolbar canvas, perform the following steps:

1. Create new or modify the existing canvas:
 - With the Canvases node selected, click the Create icon in the Object Navigator.
 - If the Property Palette is not already displayed, click the new canvas entry and select Tools > Property Palette.
 - Set the Canvas Type to either Horizontal or Vertical Toolbar.
2. Add functionality, usually with push buttons. To avoid problems with unwanted navigation, be sure to set the Mouse Navigate property of any toolbar buttons to No.
3. Resize: Use the Layout Editor to resize the toolbar canvas (not just the view) to the smallest size necessary to display its items.
4. Assign to window or form: In the Property Palette of the window or the form module, set the Horizontal or Vertical Toolbar Canvas properties.

When toolbars are placed on the form or window, the window manager controls shifting the content canvas downward or to the right in order to display the toolbar. You do not need to leave enough space in the content canvas for the placement of the toolbar.

Setting Toolbar Properties

- Canvas properties:
 - Canvas Type
 - Window
 - Width or Height
- Window properties:
 - Horizontal Toolbar Canvas
 - Vertical Toolbar Canvas
- Form module properties:
 - Form Horizontal Toolbar Canvas
 - Form Vertical Toolbar Canvas



Setting Toolbar Properties

After you create a toolbar canvas, you can set its properties as well as the properties of the associated window or form module:

- **Canvas Properties**
 - **Canvas Type:** Can be either Horizontal or Vertical Toolbar
 - **Window:** Specifies in which window the toolbar displays
 - **Width/Height:** Specifies the size of the canvas. The width of a horizontal toolbar is set to the width of the window (for example, content canvas). Likewise, the height of a vertical toolbar is set to the height of the window.
- **Window Property**
 - Horizontal/Vertical Toolbar Canvas: Identifies the horizontal/vertical toolbar to be displayed in this window
- **Form Module Property**
 - Form Horizontal/Vertical Toolbar Canvas: Identifies the horizontal/vertical toolbar to be displayed in the Multiple Document Interface (MDI) window (when the useSDI run-time parameter is set to No)

Using an MDI Toolbar

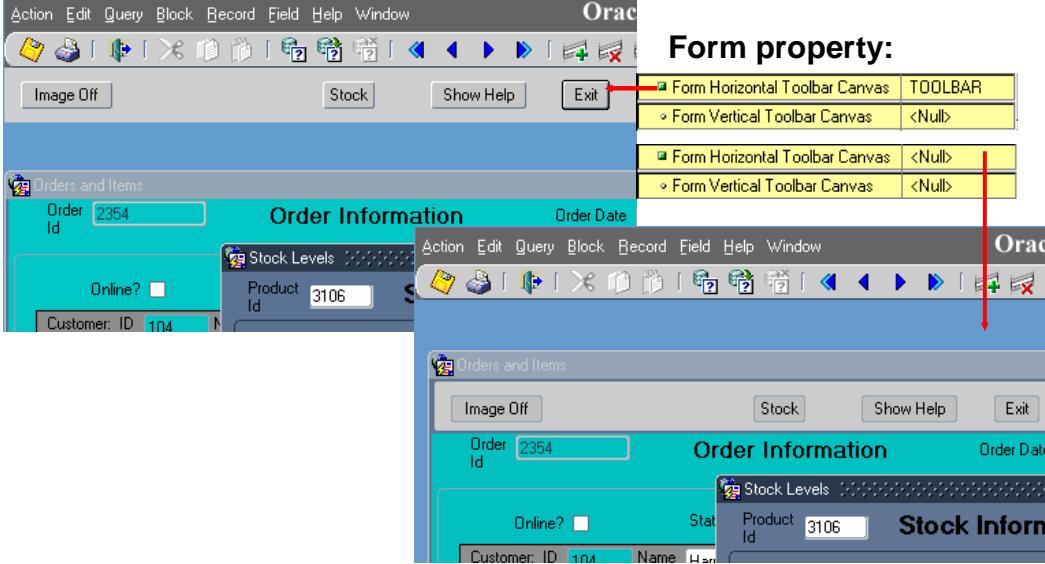
Run-time parameter:
`otherparams=useSDI=no`

Window property:

Horizontal Toolbar Canvas	TOOLBAR
Vertical Toolbar Canvas	<Null>

Form property:

Form Horizontal Toolbar Canvas	TOOLBAR
Form Vertical Toolbar Canvas	<Null>
Form Horizontal Toolbar Canvas	<Null>
Form Vertical Toolbar Canvas	<Null>



The screenshot shows two Oracle Forms windows. The top window is titled 'Orders and Items' and contains a sub-form titled 'Order Information'. The bottom window is also titled 'Orders and Items' and contains a sub-form titled 'Stock Information'. Both windows have standard menus like Action, Edit, Query, etc., and toolbar icons. Red arrows point from the 'Form property' table entries to the respective toolbars in each window. The Oracle logo is visible at the bottom right.

12 - 13 Copyright © 2009, Oracle. All rights reserved.

Using an MDI Toolbar

You can attach the toolbar to individual windows, or to the form itself. Attaching a toolbar to a form provides an MDI toolbar, so that you do not need to create more than one toolbar for a Forms application that uses multiple windows. If you display a toolbar in the MDI window, the same toolbar will not be duplicated in the individual windows of the form.

Whether an MDI toolbar is displayed is determined by a combination of Form properties and the `useSDI` run-time parameter, which you set as part of `otherparams`.

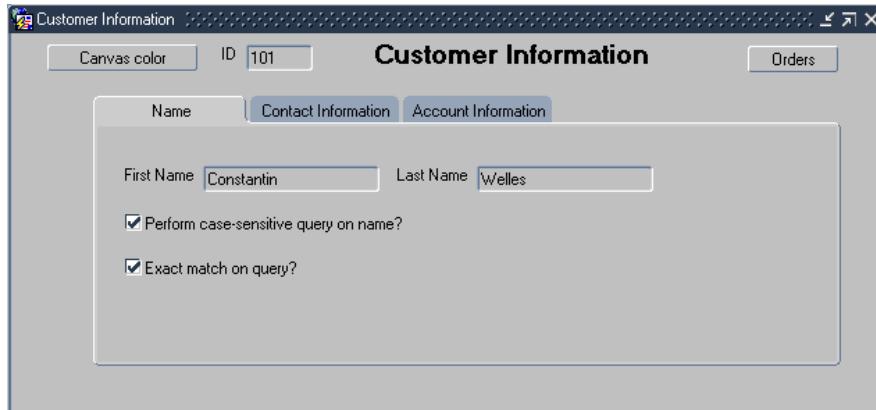
Setting	Setting Type	Effect
Form Horizontal or Vertical Toolbar Canvas	Form Property	If <code>useSDI=no</code> , displays MDI toolbar at the top or left of the MDI window
<code>otherparams=useSDI=no</code>	Run-time setting	Displays the MDI window and uses the MDI toolbar if set as a Form property
<code>otherparams=useSDI=yes</code>	Run-time setting	MDI window not displayed; Form Horizontal or Vertical Toolbar setting, if any, has no effect

Using an MDI Toolbar (continued)

The screenshots illustrate how toolbar canvases appear when `useSDI` is set to no:

- The screenshot at the left shows using an MDI toolbar by setting the Form Horizontal Toolbar Canvas to the name of the toolbar canvas. The toolbar canvas appears at the top of the form MDI window, but not on individual windows.
- The screenshot at the bottom shows that when an MDI toolbar is not defined, the toolbar appears on each individual window, rather than on the MDI window.

What Is a Tab Canvas?



- Enables you to organize and display related information on separate tabs
- Consists of one or more tab pages
- Provides easy access to data

ORACLE®

12 - 15

Copyright © 2009, Oracle. All rights reserved.

What Is a Tab Canvas?

A *tab canvas* is a special type of canvas that enables you to organize and display related information on separate tabs. Like stacked canvases, tab canvases are displayed on top of a content canvas. The screenshot in the slide shows a content canvas for Customer Information that has a tabbed canvas displayed over it. Each tab page is labeled with the type of information it contains (name, contact, and account information), but the customer ID is on the content canvas.

What Is a Tab Page?

A *tab page* is a subobject of a tab canvas. Each tab canvas is made up of one or more tab pages. A tab page displays a subset of the information in the entire tab canvas. Each tab page has a labeled tab that end users can click to access information on the page.

Each tab page occupies an equal amount of space on the tab canvas.

Uses and Benefits of Tab Canvases

You can use tab canvases to:

- Create an overlay effect within a single window
- Display large amounts of information on a single canvas
- Hide information and easily access it by clicking the tab

Creating a Tab Canvas

1. Create in:
 - Object Navigator
 - Layout Editor (usually easier)
2. Define tab pages.
3. Place items on tab pages.

ORACLE®

12 - 16

Copyright © 2009, Oracle. All rights reserved.

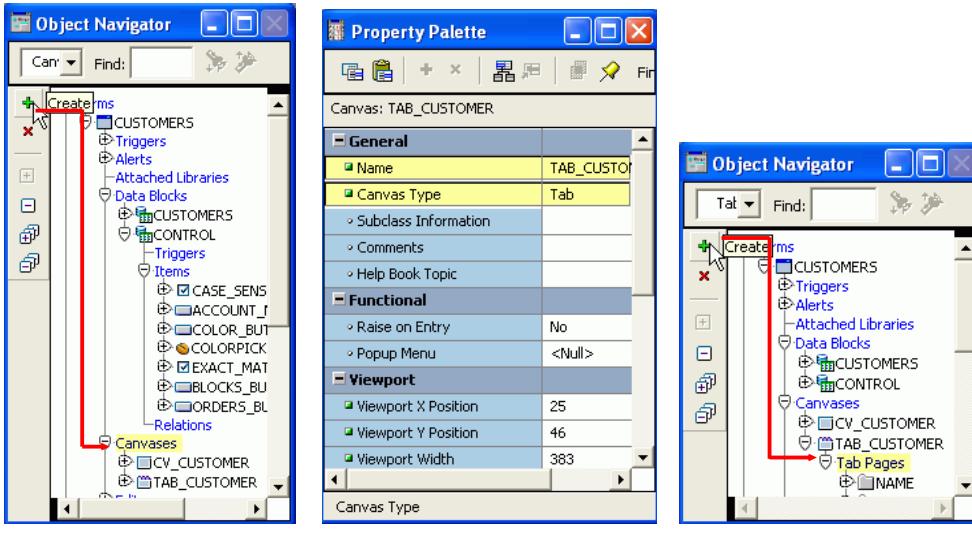
Creating a Tab Canvas

To create a functional tab canvas, you must:

1. Create an empty tab canvas in either of the following:
 - Object Navigator
 - Layout Editor; it may be easier to use the Layout Editor to create a tab canvas because you can visually set its properties
2. Define one or more tab pages for the tab canvas.
3. Place items on the tab pages.

Note: Although it is possible to get rid of a content canvas after creating a tab canvas, it is still highly recommended that there be at least one content canvas for each window in your Forms application.

Creating a Tab Canvas in the Object Navigator

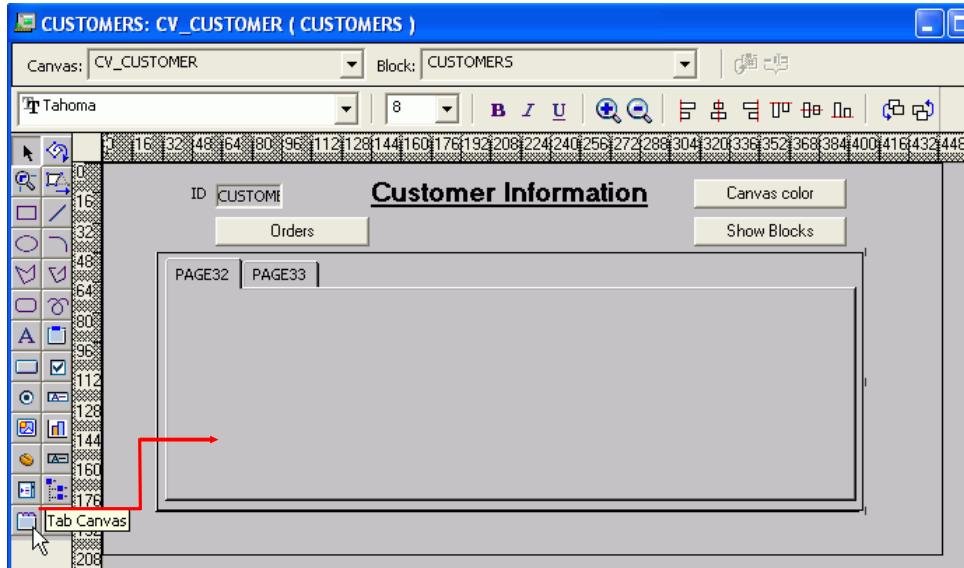


Creating a Tab Canvas in the Object Navigator

To create a tab canvas in the Object Navigator, perform the following steps:

1. Select the Canvases node in the Object Navigator.
2. Click the Create icon. A new canvas entry is displayed.
3. If the Property Palette is not already displayed, select the new canvas entry and select Tools > Property Palette.
4. Set the Canvas Type property to Tab. Additionally, set the canvas properties according to your requirements (described later in the lesson).
5. Expand the canvas node in the Object Navigator.
The Tab Pages node is displayed.
6. Click the Create icon, as shown in the screenshot at the right of the slide.
A tab page displays in the Object Navigator, with a default name of PAGEXX. The Property Palette takes on its context.
7. Set the tab page properties according to your requirements (described later in the lesson).
8. Create additional tab pages by repeating steps 6 and 7.

Creating a Tab Canvas in the Layout Editor



ORACLE®

12 - 18

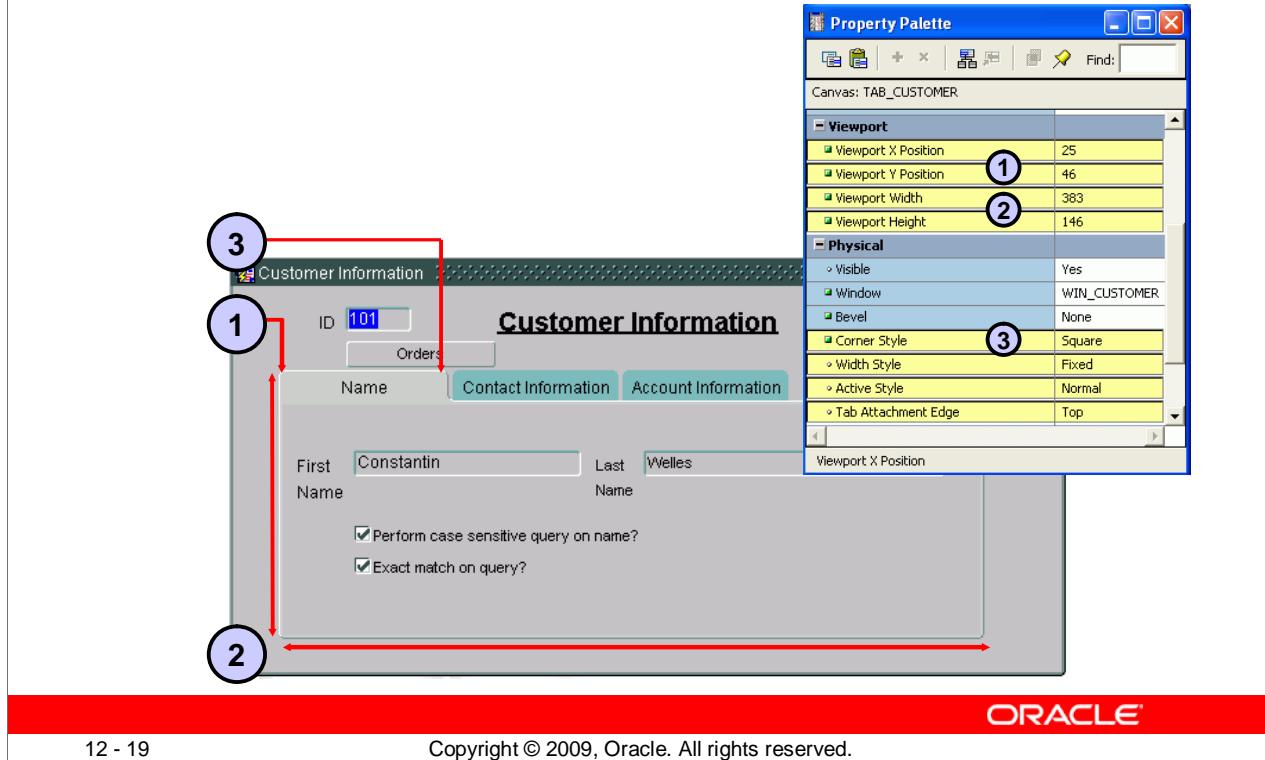
Copyright © 2009, Oracle. All rights reserved.

Creating a Tab Canvas in the Layout Editor

To create a tab canvas in the Layout Editor, perform the following steps:

1. In the Object Navigator, double-click the object icon for the content canvas on which you want to create a tab canvas.
The Layout Editor is displayed.
2. Click the Tab Canvas tool in the toolbar.
3. Click and drag in the canvas where you want to position the tab canvas.
Forms Builder creates a tab canvas with two tab pages by default. The screenshot in the slide shows the Layout Editor immediately after this step has been performed; it shows the tab canvas on top of the content canvas, with two default tab pages.
4. Open the Property Palette of the tab canvas. Set the canvas properties according to your requirements, as described on the next page of this lesson.
5. Create additional tab pages, if needed, in the Object Navigator.
6. Set the tab page properties according to your requirements, as described on the next page of this lesson.

Setting Tab Canvas, Tab Page, and Item Properties



Setting Tab Canvas, Tab Page, and Item Properties

After you create a tab canvas and its tab pages, you must set some properties for both of these objects. Place items on a tab page by setting the necessary item properties.

- **Tab Canvas Properties** (the numbered properties are illustrated in both the Property Palette and the run-time form):
 1. **Viewport X/Y Position:** Specifies the point at which the upper-left corner of the tab page is located in relation to the content canvas
 2. **Viewport Width/Height:** Defines the size of the tab canvas
 3. **Corner Style:** Specifies the shape of labeled tabs (Chamfered, Square, or Rounded); the screenshot in the slide shows a Square corner style.
 - **Tab Attachment Edge:** Specifies where tabs are attached (Top, Bottom, Left, Right, Start, or End)
 - **Width Style:** Specifies whether the width of the tab will vary with the length of the label
 - **Active Style:** Specifies whether the label displays as bold when the tab page is active
- **Tab Page Property:** Specifies the label to be displayed on the tab page's tab
- **Item Properties:**
 - **Canvas:** The tab canvas on which the item will be displayed
 - **Tab Page:** The tab page on which the item will be displayed

Placing Items on a Tab Canvas

- Place items on each tab page for user interaction.
- Set the item properties:
 - Canvas
 - Tab Page

ORACLE®

12 - 20

Copyright © 2009, Oracle. All rights reserved.

Placing Items on a Tab Canvas

After you create a tab canvas and related tab pages, you place items on the tab pages that the end users can interact with at run time:

- Open the Property Palette of the item.
- Set the item's Canvas and Tab Page properties to the desired tab canvas and tab page.

Note: Display the tab canvas as it sits on top of the content canvas, by selecting View > Stacked View in the Layout Editor.

Performance Tip: The time taken to load and initialize a tab canvas at run time is related to all objects on the canvas and not just to those initially visible. To improve this loading and initialization time, you can use a tab canvas to get the tabbed effect and place stacked canvases on the tab pages, adding the items and other components on the stacked canvases. By doing this, the components are not rendered until they are needed.

Summary

In this lesson, you should have learned that:

- You can use canvas types other than content:
 - Stacked
 - Toolbar
 - Tab
- You can create these in the Object Navigator and change the canvas type, and then set properties
- You can create stacked or tab canvases with the appropriate tool in the Layout Editor
- You can attach a toolbar canvas to a single window, or to the entire form if using MDI
- After creating a tab canvas, you create tab pages and place related items on them

ORACLE®

12 - 21

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you should have learned to:

- Use a stacked canvas to:
 - Create an overlay effect
 - Create a cascading or revealing effect within a single window
 - Display addition information
 - Conditionally display or hide information
 - Provide context-sensitive help
- Create a toolbar:
 - Display on top or to the left of a content canvas
 - Provide buttons or other frequently used GUI elements
 - Enforce a standard look and feel across canvases displayed in the same form or window
- Create a tabbed canvas to organize and display related information on multiple tab pages

Practice 12: Overview

This practice covers the following topics:

- Creating a toolbar canvas
- Creating a stacked canvas
- Creating a tab canvas
- Adding tab pages to the tab canvas



Practice 12: Overview

In this practice session, you will create different types of canvases: stacked canvas, toolbar canvas, and tab canvas.

- Create a horizontal toolbar canvas in the Orders form. Create new buttons in the CONTROL block, and place them on the horizontal toolbar. Save and run the form.
- Create a stacked canvas in the Orders form to add some help text. Position the canvas in the center of the window. Create a button in the CONTROL block. This button will be used later to display the stacked canvas. Add help text on the stacked canvas. Save and run the form.
- Create a tab canvas in the Customers form. Create three tab pages on this canvas, and make sure that each tab page displays the appropriate information. Save and run the form.

Oracle Fusion Middleware 11g: Build Applications with Oracle Forms

Volume II • Student Guide

D61530GC10

Edition 1.0

December 2009

DXXXX

ORACLE®

Author	Copyright © 2009, Oracle. All rights reserved.
Pam Gamer	Disclaimer
Technical Contributors and Reviewers	This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.
Glenn Maslen Duncan Mills Grant Ronald	The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.
Editors	Restricted Rights Notice
Raj Kumar Amitha Narayan	If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:
Publishers	U.S. GOVERNMENT RIGHTS The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.
Jayanthy Keshavamurthy Veena Narasimhan Giri Venugopal	Trademark Notice Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

13

Introduction to Triggers

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Define triggers
- Identify the different trigger categories
- Plan the type and scope of triggers in a form
- Describe the properties that affect the behavior of a trigger



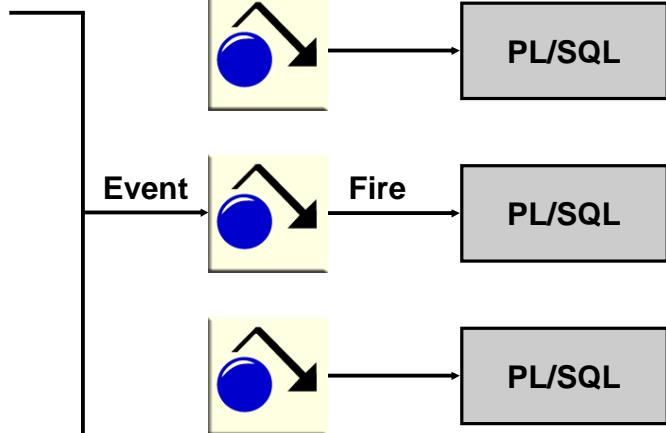
Lesson Aim

Triggers are one of the most important mechanisms that you can use to modify or add to the functionality of a form. In this lesson, you learn the essential rules and properties of triggers so that you can use them throughout your application.

Trigger: Overview

Trigger types:

Queries
Validation
Navigation
Interaction
Internal event
Errors/Messages
Others



Which trigger type would you use to perform complex calculations after a user enters data into an item?

ORACLE®

13 - 3

Copyright © 2009, Oracle. All rights reserved.

Trigger: Overview

A trigger is a program unit that is executed (fired) due to an event. As explained earlier, Forms Builder enables you to build powerful facilities into applications without writing a single line of code. However, you can use triggers to add or modify form functionality in a procedural way. As a result, you can define the detailed processes of your application.

You write Forms Builder triggers in PL/SQL. Every trigger that you define is associated with a specific event. Forms Builder defines a vast range of events for which you can fire a trigger.

These events include the following:

- Query-related events
- Data entry and validation
- Logical or physical navigation
- Operator interaction with items in the form
- Internal events in the form
- Errors and messages

Events cause the activation or firing of certain trigger types. The next page shows the categories of triggers and when they fire. By understanding when triggers fire, you can code them appropriately. For example, to perform complex calculations after a user enters data into an item, you can use a When-Validate-Item trigger.

The graphics in the slide show trigger symbols depicting the fact that each of the various types of triggers fires in response to a certain type of event and then executes the PL/SQL code of the trigger.

Grouping Triggers into Categories

Triggers may be grouped into functional categories:

- Block-processing triggers
- Interface-event triggers
- Master-detail triggers
- Message-handling triggers
- Navigational triggers
- Query-time triggers
- Transactional triggers
- Validation triggers

Triggers may be grouped into categories based on name:

- When-Event triggers
- On-Event triggers
- Pre-Event triggers
- Post-Event triggers
- Key triggers



Grouping Triggers into Categories

It is sometimes useful to think of triggers with respect to their various functions. You can also categorize triggers by their names.

The tables on the next page list triggers broken down into categories.

Grouping Triggers into Categories (continued)

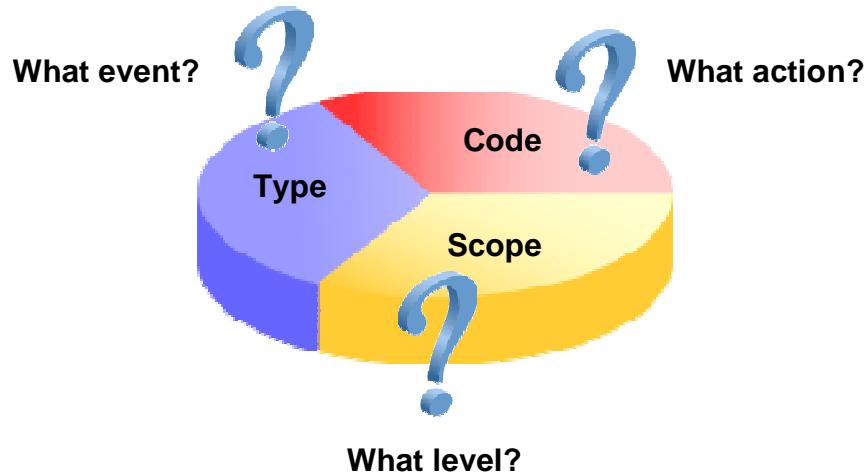
Triggers may be categorized based on their functions:

Category	Fires	Examples
Block processing	In response to events related to record management in a block	When-Create-Record When-Clear-Block When-Database-Record When-Remove-Record
Interface event	In response to events that occur in the form interface	Key-[all] When-Button-Pressed When-[Checkbox List Radio]-Changed When-Image-[Activated Pressed] When-Timer-Expired When-Window-[Activated Deactivated Closed Resized]
Master detail	To enforce coordination between records in a detail block and the master record in a master block	On-Check-Delete-Master On-Clear-Details On-Populate-Details
Message handling	In response to default messaging events	On-Error On-Message
Navigational	In response to navigational events	Pre-[Form Block Record Text-Item] Post-[Form Block Record Text-Item] When-New-[Form Block Record Item]-Instance
Query time	Just before and just after the operator or the application executes a query in a block	Pre-Query Post-Query
Transactional	During the commit process	On-Commit, On-Delete, On-Insert, On-Update
Validation	When Forms validates data after the user enters data and navigates out of the item or record	When-Validate-[Item Record]

You can also categorize triggers based on their names:

Category	Description
When-Event	Point at which Forms default processing may be <i>augmented</i> with additional tasks or operations
On-Event	Point at which Forms default processing may be <i>replaced</i>
Pre-Event	Point just prior to the occurrence of either a When-event or an On-event; use to prepare objects or data for the upcoming event
Post-Event	Point just following the occurrence of either a When-event or an On-event; use to validate or perform auditing tasks based on the prior event
Key Triggers	Fires when the operator presses a specific key or key sequence

Trigger Components



ORACLE®

13 - 6

Copyright © 2009, Oracle. All rights reserved.

Trigger Components

There are three main components to consider when you design a trigger in Forms Builder:

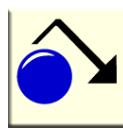
- **Trigger type:** Defines the specific event that causes the trigger to fire
- **Trigger code:** The body of PL/SQL that defines the actions of the trigger
- **Trigger scope:** The level in a form module at which the trigger is defined—determining the scope of events that is detected by the trigger

The graphic shows a trigger as a pie with three slices labeled Type, Code, and Scope. There is a question mark on each slice. The associated questions are:

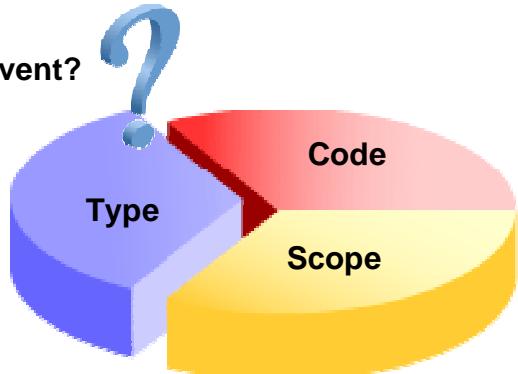
- **For Type:** What event?
- **For Code:** What action?
- **For Scope:** What level?

Trigger Type

- Pre-
- Post-
- When-
- On-
- Key-
- User-named



What event?



ORACLE®

13 - 7

Copyright © 2009, Oracle. All rights reserved.

Trigger Type

The trigger type determines which type of event fires it. There are more than 100 built-in triggers, each identified by a specific name. Triggers are mostly fired by events within a form module. Menu modules can initiate an event in a form, but the form module owns the trigger that fires.

The name of a trigger identifies its type. All built-in trigger types are associated with an event, and their names always contain a hyphen (-). For example:

- When-Validate-Item fires when Forms validates an item.
- Pre-Query fires before Forms issues a query for a block.

Note: Database events that occur on behalf of a form can fire certain triggers, but these database triggers are different from Forms Builder triggers.

Forms Builder supports user-named triggers as well as the standard built-in triggers. A user-named trigger is one that is named by the designer. These triggers fire only if called by another trigger or program unit using built-in code features.

Trigger Types

[User-named] KEY-CLRBLK KEY-CLRFRM KEY-CLRREC KEY-COMMIT KEY-CQUERY KEY-CREREC KEY-DELREC KEY-DOWN KEY-DUP-ITEM KEY-DUPREC KEY-EDIT KEY-ENTER KEY-ENTQRY KEY-EXEQRY KEY-EXIT KEY-F0 KEY-F1 KEY-F2 KEY-F3 KEY-F4 KEY-F5 KEY-F6 KEY-F7 KEY-F8 KEY-F9 KEY-HELP KEY-LISTVAL KEY-MENU KEY-NEXT-ITEM KEY-NXTBLK	KEY-NXTKEY KEY-NXTREC KEY-NXTSET KEY-OTHERS KEY-PREV-ITEM KEY-PRINT KEY-PRVBLK KEY-PRVREC KEY-SCRDOWN KEY-SCRUP KEY-UP KEY-UPDREC ON-CHECK-DELETE-MASTER ON-CHECK-UNIQUE ON-CLOSE ON-COLUMN-SECURITY ON-COMMIT ON-COUNT ON-DELETE ON-FETCH ON-INSERT ON-LOCK ON-LOGON ON-LOGOUT ON-MESSAGE ON-POPULATE-DETAILS ON-ROLLBACK ON-SAVEPOINT ON-SELECT ON-SEQUENCE-NUMBER	ON-UPDATE POST-BLOCK POST-CHANGE POST-DATABASE-COMMIT POST-DELETE POST-FORM POST-FORMS-COMMIT POST-INSERT POST-LOGON POST-LOGOUT POST-QUERY POST-RECORD POST-SELECT POST-TEXT-ITEM POST-UPDATE PRE-BLOCK PRE-COMMIT PRE-DELETE PRE-INSERT PRE-LOGON PRE-LOGOUT PRE-POPUP-MENU PRE-QUERY PRE-RECORD PRE-SELECT PRE-TEXT-ITEM PRE-UPDATE WHEN-BUTTON-PRESSED WHEN-CHECKBOX-CHANGED WHEN-CLEAR-BLOCK	WHEN-CREATE-RECORD WHEN-CUSTOM-ITEM-EVENT WHEN-DATABASE-RECORD WHEN-FORM-NAVIGATE WHEN-IMAGE-ACTIVATED WHEN-IMAGE-PRESSED WHEN-LIST-ACTIVATED WHEN-LIST-CHANGED WHEN-.MOUSE-CLICK WHEN-.MOUSE-DOUBLECLICK WHEN-.MOUSE-DOWN WHEN-.MOUSE-ENTER WHEN-.MOUSE-LEAVE WHEN-.MOUSE-MOVE WHEN-.MOUSE-UP WHEN-NEW-BLOCK-INSTANCE WHEN-NEW-ITEM-INSTANCE WHEN-NEW-RECORD-INSTANCE WHEN-RADIO-CHANGED WHEN-REMOVE-RECORD WHEN-TAB-PAGE-CHANGED WHEN-TIMER-EXPIRED WHEN-TREE-NODE-ACTIVATED WHEN-TREE-NODE-EXPANDED WHEN-TREE-NODE-SELECTED WHEN-VALIDATE-ITEM WHEN-VALIDATE-RECORD WHEN-WINDOW-ACTIVATED WHEN-WINDOW-CLOSED WHEN-WINDOW-DEACTIVATED WHEN-WINDOW-RESIZED
---	---	--	---

Forms Builder Trigger Types

ORACLE®

13 - 8

Copyright © 2009, Oracle. All rights reserved.

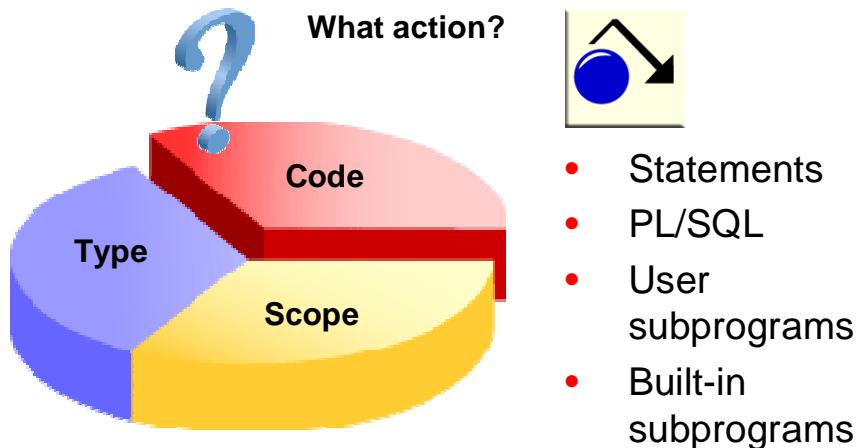
Trigger Type (continued)

The first part of a trigger name (before the first hyphen) follows a standard convention; this helps you understand the general nature of the trigger type, and plan the types to use.

Trigger Prefix	Description
Key-	Fires in place of the standard action of a function key
On-	Fires in place of standard processing (used to replace or bypass a process)
Pre-	Fires just before the action named in the trigger type (for example, before a query is executed)
Post-	Fires just after the action named in the trigger type (for example, after a query is executed)
When-	Fires in addition to standard processing (used to augment functionality)

The screenshot shows the triggers that are available in Forms Builder, sorted by trigger type.

Trigger Code



ORACLE®

13 - 9

Copyright © 2009, Oracle. All rights reserved.

Trigger Code

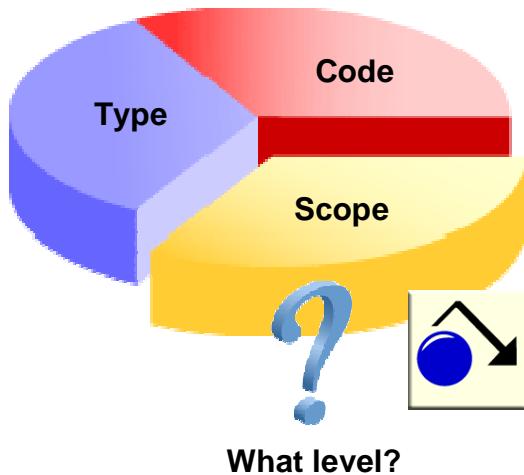
The code of the trigger, shown by the exploded Code pie slice in the graphic, defines the actions for the trigger to perform when it is fired. Write this code as an anonymous PL/SQL block. Enter the BEGIN...END structure in your trigger text only if starting your block with a DECLARE statement or if coding subblocks. Statements that you write in a trigger answer the “What Action?” question in the slide graphic, and can include:

- SQL statements that are legal in a PL/SQL block; these are passed to the server
- Standard PL/SQL constructs (assignments, control statements, and so on)
- Calls to user-named subprograms (procedures and functions) in the form, a library, or the database
- Calls to built-in subprograms and package subprograms

Although you can include SQL statements in a trigger, keep in mind the following rules:

- INSERT, UPDATE, and DELETE statements can be placed in transactional triggers. These triggers fire during the commit process.
- Transaction control statements (COMMIT, ROLLBACK, and SAVEPOINT) should not be included directly as SQL trigger statements. These actions are carried out automatically by Forms as a result of either commands or built-in procedures that you issue. If included in triggers, these commands are redirected to be handled by Forms. For example, COMMIT issues a COMMIT_FORM.

Trigger Scope



ORACLE®

13 - 10

Copyright © 2009, Oracle. All rights reserved.

Trigger Scope

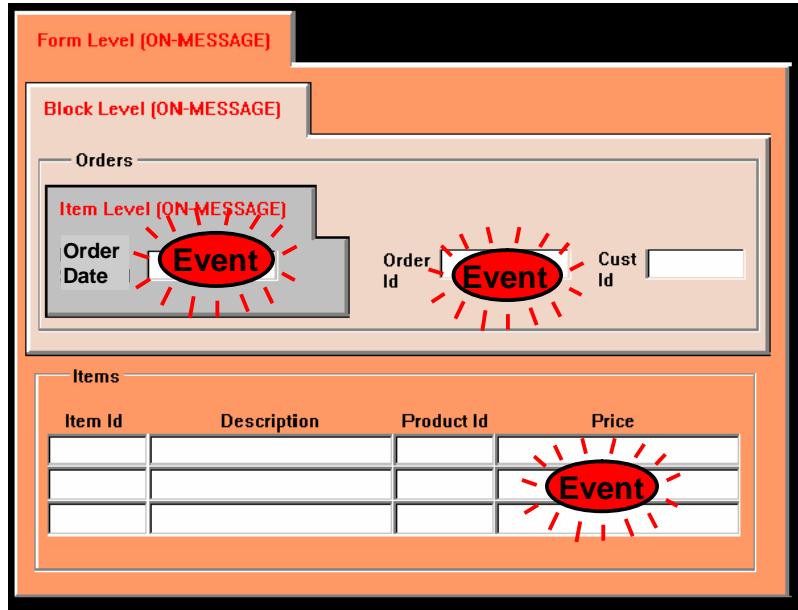
The scope of a trigger is determined by its position in the form object hierarchy—that is, the type of object under which you create the trigger. There are three possible levels that answer the “What Level?” question in the slide graphic:

- **Form level:** The trigger belongs to the form and can fire due to events across the entire form.
- **Block level:** The trigger belongs to a block and can fire only when this block is the current block.
- **Item level:** The trigger belongs to an individual item and can fire only when this item is the current item.

Some triggers cannot be defined below a certain level. For example, Post-Query triggers cannot be defined at item level because they fire due to a global or restricted query on a block.

By default, only the trigger that is most specific to the current location of the cursor fires.

Trigger Scope: Example



ORACLE®

13 - 11

Copyright © 2009, Oracle. All rights reserved.

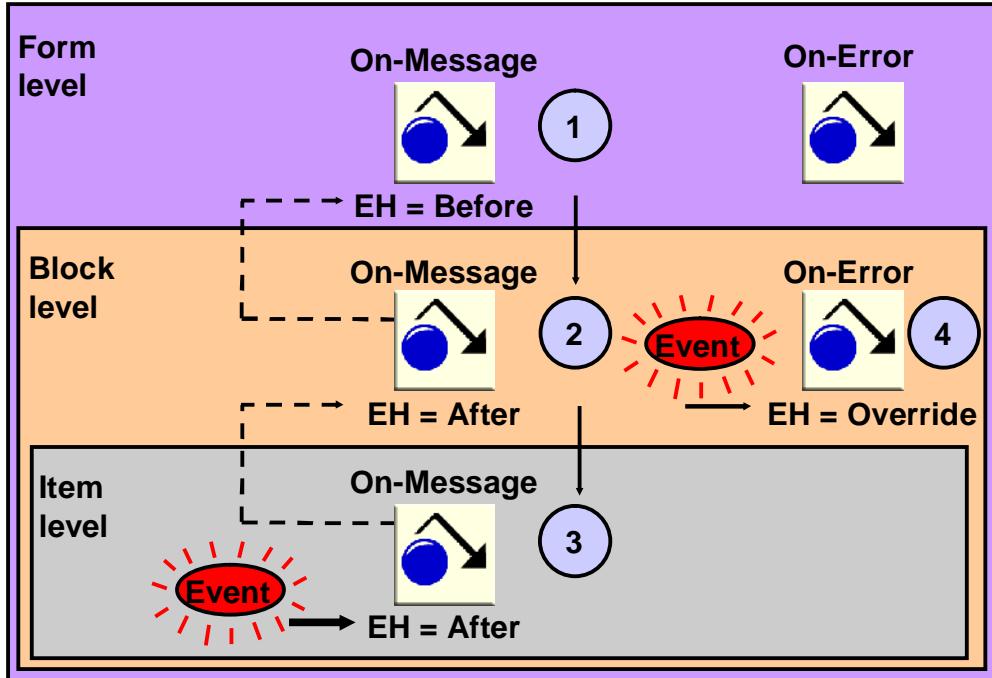
Trigger Scope: Example

Consider the example in the slide, which depicts a form with two blocks: ORDERS and ITEMS . There are three On-Message triggers in the form: one at form level, one at block level for the ORDERS block, and one at item level for the ORDERS.Order_Date item. Trigger scope has the following effects in this example:

- When the cursor is in the Order_Date item, a message fires the On-Message trigger of the Order_Date item because this is more specific than the other triggers of this type.
- When the cursor is elsewhere in the ORDERS block, a message causes the block-level On-Message trigger to fire because its scope is more specific than the form-level trigger. (You are outside the scope of the item-level trigger.)
- When the cursor is in the ITEMS block, a message causes the form-level On-Message trigger to fire because the cursor is outside the scope of the other two On-Message triggers.

Note: The On-Message trigger fires whenever Forms displays a message. It is recommended that you code On-Message triggers only at the form level.

Specifying Execution Hierarchy



ORACLE®

Specifying Execution Hierarchy

As already stated, when there is more than one trigger of the same type, Forms by default fires the trigger most specific to the cursor location. However, you can alter the firing sequence of a trigger by setting the Execution Hierarchy trigger property.

Execution Hierarchy (EH) is a trigger property that specifies how the current trigger code should execute if there is a trigger with the same name defined at a higher level in the object hierarchy. Setting EH for form-level triggers has no effect because there is no higher-level trigger.

Settings for Execution Hierarchy are the following:

- **Override:** Only the trigger that is most specific to the cursor location fires. This is the default.
- **After:** The trigger fires *after* firing the same trigger, if any, at the next highest level.
- **Before:** The trigger fires *before* firing the same trigger, if any, at the next highest level.

In the cases of Before and After, you can fire more than one trigger of the same type due to a single event. However, you must define each trigger at a different level.

Specifying Execution Hierarchy (continued)

The graphic in the slide shows how setting EH affects the firing order of triggers. The form in the example has four triggers:

1. An On-Message trigger at form level with Execution Hierarchy set to Before
2. An On-Message trigger at Block level with Execution Hierarchy set to After
3. An On-Message trigger at Item level with Execution Hierarchy set to After
4. An On-Error trigger at Block level with Execution Hierarchy set to Override

When an item-level event causes a message to display, the execution hierarchy causes the triggers to fire in the following order:

1. Fires first
2. Fires second
3. Fires third
4. Fires independently

Note: Broken lines indicate the analysis path before firing.

Summary

In this lesson, you should have learned that:

- Triggers are event-activated program units
- You can categorize triggers based on function or name to help you understand how they work
- Trigger components are:
 - Type: Defines the event that fires the trigger
 - Code: The actions a trigger performs
 - Scope: Specifies the level (form, block, or item) at which the trigger is defined
- The Execution Hierarchy trigger property alters the firing sequence of a trigger

ORACLE®

13 - 14

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you should have learned the essential rules and properties for triggers.

- Triggers are event-activated program units.
- You can categorize triggers based on:
 - **Function:** Block processing, interface event, master detail, message handling, navigational, query time, transactional, and validation
 - **Name:** When-, On-, Pre-, Post-, and Key-
- Trigger components include:
 - The trigger type that defines the event that fires the trigger
 - The trigger code that consists of a PL/SQL anonymous block
 - The trigger scope that determines which events are detected by the trigger; the three possible levels for a trigger are form, block, and item
- When an event occurs, the most specific trigger overrides the triggers at a more general level. This can be affected by the Execution Hierarchy trigger property.

14

Producing Triggers

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Write trigger code
- Explain the use of built-in subprograms in Forms applications
- Describe the When-Button-Pressed trigger
- Describe the When-Window-Closed trigger



Lesson Aim

This lesson shows you how to create triggers. You specifically learn how to use built-in subprograms in Oracle Forms applications, and how to program triggers that fire when a user clicks a button or closes a window.

Creating Triggers in Forms Builder

To create a trigger, perform the following steps:

1. Select a scope in the Object Navigator.
2. Create a trigger and select a name from the Trigger list of values (LOV), or use the SmartTriggers menu option.
3. Define code in the PL/SQL Editor.
4. Compile.



Creating Triggers in Forms Builder

Using Smart Triggers

When you right-click an object in the Object Navigator or Layout Editor, a pop-up menu is displayed that includes the selection Smart Triggers. The Smart Triggers item expands to an LOV of common triggers that are appropriate for the selected object. When you click one of these triggers, Forms Builder creates the trigger.

Using the Trigger LOV

Using Smart Triggers is the easiest way to create a new trigger, but you can also do it by displaying the Trigger LOV, a list of all Forms Builder triggers. You can invoke the Trigger LOV from the Object Navigator, the Layout Editor, the PL/SQL Editor, or the Smart Triggers LOV.

How to Create a Trigger

Use the steps on the following pages to create a trigger.

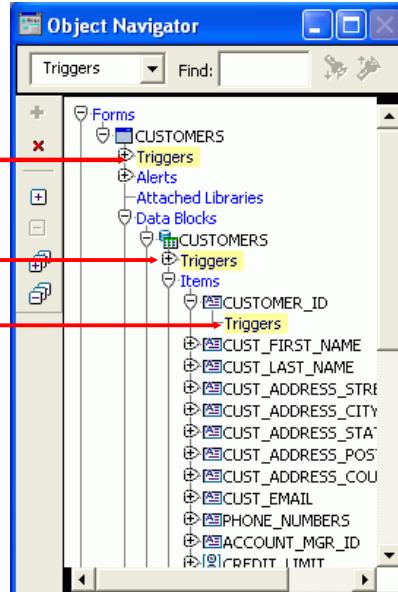
Creating a Trigger: Selecting the Trigger Scope

Step 1: Select the trigger scope.

Form level

Block level

Item level



ORACLE®

14 - 4

Copyright © 2009, Oracle. All rights reserved.

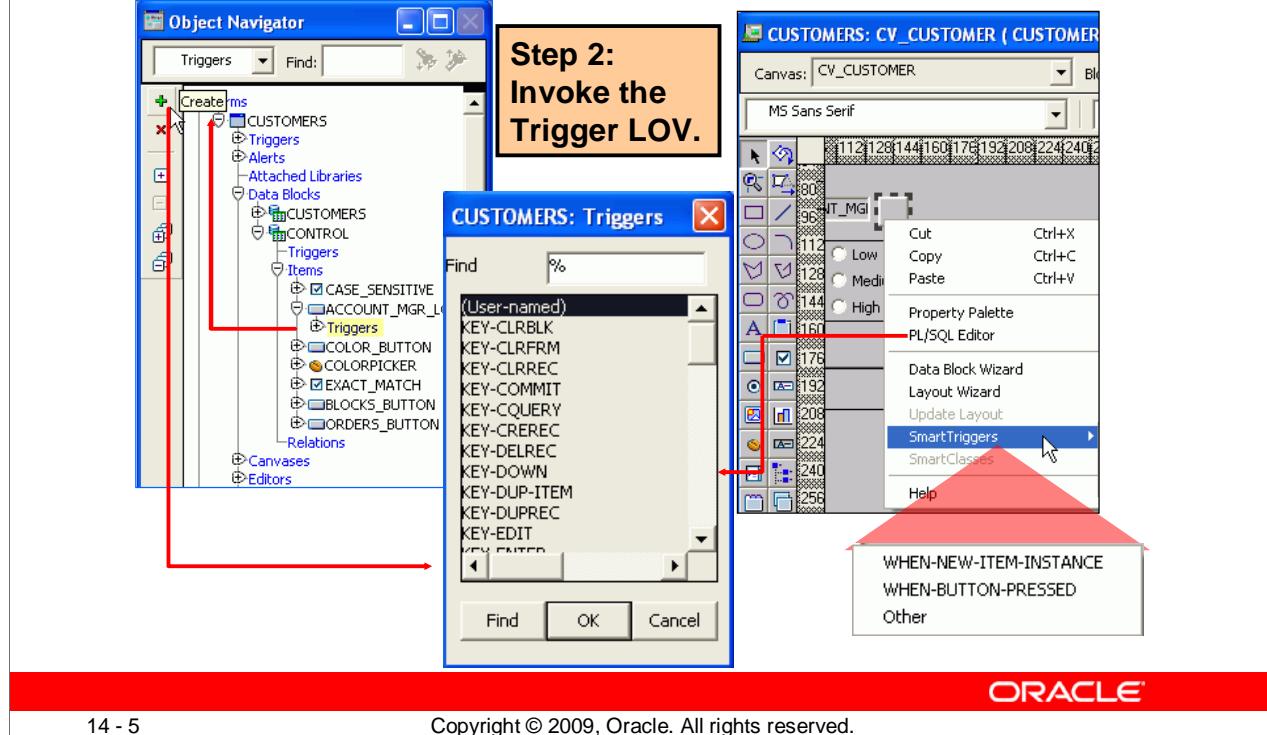
Creating a Trigger: Selecting Trigger Scope

In the Object Navigator, select the Triggers node of the form, block, or item that will own the trigger. The screenshot in the slide shows that selecting the Triggers node under the form name creates a form-level trigger, selecting the Triggers node under the block name creates a block-level trigger, and selecting the Triggers node under the item name creates an item-level trigger.

Alternatively, you can select an item in the Layout Editor if you are defining an item-level trigger.

A mistake often made by inexperienced Forms developers is to define a trigger at the wrong scope. For example, if you want a message to display when the cursor is placed on an item, you should code a When-New-Item-Instance trigger at the item level for that item. If you mistakenly put the trigger at the block or form level, the trigger will fire whenever the operator navigates to any item in the block or form.

Creating a Trigger: Selecting the Trigger Type



14 - 5

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Creating a Trigger: Selecting the Trigger Type

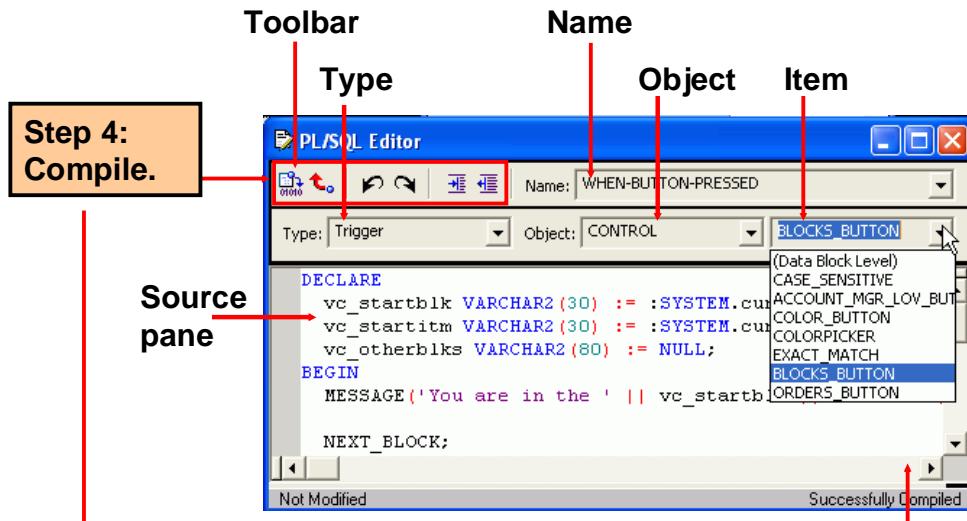
After the trigger scope is selected, there are several ways to invoke the Trigger LOV (as shown in the graphics in the slide):

- **Smart Triggers:** Right-click to display the pop-up menu. Select Smart Triggers. This invokes the Smart Triggers LOV displaying a list of common triggers for the selected object. If you do not see the desired trigger, select Other to display the Trigger LOV with a full list of triggers.
- **Create:** If you are in the Object Navigator with a Triggers node highlighted, select Edit > Create from the menu, or click Create in the toolbar. This invokes the Trigger LOV.
- **PL/SQL Editor:** Right-click to display the pop-up menu. Select PL/SQL Editor.
 - If there is no trigger defined for the object, this invokes the Trigger LOV.
 - If there are triggers already defined for the object, the name and code of the first one appear in the editor. To define an additional trigger for the item, click Create in the Object Navigator toolbar to invoke the Trigger LOV.

After the Trigger LOV is invoked, select the trigger type.

Creating a Trigger: Defining the Trigger Code

Step 3: Use the PL/SQL Editor to define the trigger code.



ORACLE®

14 - 6

Copyright © 2009, Oracle. All rights reserved.

Creating a Trigger: Defining the Trigger Code

Use the PL/SQL Editor to Define the Trigger Code

The trigger type and scope are now set in the PL/SQL Editor. You can enter the code for the trigger in the source pane of the editor.

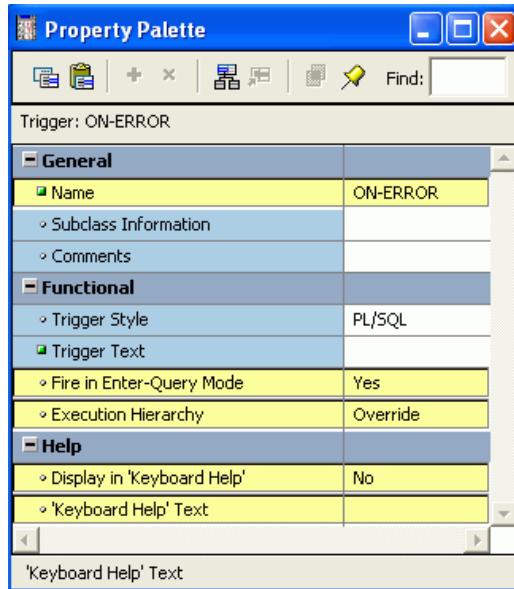
In Forms Builder, the PL/SQL Editor (shown in the slide) has the following specific trigger components:

- **Name:** Trigger name; pop-up list enables you to switch to a different trigger
- **Type:** Set to Trigger (The other types are Program Unit and Menu Item Code, but these are not valid for triggers.)
- **Object:** Enables you to set the scope to either Form Level or a specific block
- **Item:** Enables you to change between specific items (at item level) to access other triggers. The Item trigger component is not labeled.
- **Source pane:** Where trigger code is entered or modified
- **Toolbar:** Buttons to compile, revert, undo, redo, indent, or outdent the code

Compile

Click the Compile icon in the PL/SQL Editor to compile the trigger. This displays immediate feedback in the form of compilation error messages, which you can correct. If the trigger compiles correctly, a message displays in the lower-right corner of the editor.

Setting Trigger Properties



ORACLE®

14 - 7

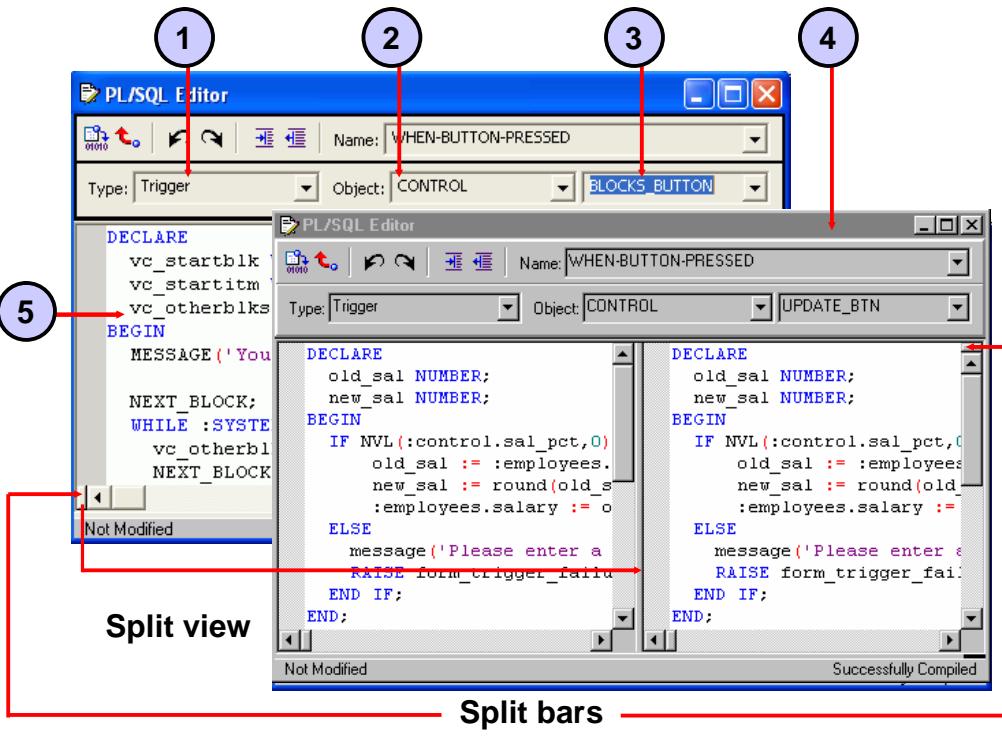
Copyright © 2009, Oracle. All rights reserved.

Setting Trigger Properties

In the Property Palette, you can set the following trigger properties to affect the behavior of triggers:

- **General**
 - **Name:** Specify the internal name of the trigger; this corresponds to the trigger type and you should not modify it except in the case of a user-defined trigger.
- **Functional**
 - **Fire in Enter-Query Mode:** Specify whether the trigger can fire when an event occurs in Enter Query as well as Normal mode.
 - **Execution Hierarchy:** Use to change the default order of trigger firing when multiple triggers of the same name are defined at different levels.
- **Help** (valid only for Key triggers)
 - **Display in 'Keyboard Help':** Specify whether you want the name or description to appear in the Show Keys window.
 - **'Keyboard Help' Text:** It is the optional description to replace the default key description.

PL/SQL Editor: Features



PL/SQL Editor: Features

The PL/SQL Editor provides the following features:

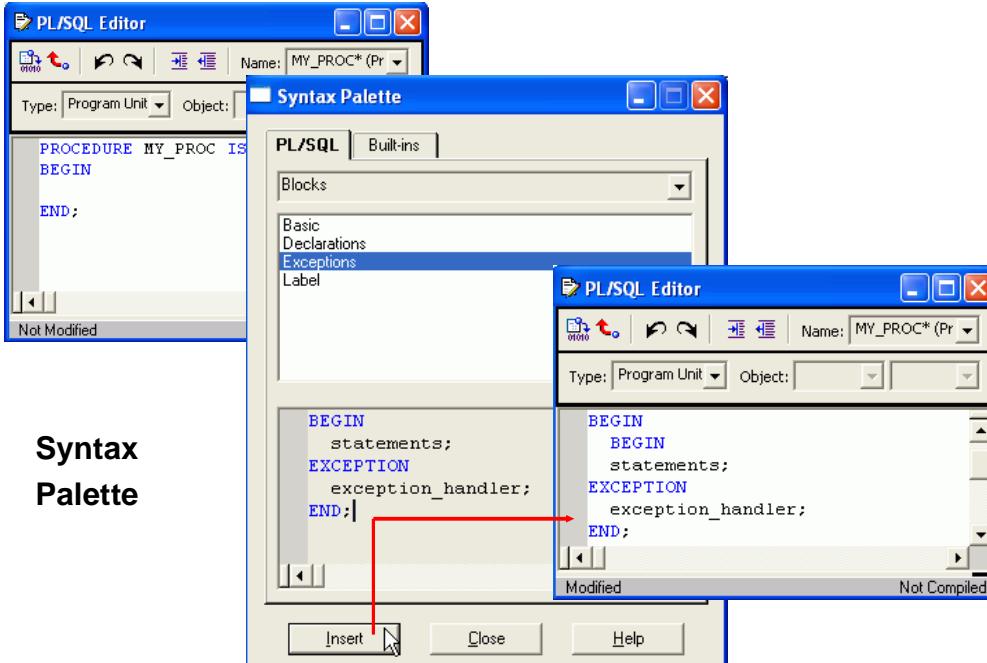
- Automatic formatting and coloring of PL/SQL code
- Automatic indenting and color syntax highlighting
- Drag-and-drop text manipulation
- Unlimited undo/redo
- Multiple split views

You can create up to four separate views of the current program unit in the PL/SQL Editor by using split bars. Place the cursor over the split bar; it changes to a double-headed arrow. Click and drag the split bar to the desired location. To remove the split, drag it back to its original location. The split bars are shown in the screenshots.

Trigger Components of the PL/SQL Editor

1. **Type:** Is set to Trigger
2. **Object:** Enables you to set scope to Form Level or a specified block
3. **Item:** Enables you to switch between items (if item-level trigger) to access other triggers
4. **Name:** Trigger name; enables you to switch to another existing trigger
5. **Source Pane:** Where the trigger code is entered or modified

PL/SQL Editor: Features



PL/SQL Editor: Features (continued)

Syntax Palette: Enables you to display and copy the constructs of PL/SQL language elements and built-in packages into an editor. To invoke the Syntax Palette, select Tools > Syntax Palette from the menu system. The Syntax Palette has an Insert button that enables you to insert the selected snippet into your code.

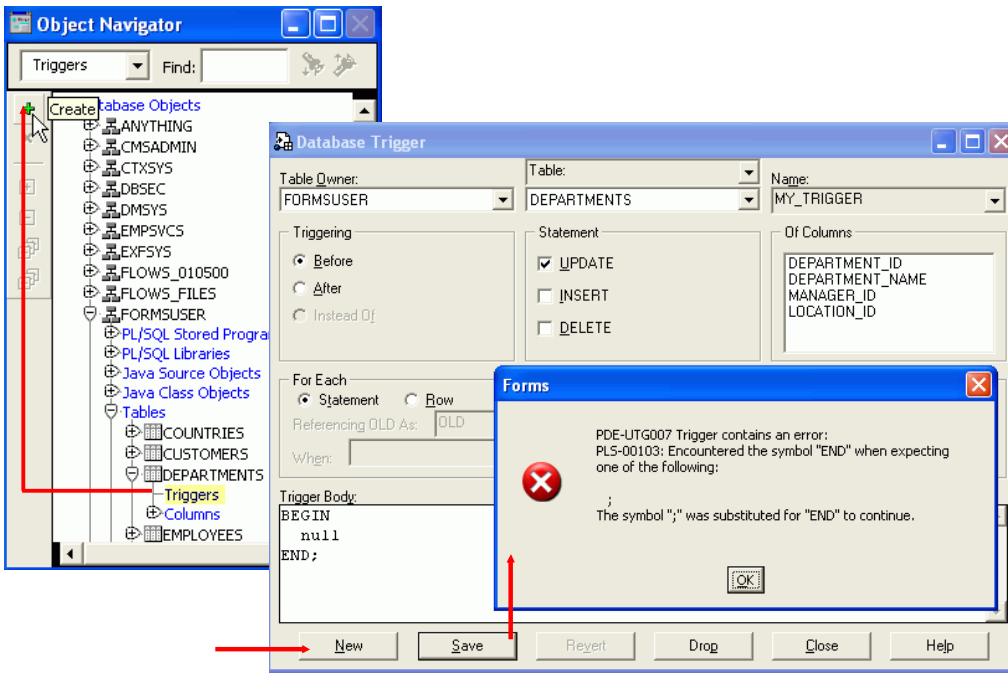
Global Search and Replace: Enables you to search for text across multiple program units without opening individual instances of the Program Unit Editor. You can replace every occurrence of the search string or only selected occurrences.

Invoke the “Find and Replace in Program Units” dialog box by selecting Edit > Find and Replace PL/SQL from the menu system.

Things to Remember About the PL/SQL Editor

- New or changed text in triggers remains uncompiled until you click Compile.
- Compiling triggers that contain SQL requires connection to the database.
- All uncompiled triggers are compiled when the form module is compiled.
- The Block and Item pop-up lists do not change the current trigger scope. They enable you to switch to another trigger.

Database Trigger Editor



14 - 10

Copyright © 2009, Oracle. All rights reserved.

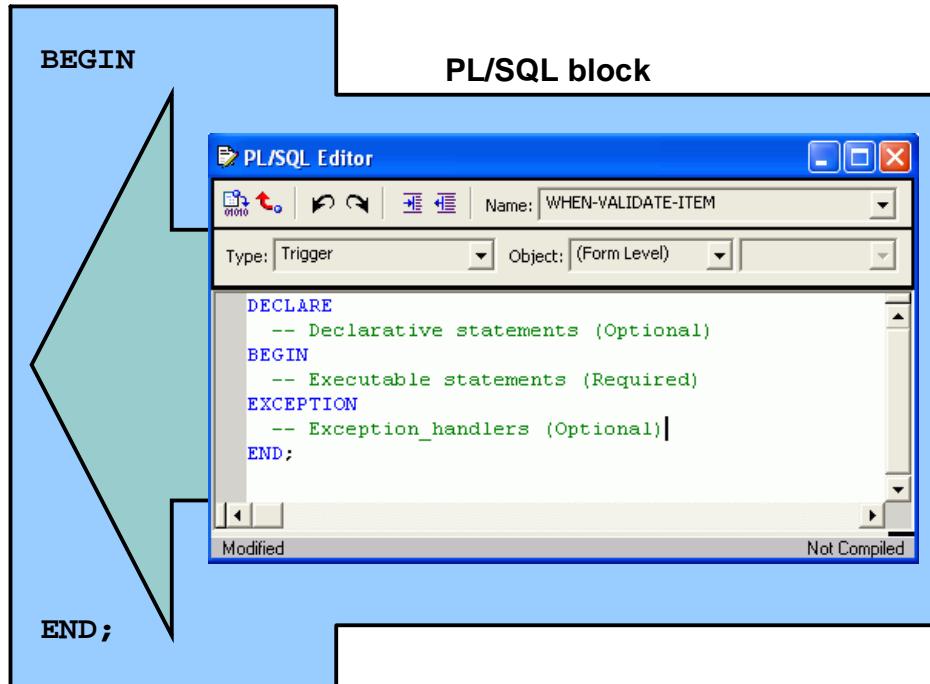
ORACLE®

Database Trigger Editor

You can use Forms Builder to define database triggers. The logical grouping of items within the Database Trigger editor enables developers to create row and statement triggers easily. An error message box displays an error when you try to retrieve, store, or drop an invalid trigger. To create a database trigger by using the Database Trigger editor, perform the following steps:

1. In the Object Navigator, expand the Database Objects node to display the schema nodes.
2. Expand a schema node to display the database objects.
3. Expand the Tables node to display the schema's database tables.
4. Select and expand the desired table.
5. Select the Triggers node as shown in the screenshot and select Edit > Create, or click Create on the toolbar. The Database Trigger editor appears.
6. In the Database Trigger editor, click New.
7. Define and save the desired program units. The compiler returns an error if the trigger does not compile successfully (as shown in the screenshot).

Writing Trigger Code



14 - 11

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Writing Trigger Code

The code text of a Forms Builder trigger is a PL/SQL block that consists of three sections as shown in the slide:

- A declaration section for variables, constants, and exceptions (optional)
- An executable statements section (required)
- An exception handlers section (optional)

If your trigger code does not require defined variables, you do not need to include the BEGIN and END keywords; they are added implicitly. The graphic in the slide illustrates the concept of implicit BEGIN and END keywords surrounding the trigger code.

Writing Trigger Code (continued)

Examples

1. If the trigger does not require declarative statements, the BEGIN and END keywords are optional. The When-Validate-Item trigger:

```
IF :order_items.unit_price IS NULL THEN
    :order_items.unit_price := :products.list_price;
END IF;
calculate_total; -- User-named procedure
```

2. If the trigger requires declarative statements, the BEGIN and END keywords are required. The When-Button-Pressed trigger:

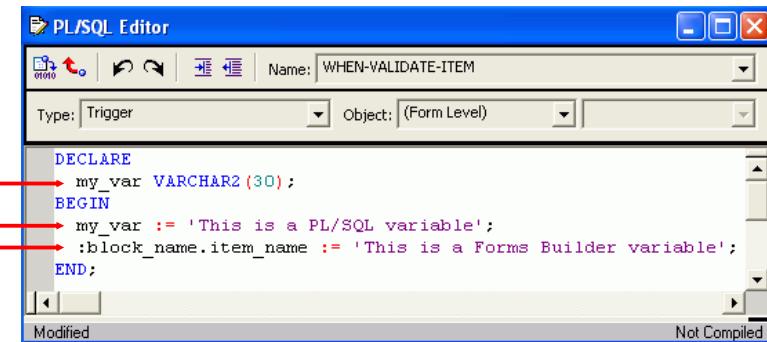
```
DECLARE
vn_discount NUMBER;
BEGIN
vn_discount:=calculate_discount
(:order_items.product_id,:order_items.quantity);
MESSAGE('Discount: ' || TO_CHAR(vn_discount));
END;
```

3. To handle exceptions, include the EXCEPTION section in the trigger. The Post-Insert trigger:

```
INSERT INTO LOG_TAB (LOG_VAL, LOG_USER)
VALUES(:departments.department_id,:GLOBAL.username);
EXCEPTION
WHEN OTHERS THEN
MESSAGE('Error! ', || SQLERRM);
```

Using Variables in Triggers

- PL/SQL variables must be declared in a trigger or defined in a package.



```
PL/SQL Editor
Name: WHEN-VALIDATE-ITEM
Type: Trigger Object: (Form Level)
DECLARE
  my_var VARCHAR2(30);
BEGIN
  my_var := 'This is a PL/SQL variable';
  :block_name.item_name := 'This is a Forms Builder variable';
END;
Modified Not Compiled
```

- Forms Builder variables:
 - Are not formally declared in PL/SQL
 - Need a colon (:) prefix in reference

ORACLE®

14 - 13

Copyright © 2009, Oracle. All rights reserved.

Using Variables in Triggers

In triggers and subprograms, Forms Builder generally accepts two types of variables for storing values:

- PL/SQL variables:** These must be declared in a DECLARE section, and remain available until the end of the declaring block. They are not prefixed by a colon. If declared in a PL/SQL package, a variable is accessible across all triggers that access this package. The screenshot in the slide shows the DECLARE section:

```
DECLARE
  my_var VARCHAR2(30);
```
- Forms Builder variables:** These are the variable types maintained by the Forms Builder. These are seen by PL/SQL as external variables, and require a colon (:) prefix to distinguish them from PL/SQL objects (except when the name is passed as a character string to a subprogram). Forms Builder variables are not formally declared in a DECLARE section, and can exist outside the scope of a PL/SQL block. The screenshot in the slide shows the use of both the PL/SQL variable and a Forms variable in code. A similar example is as follows:

```
my_var := :block_name.item_name;
```

This code assigns to the PL/SQL variable the value contained in the Forms item. You can also assign values to Forms items in code.

Using Forms Builder Variables

Variable type	Purpose	Syntax
Items	Presentation and user interaction	:block_name.item_name
Global variable	Sessionwide character variable	:GLOBAL.variable_name
System variables	Form status and control	:SYSTEM.variable_name
Parameters	Passing values in and out of module	:PARAMETER.name

ORACLE®

14 - 14

Copyright © 2009, Oracle. All rights reserved.

Using Forms Builder Variables

Form Builder Variable Type	Scope	Use
Item (text, list, check box, and so on)	Current form and attached menu	Presentation and interaction with user
Global variable	All modules in the current session	Sessionwide storage of character data
System variable	Current form and attached menu	Form status and control Note: The contents of system variables are in uppercase.
Parameter	Current module	Passing values in and out of the module

Using Forms Builder Variables (continued)

In each of the following examples of using Forms Builder variables, note that a colon (:) prefixes Forms Builder variables, and a period (.) separates the components of their name. The examples are not complete triggers.

Examples

1. References to items should be prefixed with the name of the owning Forms Builder block, which prevents ambiguity when items of the same name exist in different blocks. This is also more efficient than the item name alone:
`:block3.product_id := :block2.product_id;`
2. References to global variables must be prefixed with the word “global.” They may be created as the result of an assignment:
`:GLOBAL.customer_id := :block1.id;`
3. References to system variables must be prefixed with the word “system,” and the contents must be in uppercase ('NORMAL', not 'normal'):
`IF :SYSTEM.MODE = 'NORMAL' THEN
 ok_to_leave_block := TRUE;
END IF;`
4. Parameters defined at design-time have the prefix parameter:
`IF :PARAMETER.starting_point = 2 THEN
 GO_BLOCK('block2'); -- built-in procedure
END IF;`

Initializing Global Variables with Default Value

You can use the DEFAULT_VALUE built-in to assign a value to a global variable. Forms Builder creates the global variable if it does not exist. If the value of the indicated variable is not null, DEFAULT_VALUE does nothing. The following example creates a global variable named `country` and initializes it with the value TURKEY:

```
Default_Value('TURKEY', 'GLOBAL.country');
```

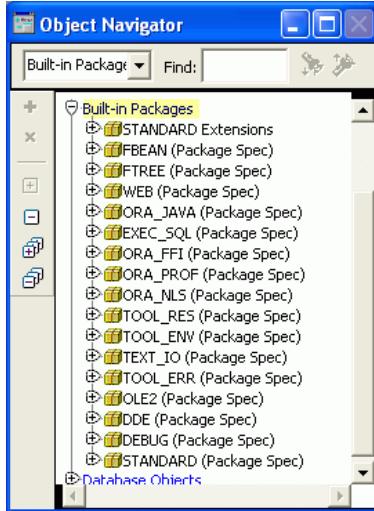
Removing Global Variables

You can use the ERASE built-in to remove a global variable. Globals always allocate 255 bytes of storage. To ensure that performance is not impacted more than necessary, always erase any global variable when it is no longer needed.

Adding Functionality with Built-in Subprograms

Built-ins belong to either of the following:

- The Standard Extensions package where no prefix is required
- Another Forms Builder package where a prefix is required



ORACLE®

14 - 16

Copyright © 2009, Oracle. All rights reserved.

Adding Functionality with Built-in Subprograms

Forms Builder provides a set of predefined subprograms as part of the product. These subprograms are defined within built-in packages as either a procedure or function. The screenshot shows the categories of built-ins in the Object Navigator.

Forms Builder built-in subprograms belong to one of the following:

- **Standard Extensions package:** Contains hundreds of core Forms built-ins that are integrated into the Standard PL/SQL command set in Forms Builder. You can call them directly, without any package prefix.
Example: EXECUTE_QUERY ;
- **Other Forms Builder packages:** Subprograms in other built-in packages provide functionality related to a particular supported feature. These require the package name as a prefix when called.
Example: ORA_JAVA.CLEAR_EXCEPTION ;

Note: All the built-in subprograms used in this lesson are part of the Standard Extensions package.

Adding Functionality with Built-in Subprograms (continued)

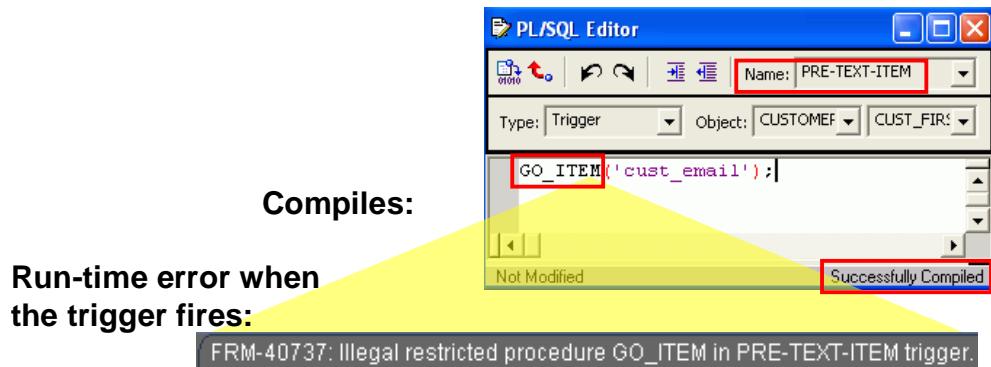
In addition to the standard extensions, Forms Builder provides the following packages:

Package	Description
DDE	Provides Dynamic Data Exchange support
DEBUG	Provides built-ins for debugging PL/SQL program units
EXEC_SQL	Provides built-ins for executing dynamic SQL within PL/SQL procedures
FBEAN	Provides built-ins to interact with client-side Java beans
FTREE	Provides built-ins for manipulating hierarchical tree items
OLE2	Provides a PL/SQL API for creating, manipulating, and accessing attributes of OLE2 automation objects
ORA_FFI	Provides built-ins for calling out to foreign (C) functions from PL/SQL
ORA_JAVA	Enables you to call Java procedures from PL/SQL
ORA_NLS	Enables you to extract high-level information about your current language environment
ORA_PROF	Provides built-ins for tuning PL/SQL program units
STANDARD	Globally declares types, exceptions, and subprograms that are available to every PL/SQL program
TEXT_IO	Provides built-ins to read and write information to and from files
TOOL_ENV	Enables you to interact with Oracle environment variables
TOOL_ERR	Enables you to access and manipulate the error stack created by other built-in packages such as Debug
TOOL_RES	Provides built-ins to manipulate resource files
WEB	Provides built-ins for the Web environment

Note: Some of these packages, such as OLE2, ORA_FFI, and TEXT_IO, function on the application server side, not on the client side. For example, TOOL_ENV enables you to get and set environment variables on the application server. To interact with the client, you would need to provide similar functionality in a JavaBean.

Limits of Use

- Unrestricted built-ins are allowed in any trigger or subprogram.
- Restricted built-ins are allowed only in certain triggers and subprograms called from such triggers.
- Consult the Help system.



14 - 18

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Limits of Use

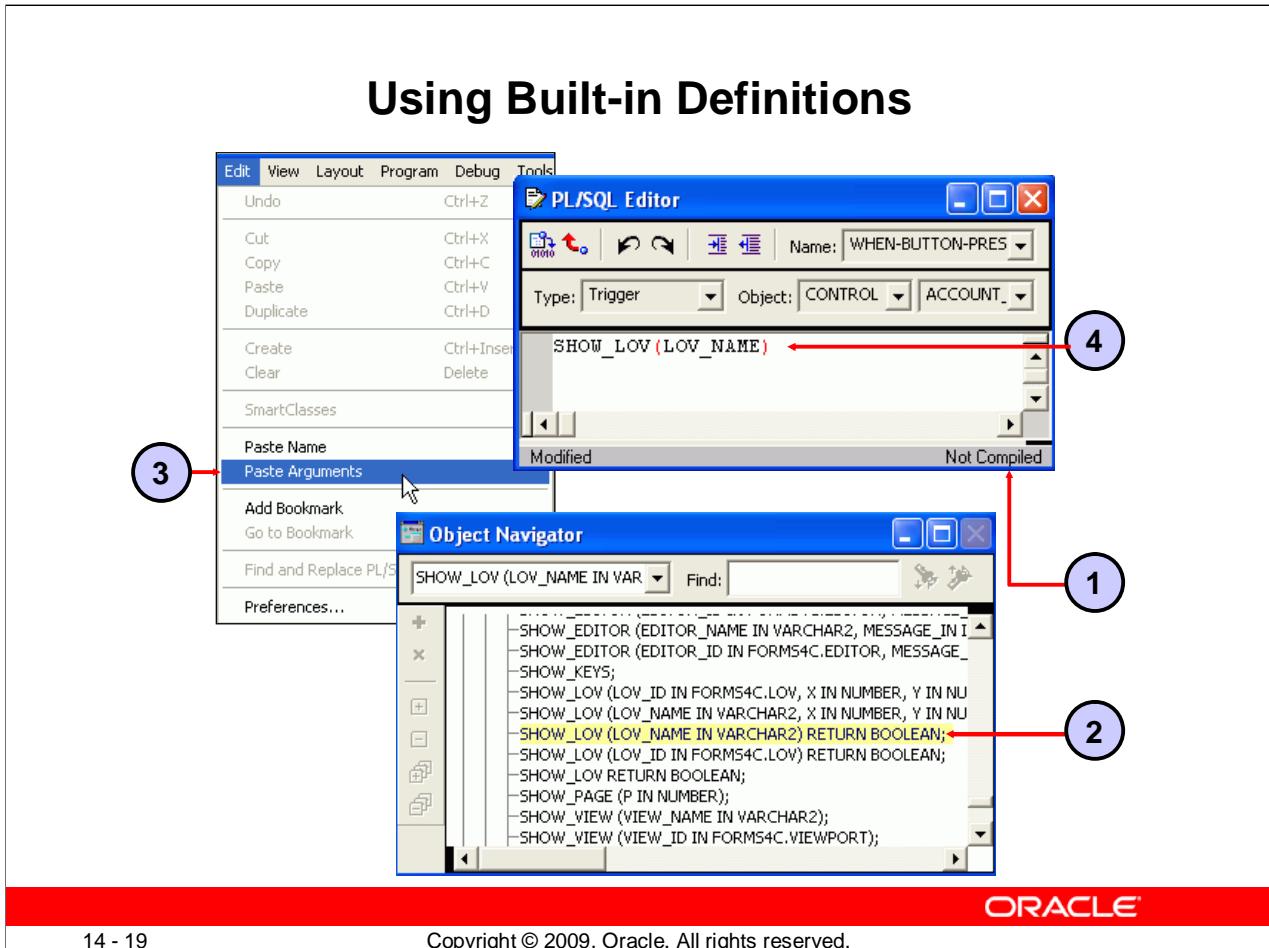
You can call built-ins in any trigger or user-named subprogram in which you use PL/SQL. However, some built-ins provide functionality that is not allowed in certain trigger types. Built-ins are, therefore, divided into two groups:

- **Unrestricted built-ins:** Unrestricted built-ins do not affect logical or physical navigation and can be called from any trigger, or from any subprogram.
- **Restricted built-ins:** Restricted built-ins affect navigation in your form, either external screen navigation, or internal navigation. You can call these built-ins only from triggers while no internal navigation occurs. The online Help specifies which groups of built-ins can be used in each trigger.

Calling a restricted built-in from a navigational trigger compiles successfully but causes a run-time error. For example, the screenshots show that you can compile a Pre-Text-Item trigger that calls the restricted GO_ITEM built in. However, when the trigger fires at run time, it produces the following error:

FRM-40737: Illegal restricted procedure GO_ITEM in PRE-TEXT-ITEM trigger.

Using Built-in Definitions



Using Built-in Definitions

When you are writing a trigger or a program unit, the Forms Builder enables you to look up built-in definitions, and optionally copy their names and argument prototypes into your code. The following steps are illustrated in the corresponding numbered screenshots in the slide:

1. Place the cursor at the point in your PL/SQL code (in the PL/SQL Editor) where a built-in subprogram is to be called.
2. Expand the Built-in Packages node in the Navigator, and select the procedure or function that you need to use (usually from Standard Extensions).
3. If you want to copy the built-in prototype arguments or name, or both, select **Edit > Paste Name** or **Edit > Paste Arguments** from the menu (**Paste Arguments** includes both the built-in name and its arguments).
4. The definition of the built-in is copied to the cursor position in the PL/SQL Editor, where you can insert your own values for arguments, as required.

Using Built-in Definitions (continued)

Note: A subprogram can be either a procedure or a function. Built-in subprograms are therefore called in two distinct ways:

- **Built-in procedures:** Called as a complete statement in a trigger or program unit with mandatory arguments
- **Built-in functions:** Called as part of a statement, in a trigger or program unit, at a position where the function's return value will be used. Again, the function call must include any mandatory arguments.

Example

The SHOW_LOV built-in is a function that returns a Boolean value (indicating whether the user has chosen a value from the LOV). It might be called as part of an assignment to a Boolean variable, as in the following example (not a complete trigger):

```
DECLARE
    customer_chosen BOOLEAN;
BEGIN
    customer_chosen := SHOW_LOV('customer_list');
    .
    .
```

Useful Built-Ins

- EDIT_TEXTITEM
- ENTER_QUERY, EXECUTE_QUERY
- EXIT_FORM
- GET_ITEM_PROPERTY, SET_ITEM_PROPERTY
- GO_BLOCK, GO_ITEM
- LIST_VALUES
- MESSAGE
- SHOW_ALERT, SHOW_EDITOR, SHOW_LOV
- SHOW_VIEW, HIDE_VIEW

ORACLE®

14 - 21

Copyright © 2009, Oracle. All rights reserved.

Useful Built-Ins

The table on the next page describes some built-ins that you can use in triggers to add functionality to items. They are discussed in later lessons.

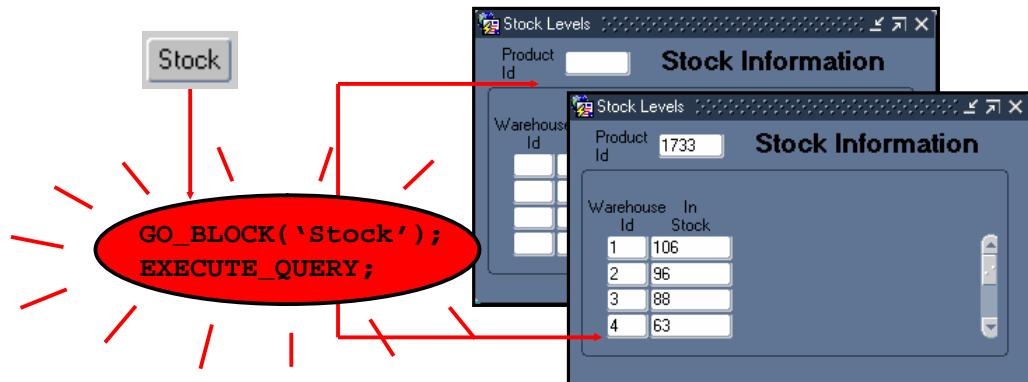
Useful Built-Ins (continued)

Built-in Subprogram	Description
EDIT_TEXTITEM procedure	Invokes the Form Runtime item editor for the current text item
ENTER_QUERY procedure	Clears the current block and creates a sample record. Operators can then specify query conditions before executing the query with a menu or button command. If there are changes to commit, the Forms Builder prompts the operator to commit them before continuing ENTER-QUERY processing.
EXECUTE_QUERY procedure	Clears the current block, opens a query, and fetches a number of selected records. If there are changes to commit, Forms Builder prompts the operator to commit them before continuing EXECUTE-QUERY processing.
EXIT_FORM procedure	If in normal mode, exits the current form; if in ENTER-QUERY mode, cancels the query
GET_ITEM_PROPERTY function	Returns specified property values for the indicated item
GO_BLOCK procedure	Navigates to the specified block
GO_ITEM procedure	Navigates to the specified item
HIDE_VIEW procedure	Hides the indicated canvas
LIST_VALUES procedure	Invokes the LOV attached to the current item
MESSAGE procedure	Displays specified text on the message line
SET_ITEM_PROPERTY procedure	Changes the setting of specified property for an item
SHOW_ALERT function	Displays the given alert and returns a numeric value when the operator selects one of three alert buttons
SHOW_EDITOR procedure	Displays the specified editor at the given coordinates and passes a string to the editor, or retrieves an existing string from the editor
SHOW_LOV function	Invokes a specified LOV and returns a Boolean value that indicates whether the user selected a value from the list
SHOW_VIEW procedure	Displays the indicated canvas at the coordinates specified by the X Position and Y Position of the canvas property settings. If the view is already displayed, SHOW_VIEW raises it in front of any other views in the same window.

Using the When-Button-Pressed Trigger

The When-Button-Pressed trigger:

- Fires when the operator clicks a button
- Accepts restricted and unrestricted built-ins
- Is used to provide convenient navigation, and to display LOVs and many other frequently used functions



Using the When-Button-Pressed Trigger

The When-Button-Pressed trigger fires when the user clicks a button. You can define the trigger on an individual item or at higher levels if required.

When-Button-Pressed accepts both restricted and unrestricted built-ins. You can use buttons to provide a wide range of functions for users. These functions include:

- Navigation
- Displaying LOVs
- Invoking calculations and other functions

Example

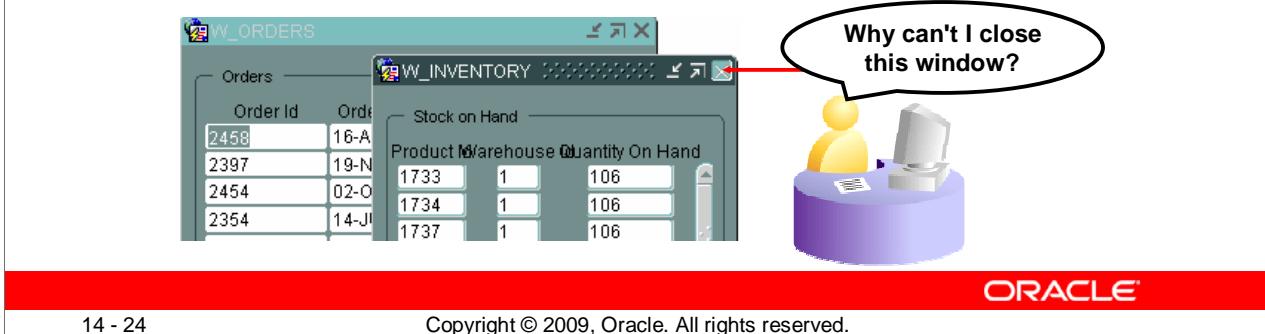
The example in the slide shows a Stock_Button in the CONTROL block that is situated on the TOOLBAR canvas of the Orders form. When clicked, the button activates the When-Button-Pressed trigger. The trigger code results in navigation to the INVENTORIES block and execution of a query on the INVENTORIES block.

```
GO_BLOCK('inventories');
EXECUTE_QUERY;
```

Using the When-Window-Closed Trigger

The When-Window-Closed trigger:

- Fires when the operator closes a window by using a window manager-specific close command
- Accepts restricted and unrestricted built-ins
- Is used to programmatically close a window when the operator issues a window manager-specific close command. You can close a window by using built-ins.



14 - 24

Copyright © 2009, Oracle. All rights reserved.

Using the When-Window-Closed Trigger

The screenshot shows a user clicking the X at the upper-right of a window and wondering why it does not close. Forms Builder does not close a window when the operator issues a window manager-specific close command; it only fires the When-Window-Closed trigger. It is the developer's responsibility to write the required functionality in this trigger.

Use the When-Window-Closed trigger to close a window programmatically when the operator issues the window manager close command. You can close a window with the HIDE_WINDOW, SET_WINDOW_PROPERTY, and EXIT_FORM built-in subprograms. The When-Window-Closed trigger accepts both restricted and unrestricted built-ins. You define this trigger at the form level.

You cannot hide the window that contains the current item.

Example

When the operator issues the window manager-specific close command, the following code in a When-Window-Closed trigger closes the WIN_INVENTORY window by setting the VISIBLE property to FALSE.

```
GO_ITEM('orders.order_id');
SET_WINDOW_PROPERTY('win_inventory',VISIBLE,PROPERTY_FALSE);
```

Summary

In this lesson, you should have learned that:

- You can use the PL/SQL Editor to write trigger code
- Trigger code has three sections
- You can add functionality by calling built-in subprograms from triggers
- Restricted built-ins are not allowed in triggers that fire while the navigation is occurring
- The When-Button-Pressed trigger fires when the user presses a button
- The When-Window-Closed trigger fires when the user closes a window



Summary

- To create a trigger:
 - Select a scope in the Object Navigator
 - Create a trigger and select the trigger type from the LOV, or use the SmartTriggers menu option
 - Define code in the PL/SQL Editor. Trigger code has three sections:
 - Declaration section (optional)
 - Executable statements section (required)
 - Exception handlers section (optional)
 - Compile
- Find built-ins in the Navigator under Built-in Packages:
 - Paste the built-in name and arguments to your code by using the Paste Name and Paste Arguments options.
 - Refer to online Help.
- The When-Button-Pressed trigger provides a wide range of functionality to users.
- Use the When-Window-Closed trigger to provide functionality when the user issues a window manager-specific Close command.

Practice 14: Overview

This practice covers the following topics:

- Using built-ins to display LOVs
- Using the When-Button-Pressed and When-Window-Closed triggers to add functionality to applications
- Using built-ins to display and hide the Help stack canvas



Practice 14: Overview

This practice focuses on how to use the When-Button-Pressed and When-Window-Closed triggers.

15

Debugging Triggers

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe the components of the Debug Console
- Use the Run Form Debug button to run a form module in debug mode
- Explain how to use remote debugging
- Debug PL/SQL code



Lesson Aim

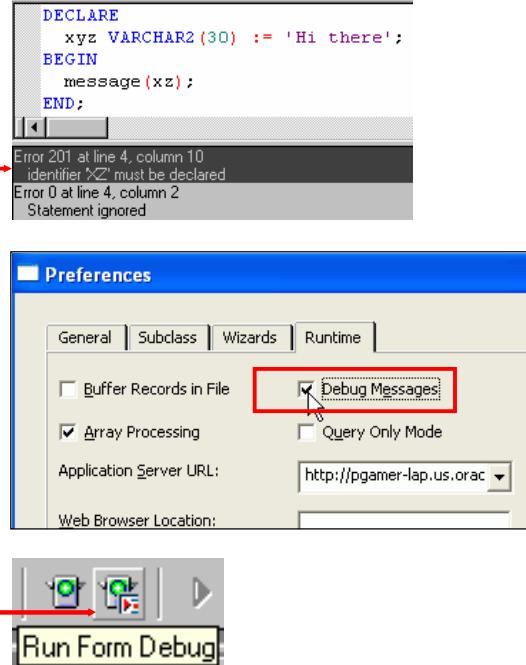
As you begin to add code to your form by writing triggers, you will probably find that you sometimes obtain incorrect or unexpected results. In a large application, it may be difficult to determine the source of the problem.

You can use Forms Builder's integrated PL/SQL Debugger to locate and correct coding errors. This lesson explains how to debug your PL/SQL code.

Debugging Process

Monitor and debug triggers by:

- Compiling and correcting errors in the PL/SQL Editor
- Displaying debug messages at run time
- Invoking the PL/SQL Debugger



Debugging Process

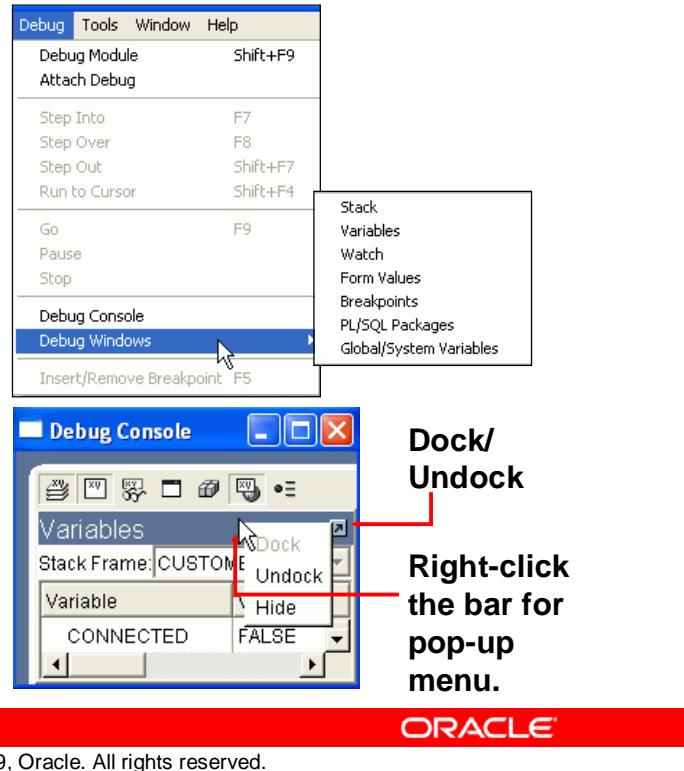
With Forms Builder, you can monitor and debug triggers in several ways:

- **Compiling:** Syntax errors and object reference errors (including references to database objects) are reported when you compile a trigger or generate the form module, as shown in the top screenshot in the slide. This enables you to correct these problems in the PL/SQL Editor before run time.
- **Running a form with run-time parameter `debug_messages=Yes`:** You can set preferences for messages to be displayed to indicate when the triggers fire. This helps you to see whether certain triggers are firing, their origin and level, and the time at which they fire. If an error occurs, you can look at the last trigger that fired to narrow the scope of the source of the error.
- **Invoking the PL/SQL Debugger:** You invoke the Debugger from Forms Builder by clicking Run Form Debug on the toolbar.

With the Debugger, you can monitor the execution of code within a trigger and other program units. You can step through the code on a line-by-line basis, and you can monitor called subprograms and variables as you do so. You can also modify variables as the form is running, which allows you to test how various changes in form item values and variable values affect your application.

Debug Console

- Stack
- Variables
- Watch
- Form Values
- PL/SQL Packages
- Global/System Variables
- Breakpoints



Debug Console

The Debug Console is a workspace in Forms Builder that enables you to see various aspects of the running form and its environment. You can display the Debug Console by selecting Debug > Debug Console from the menu. It displays automatically when you encounter a breakpoint in a form that is running in debug mode.

Within the console, you can display several types of panels: Stack, Variables, Watch, Form Values, PL/SQL Packages, Global/System Variables, and Breakpoints. You can resize the Debug Console and any of the panels displayed within it.

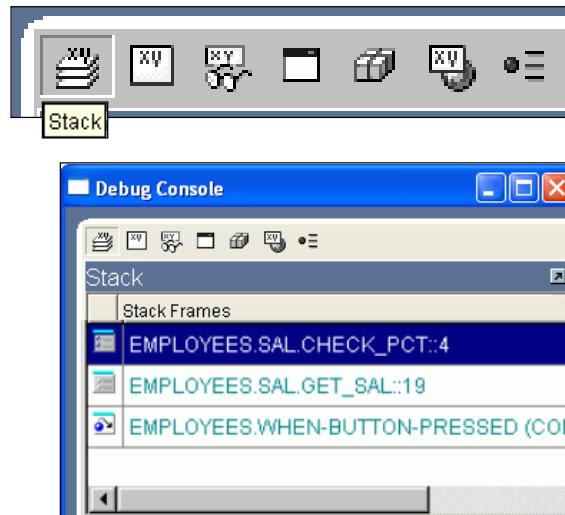
You choose which panels to display or close by clicking the toggle buttons on the toolbar of the Debug Console, or by selecting Debug > Debug Windows from the menu. As you show and hide panels within the console, the panels automatically expand and contract to fill the console.

You can undock any of the panels to display them outside the Debug Console by clicking the up-arrow button in the top-right of the panel; you redock the panel by clicking its down-arrow button.

When you right-click the area beside the Dock/Undock arrow buttons, a pop-up menu is displayed. This enables you to hide, dock, or undock the panel.

Debug Console: Stack Panel

- Stack
- Variables
- Watch
- Form Values
- PL/SQL Packages
- Global/System Variables
- Breakpoints



Debug Console: Stack Panel

The call stack represents the chain of subprograms starting from the initial entry point to the currently executing subprogram. The call stack can be displayed by clicking the left-most button on the Debug Console toolbar.

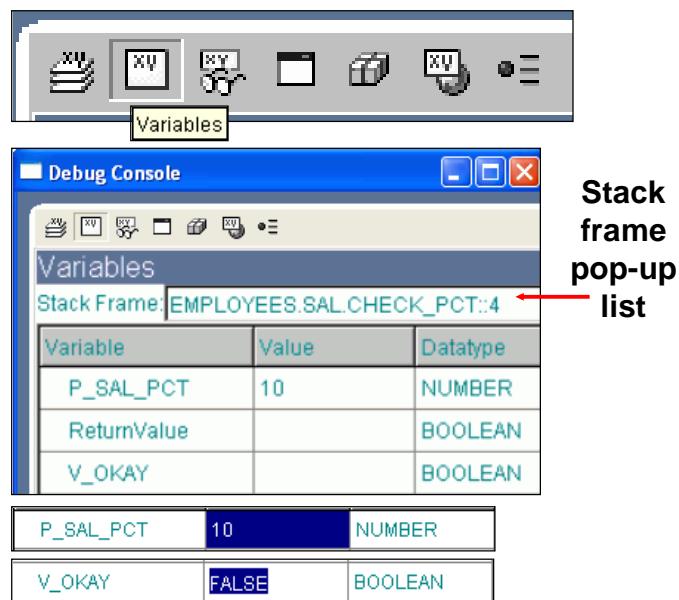
In the Stack panel, frames are listed in the reverse order in which the subprograms were executed. The earliest frame is at the bottom of the stack, while the latest frame is at the top of the stack.

As you can see in the screenshot in the slide, the Stack panel shows that the program currently executing is EMPLOYEES . SAL . CHECK_PCT, which was called by EMPLOYEES . SAL . GET_SAL. This program was called from a When-Button-Pressed trigger, the initial entry point of the current call stack.

A stack frame contains information about the corresponding subprogram, including the module name, the package name if any, the subprogram name, and the next statement that is to be executed. For example, EMPLOYEES . SAL . GET_SAL : : 19 indicates that line 19 of the subprogram GET_SAL in the SAL package contained in the Employees module will be executed when the application returns to that subprogram. When that occurs, the EMPLOYEES . SAL . CHECK_PCT frame is pushed off the stack because the EMPLOYEES . SAL . CHECK_PCT subprogram has finished executing.

Debug Console: Variables Panel

- Stack
- **Variables**
- Watch
- Form Values
- PL/SQL Packages
- Global/System Variables
- Breakpoints



Debug Console: Variables Panel

The Variables panel displays the variables of the current stack frame, along with their values. You can display the variables panel by clicking the second button on the Debug Console toolbar.

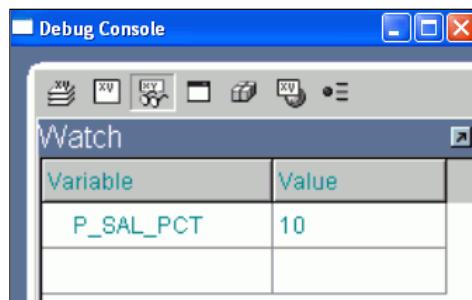
You can select the stack frame whose variables you want to view from a pop-up list. You can also switch the variables shown in the Variables panel by clicking a different stack frame in the Stack panel if it is open as well. This does not change the order of execution of program statements, but only the information that is displayed in the Debug Console.

A valuable feature of the Debugger is that when the form is suspended, you can change the variable values by clicking in the Value column and entering a new value. When the program continues, it will use the new value that you have entered, so that you can test the effect of changing a value on the final result.

Some variables, such as parameters, cannot be modified. When you click a read-only variable, the entire cell is highlighted, as illustrated in the screenshot in the slide. Clicking a modifiable variable highlights only the value, not the entire cell. This is shown in the last screenshot in the slide.

Debug Console: Watch Panel

- Stack
- Variables
- Watch
- Form Values
- PL/SQL Packages
- Global/System Variables
- Breakpoints



Debug Console: Watch Panel

A running application may have dozens of variables that can be displayed in various panels in the Debug Console, but there may be very few that you need to monitor. The Watch panel provides a central place where you can keep track of any valid variables that you specify. You can display the Watch panel by clicking the third button on the Debug Console toolbar.

Only variables that resolve to a scalar value are valid. Stored package variables are not valid. After a variable is displayed in your watch list, when execution is next suspended at a different location, the variable values in the list are updated as needed if they are available in the current execution context. If a variable is not defined in the currently executing subprogram, “#####” is displayed in the cell instead of a value.

To add a variable to the Watch panel, perform the following steps:

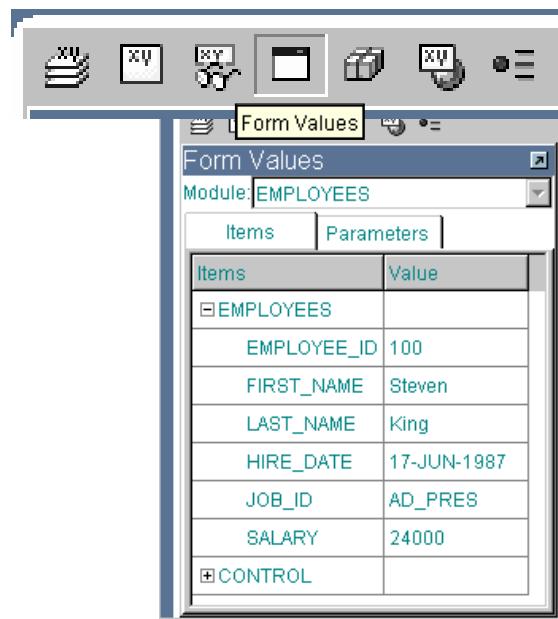
1. Open the window where the variable is displayed.
2. Select the variable that you want to add to the Watch panel.
3. Right-click the selection and select “Add to Watch” from the pop-up menu.

To delete an item from the Watch panel, perform the following:

- Select the item in the Watch panel, right-click and select Remove from the pop-up menu.
- To clear the entire Watch panel, select Remove All.

Debug Console: Form Values Panel

- Stack
- Variables
- Watch
- Form Values
- PL/SQL Packages
- Global/System Variables
- Breakpoints



ORACLE®

Debug Console: Form Values Panel

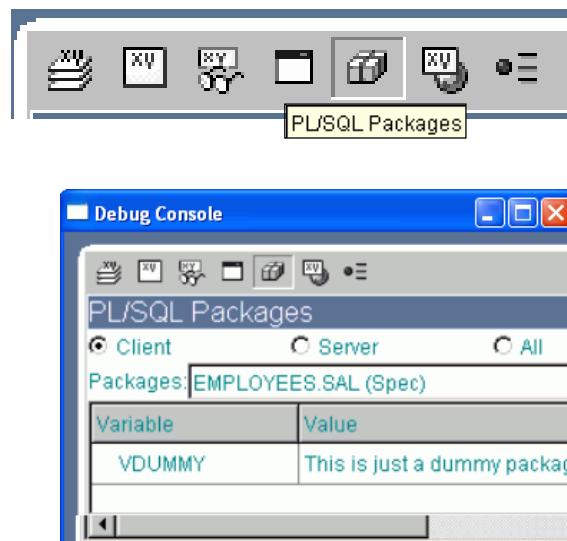
You can use the Form Values panel to display the values of all items and parameters in modules that are currently running. You can display the Form Values panel by clicking the fourth button on the Debug Console toolbar.

You switch between a view of items and a view of parameters by clicking the corresponding tabs in the panel.

You can change the values of modifiable items to test the effects of such changes. If the value is read-only, such as display item values, the entire cell is highlighted when you try to edit it. If the value is modifiable, only the value in the cell is highlighted when you select it.

Debug Console: PL/SQL Packages Panel

- Stack
- Variables
- Watch
- Form Values
- PL/SQL Packages
- Global/System Variables
- Breakpoints



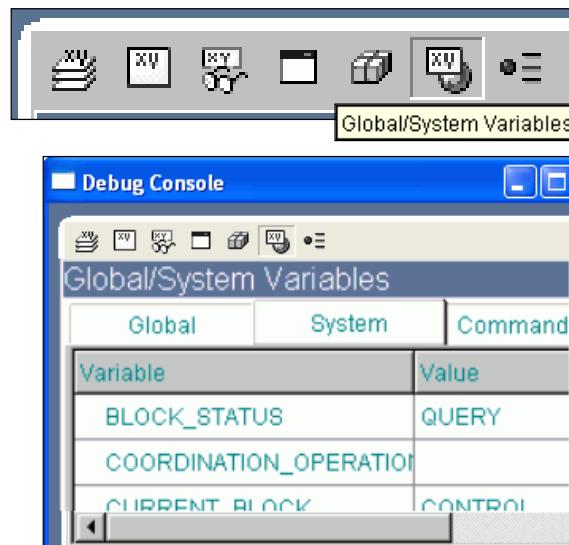
Debug Console: PL/SQL Packages Panel

Use the PL/SQL Packages panel to browse and examine PL/SQL packages that have been instantiated. You can display the PL/SQL Packages panel by clicking the fifth button on the Debug Console toolbar.

Both package specification and package body global variables are listed. You can view packages only while the Run Form process is currently executing PL/SQL. You can also select an option button to determine which packages are displayed: Client, Server, or All.

Debug Console: Global/System Variables Panel

- Stack
- Variables
- Watch
- Form Values
- Loaded PL/SQL Packages
- Global/System Variables
- Breakpoints



Debug Console: Global/System Variables Panel

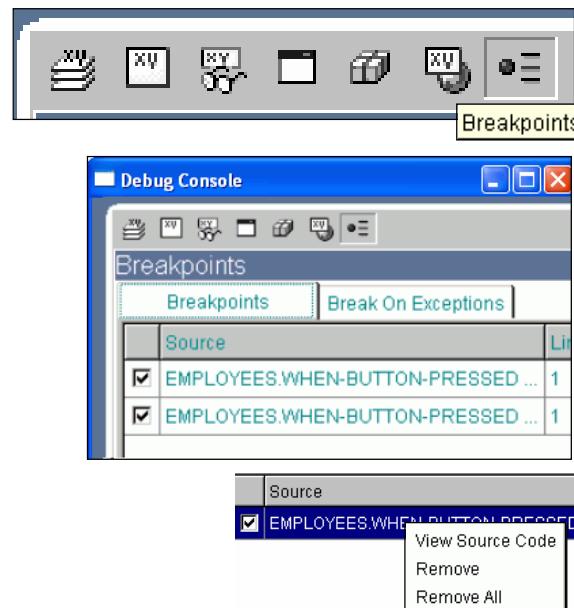
Use the Global/System Variables panel to display the current system, global, and command-line variables and their values. You can display the Global/System panel by clicking the sixth button on the Debug Console toolbar.

You can switch among these types of variables by clicking the corresponding tabs in the panel.

Command-line variables and most system variables are read-only. The only modifiable system variables are DATE_THRESHOLD, EFFECTIVE_DATE, MESSAGE_LEVEL, and SUPPRESS_WORKING. You can modify global variables, and the new values will be used subsequently by the running application.

Debug Console: Breakpoints Panel

- Stack
- Variables
- Watch
- Form Values
- Loaded PL/SQL Packages
- Global/System Variables
- Breakpoints



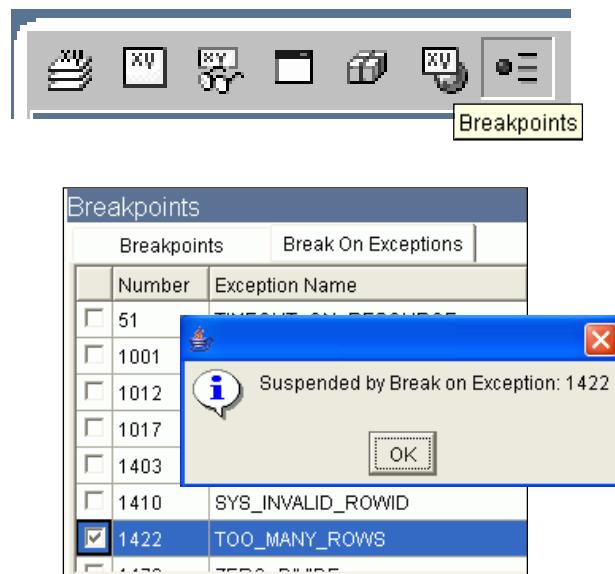
Debug Console: Breakpoints Panel

You can display the Breakpoints panel by clicking the seventh button on the Debug Console toolbar. The Breakpoints panel consists of two tabs:

- The Breakpoints tab, which displays any breakpoints set in the code during the current Forms Builder session, in the order of breakpoint creation. The display includes the name of the trigger or program unit, the line number where the breakpoint is set, and a check box to temporarily enable or disable the breakpoint. It also enables navigation to the source code where the breakpoint is set by:
 - Double-clicking the breakpoint name
 - Highlighting (clicking) it, then selecting View Source Code from the pop-up menu (right-click in Windows). From this pop-up menu, you can also remove the breakpoint or remove all breakpoints.

Debug Console: Breakpoints Panel

- Stack
- Variables
- Watch
- Form Values
- Loaded PL/SQL Packages
- Global/System Variables
- Breakpoints



Debug Console: Breakpoints Panel (continued)

- The Break On Exceptions tab, which displays a list of frequently used system exceptions that you can use during debugging. The display includes the exception name, the associated ORA error number, and a check box that you use to set or unset the breakpoint.

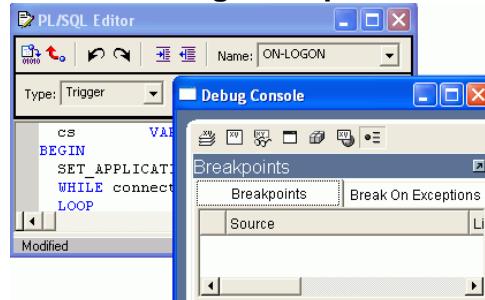
When an exception on which you have set a breakpoint is encountered at run time, a message displays the number of the exception.

Setting Breakpoints in Client Code

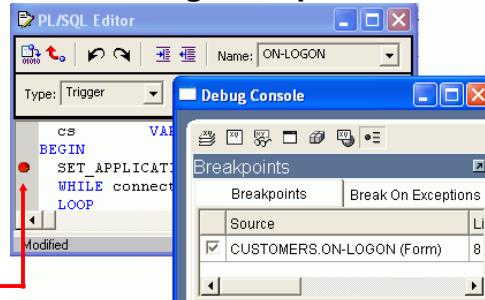
Breakpoints:

- Suspend form execution
- Return control to the Debugger
- Remain in effect for the Forms Builder session
- May be enabled and disabled
- Are set in the PL/SQL Editor on executable lines of code

Before setting breakpoint:



After setting breakpoint:



ORACLE®

Setting Breakpoints in Client Code

You can set breakpoints in code so that the form running in debug mode is suspended when a breakpoint is encountered and control returns to the Forms Builder Debugger, allowing you to monitor or change the environment at that point. When you set a breakpoint, it remains set until you exit the Forms Builder session. However, you can disable and enable breakpoints as needed by selecting and deselecting the check box in the Breakpoints panel of the Debug Console.

You can set breakpoints only on executable lines of code, such as assignments or subprogram calls. There are three ways to set a breakpoint:

- By double-clicking to the left of a line of code in the PL/SQL Editor (double-click again to remove the breakpoint)
- By right-clicking a line of code and selecting Insert/Remove Breakpoint
- By selecting Debug > Insert/Remove Breakpoint from the main menu

Performing the same action again unsets the breakpoint. You can also remove one or all breakpoints from the pop-up menu in the Breakpoint panel, as described previously.

The screenshots in the slide show that when you set a breakpoint in code, a red dot appears in the left margin of the code editor; the breakpoint appears in the list in the Breakpoints panel of the debugger, and can be enabled or disabled there.

Setting Breakpoints in Stored Code

- You can set breakpoints on stored program units:
 - Expand the Database Objects node.
 - Expand the <schema> node.
 - Expand the PL/SQL Stored Program Units node.
 - Double-click the program unit.
 - Set breakpoint in the PL/SQL Editor.
- You cannot set breakpoints on database triggers or stored PL/SQL libraries.
- You can compile the stored program unit with debug information.

ORACLE®

15 - 14

Copyright © 2009, Oracle. All rights reserved.

Setting Breakpoints in Stored Code

If you are connected to the database, you can set breakpoints in stored packages, procedures, and functions just as you do in client-side programs. To do so, perform the following steps:

1. Expand the Database Objects node.
2. Navigate to the stored subprogram.
3. Open it in the PL/SQL Editor.
4. Set the breakpoint as you would in client-side programs.

You cannot set breakpoints on database triggers or stored PL/SQL libraries.

If the program unit does not appear in a stack frame, or if you are not able to see it as you step through its code, it has not been compiled with debug information included. There are three methods to compile the stored program unit with debug information:

- Create the stored procedure in Forms Builder, which creates it with debugging information.
- Use `ALTER SESSION SET PLSQL_DEBUG=TRUE` before creating the stored procedure.
- Manually recompile an existing PL/SQL program unit using:
`ALTER PROCEDURE <schema.procedure> COMPILE DEBUG`

Debugging Tips

- Connect to the database for SQL compilation.
- The line that fails is not always responsible.
- Watch for missing semicolons and quotation marks.
- Define triggers at the correct level.
- Place triggers where the event will happen.

ORACLE®

15 - 15

Copyright © 2009, Oracle. All rights reserved.

Debugging Tips

- Make sure that you are connected to the (correct) database when you compile triggers that contain SQL. Error messages can be deceiving.
- The PL/SQL Editor reports the line that fails, but the error may be due to a dependency on an earlier line of code.
- Missing semicolons (;) and mismatched quotation marks are a common cause of compile errors. Check for this if a compile error does not give an obvious indication to the problem.
- If a trigger seems to fire too often, or on the wrong block or item in the form, check whether it is defined at the required level. For example, a form-level When-Validate-Item trigger fires for every changed item in the form. To check this, you can run the form with Debug Messages on.
- For triggers that populate other items, make sure the trigger belongs to the object where the firing event occurs, not on the items to be populated.

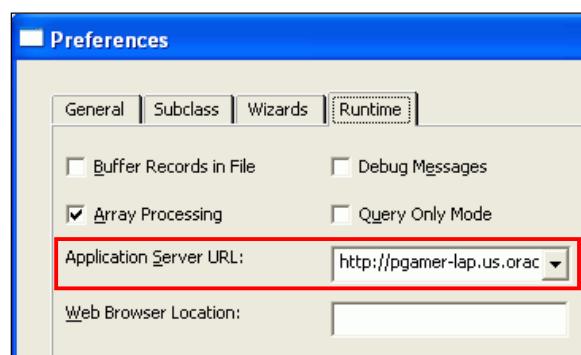
Running a Form in Debug Mode

Run Form
Debug



(Compiles automatically)

Runs Form in
debug mode on
server specified
in Runtime
Preferences



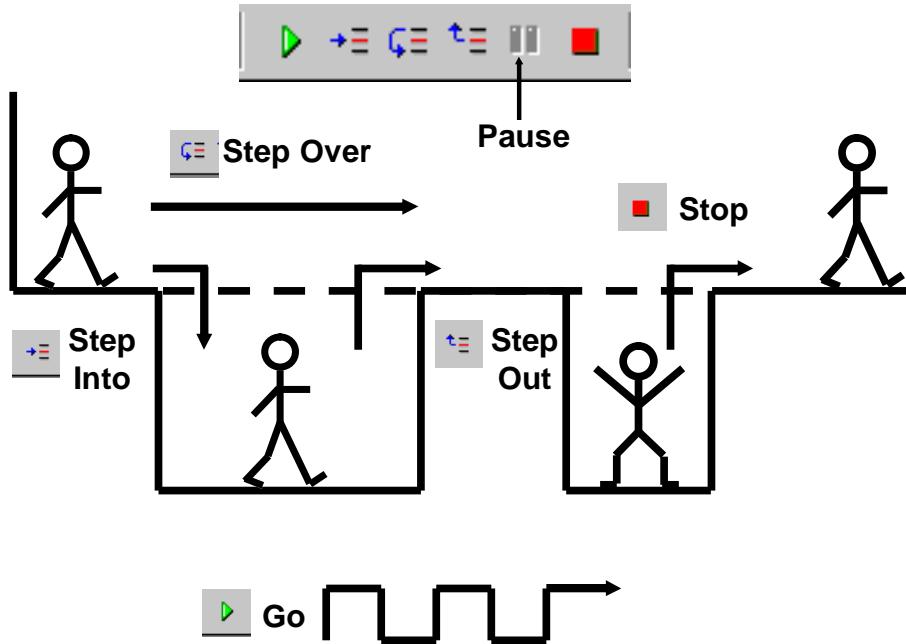
ORACLE®

Running a Form in Debug Mode

The Run Form Debug button in Forms Builder runs the form in debug mode. When a breakpoint is encountered and control passes to the Debugger in Forms Builder, you can use the debug commands to resume execution or step through the code in a variety of ways to see how each line affects the application and the environment, as you see in the next few slides.

As in the case when you run a form from Forms Builder with the Run Form button, the Run Form Debug button runs the form in a three-tier environment. It takes its settings from the Preferences window that you access by selecting Edit > Preferences from the main menu and clicking the Runtime tab.

Stepping Through Code



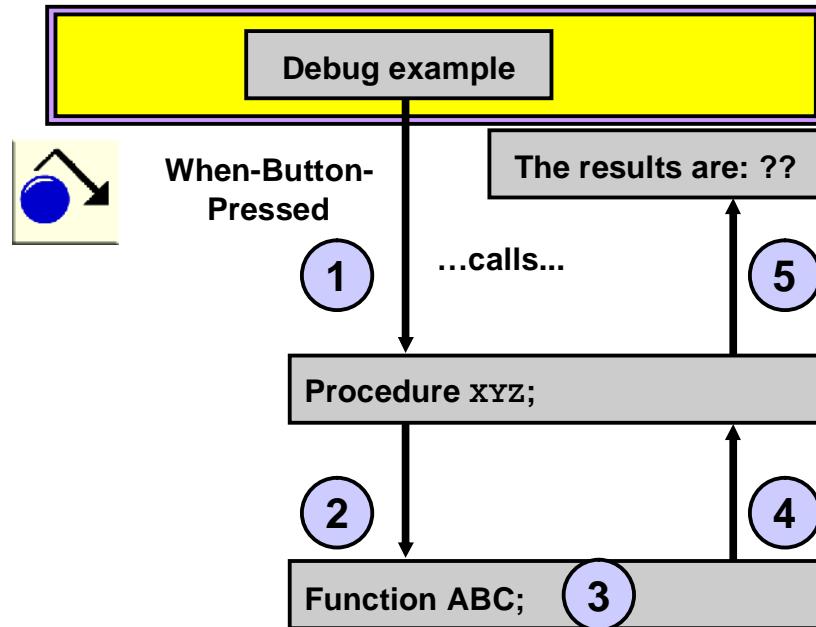
Stepping Through Code

Whether debugging a form that is running remotely or from Forms Builder, when the program encounters a breakpoint, the Debugger enables you to step through program units in a variety of ways, as illustrated in the slide, by using the following buttons:

- **Step Into:** Executes the next statement, whether in the current code or a subprogram
- **Step Over:** Executes the next statement without stepping into a nested subprogram
- **Step Out:** Completes the nested subprogram and steps to the next executable statement in the calling program
- **Go:** Resumes execution until the program terminates normally or is interrupted by the next breakpoint
- **Pause:** Pauses the execution of running PL/SQL code to enable you to examine the environment (for example, you could check variable values)
- **Stop:** Terminates debugging and program execution completely; the Debug Console and any opened debug panels close and application execution terminates

Another command, available from the Debug menu, is Run to Cursor. When you click a line of code in the PL/SQL Editor and select this command, it executes all code up to that line, and then stops and marks that line as the next executable line of code.

Debug Example Used in Demonstration



Debug Example Used in Demonstration

This simple example demonstrates some of the basic features available in the Debugger. The example form consists of a single button with trigger code for the When-Button-Pressed event. The code works as follows:

1. The trigger calls the XYZ procedure, passing it a value for the xyz_param input parameter.
2. The XYZ procedure calls the ABC function, passing it a value for the abc_param input parameter.

```
PROCEDURE xyz(xyz_param IN NUMBER) IS
    v_results NUMBER;
BEGIN
    v_results := ABC(10);
    v_results := v_results + xyz_param;
    MESSAGE('The results are: ' || TO_CHAR(v_results));
END xyz;
```

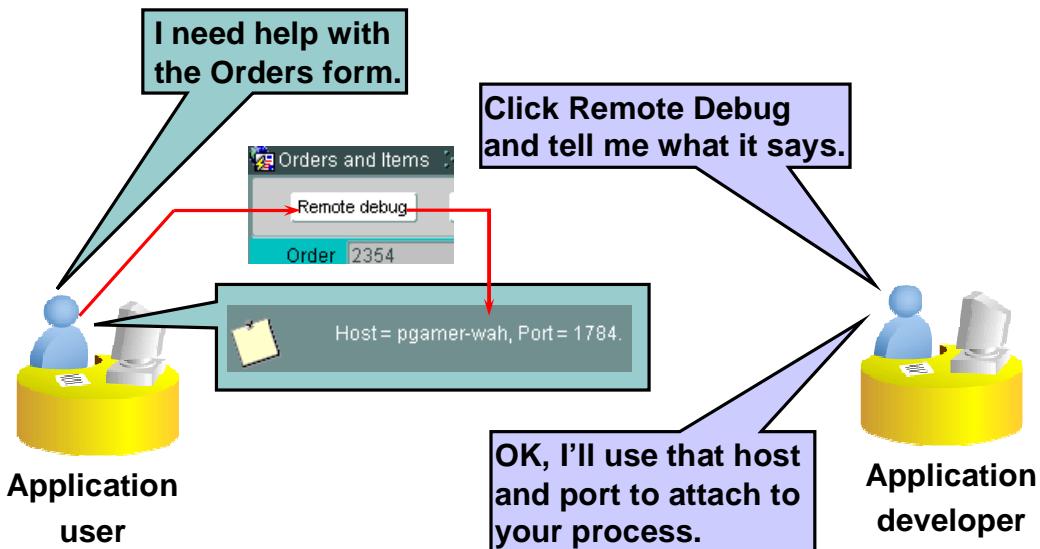
Debug Example Used in Demonstration (continued)

3. The ABC function multiplies two variables and adds the result to the abc_param input parameter.
4. The ABC function returns the result to the XYZ procedure.

```
FUNCTION abc (abc_param IN NUMBER) RETURN NUMBER IS
    v_total NUMBER := 0;
    v_num3 NUMBER := 3;
    v_num6 NUMBER := 8;
    /*-- wrong value should be 6 */
    BEGIN
        v_total := v_num3 * v_num6;
        v_total := v_total + abc_param;
        RETURN v_total;
    END abc;
```

5. The XYZ procedure adds the result to xyz_param and displays it in the console at the bottom of the form window.

Remote Debugging



ORACLE®

15 - 20

Copyright © 2009, Oracle. All rights reserved.

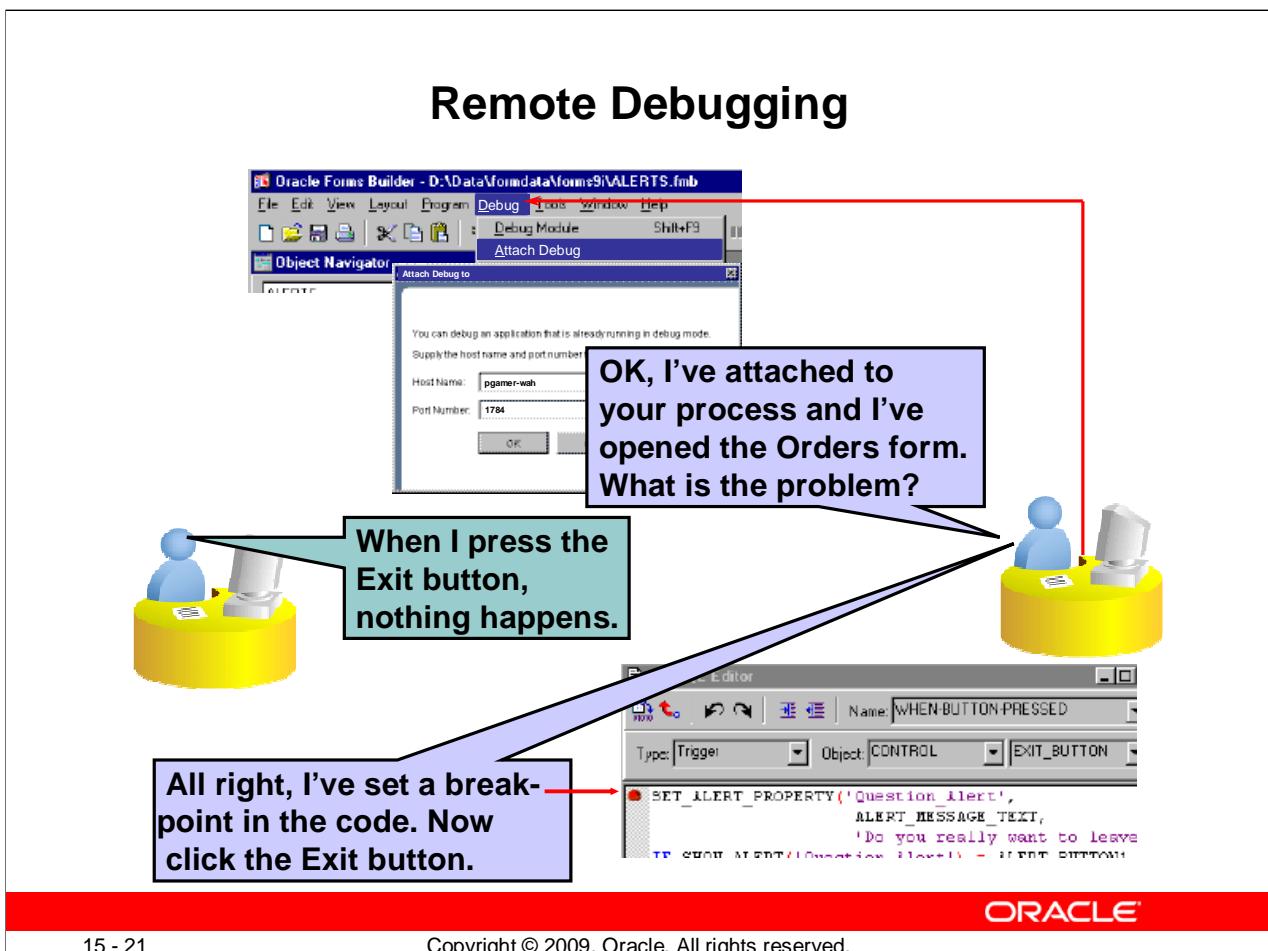
Remote Debugging

You can also debug an application that runs remotely within the intranet. The application that is being debugged makes a call to the DEBUG . ATTACH built-in procedure. This displays a dialog box containing host and port information, which the developer can then use from Forms Builder to attach to the problem session in order to debug it. A direct socket connection is needed, so remote debugging is usually not possible over the Internet unless the firewall allows a direct socket connection.

This ability provides developers with a great head start in resolving problems. Rather than having to obtain a list of instructions from an end user on how to reproduce the problem, they can watch the problem happen as the user carries out the actions. If you want to enable remote debugging, consider adding a hook into DEBUG . ATTACH within all your application screens for activation by the user as required.

In the example illustrated in the slide, the application developer has created a button labeled Remote Debug that calls DEBUG . ATTACH. With such a button or menu option, the user can find out the host and port to convey that information to the developer who is debugging the problem.

Remote Debugging



Remote Debugging (continued)

With information about the host and port of the running application, the developer can attach to that process by invoking the Debug > Attach menu option in Forms Builder, as shown in the top screenshot. This invokes a window in which to enter the host and port. After it is successfully attached, the developer can open the .fmb file from which the running .fmx was generated and set any breakpoints in the code that will assist in debugging the problem described by the user.

In the slide example, the user describes a nonfunctioning Exit button. The developer opens the When-Button-Pressed trigger for that button, whose editor is shown in the screenshot at the bottom of the slide, sets a breakpoint in the first line of code, then instructs the user to click Exit to invoke the code.

When the process encounters a breakpoint in the code, the control passes to the Debugger. This can cause confusion for the user because the browser's applet window may go blank, just as it does when running in debug mode to test a form in Forms Builder.

The developer can view the Debug Console and its panels and step through the code as needed to debug the problem. The developer does not see the running application. If user interaction is needed (such as responding to an alert), the form reappears in the client browser and the user regains control until the response is given, at which point the control passes back to the Debugger.

Summary

In this lesson, you should have learned that:

- The Debug Console consists of panes to view the call stack, program variables, a user-defined watch list, Form values, loaded PL/SQL packages, global and system variables, and breakpoints
- You use the Run Debug button to run a form module in debug mode within Forms Builder
- You can set breakpoints in the PL/SQL Editor by double-clicking to the left of an executable line of code
- The debug buttons on the Forms Builder toolbar enable you to step through code in various ways
- Remote debugging enables users and developers to work together to debug a running application

ORACLE®

15 - 22

Copyright © 2009, Oracle. All rights reserved.

Summary

To debug a form from within Forms Builder, you can:

- Click Run Form Debug (compiles and runs the form automatically)
- Set breakpoints in the code
- Use various panes of the Debug Console to view aspects of the running form
- Use the debug-related toolbar buttons to step through the code

Enable remote debugging of a running form by calling DEBUG.ATTACH.

Practice 15: Overview

This practice covers the following topics:

- Running a form in debug mode from Forms Builder
- Setting breakpoints
- Stepping through code
- Viewing variable values while the form is running



Practice 15: Overview

In this practice, you run a form in debug mode from within Forms Builder, set a breakpoint, and step through code, looking at variable values as the form runs.

Adding Functionality to Items

16

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

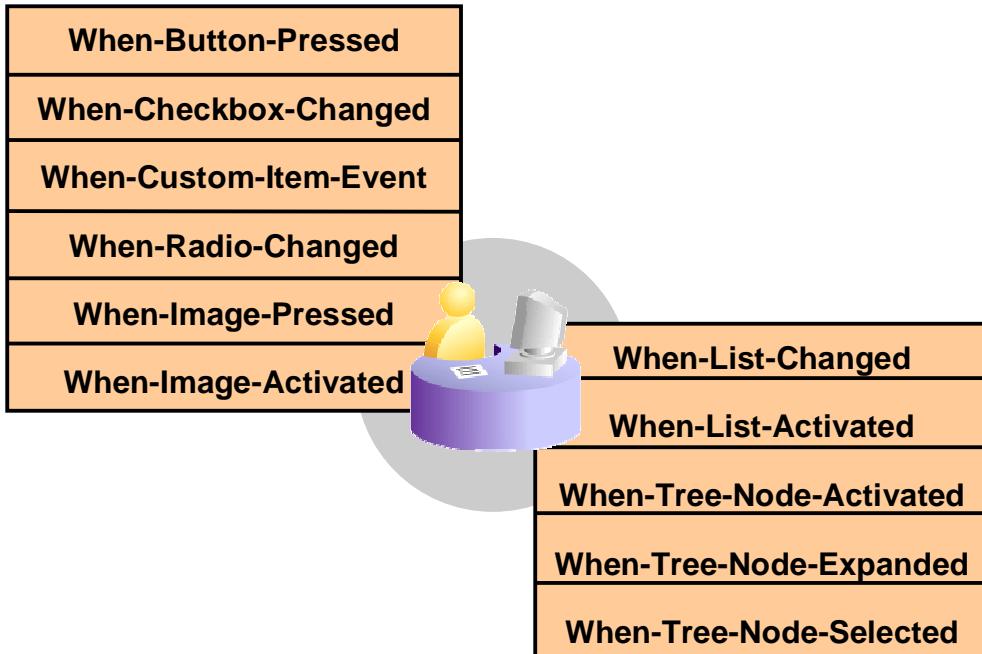
- Supplement the functionality of input items by using triggers and built-ins
- Supplement the functionality of noninput items by using triggers and built-ins



Lesson Aim

In this lesson, you learn how to use triggers to provide additional functionality to graphical user interface (GUI) items in Forms applications.

Item Interaction Triggers



Item Interaction Triggers

The graphic in the slide shows a user interacting with a form. There are several types of GUI items that the user can interact with by using the mouse or by pressing a function key. Most of these items have default functionality. For example, by selecting an option button, the user can change the value of the radio group item.

You will often want to add triggers to provide customized functionality when these events occur. For example:

- Performing tests and appropriate actions as soon as the user selects an option button, a list, or a check box
- Conveniently displaying an image when the user clicks an image item
- Defining the functionality of a push button (which has none until you define it)

Item Interaction Triggers (continued)

The following triggers fire due to user interaction with an item. They can be defined at any scope.

Trigger	Firing Event
When-Button-Pressed	User clicks or uses the function key to select
When-Checkbox-Changed	User changes the check box state by selecting or by pressing a function key
When-Custom-Item-Event	User selects or changes the value of a JavaBean component
When-Radio-Changed	User selects a different option, or deselects the current option, in a radio group
When-Image-Pressed	User clicks image item
When-Image-Activated	User double-clicks image item
When-List-Changed	User changes the value of a list item
When-List-Activated	User double-clicks element in a Tlist
When-Tree-Node-Activated	User double-clicks a node or presses Enter when a node is selected
When-Tree-Node-Expanded	User expands or collapses a node
When-Tree-Node-Selected	User selects or deselects a node

Coding Item Interaction Triggers

- Valid commands:
 - SELECT statements
 - Standard PL/SQL constructs
 - All built-in subprograms
- Do not fire during:
 - Navigation
 - Validation (use When-Validate-“object” to code actions to take place during validation)

ORACLE®

16 - 5

Copyright © 2009, Oracle. All rights reserved.

Coding Item Interaction Triggers

You can use standard SQL and PL/SQL statements in these triggers, as in the example on the next page. However, you would often want to add functionality to items by calling built-in subprograms, which provide a wide variety of mechanisms.

Although Forms allows you to use data manipulation language (DML) (INSERT, UPDATE, or DELETE) statements in any trigger, it is best to use them in commit triggers only. Otherwise, the DML statements are not included in the administration kept by Forms concerning commit processing. This may lead to unexpected and unwanted results. You learn about commit triggers in the lesson titled “Transaction Processing.”

Note: During an unhandled exception, the trigger terminates and sends the Unhandled Exception message to the operator. The item interaction triggers do not fire on navigation or validation events.

Coding Item Interaction Triggers (continued)

Example of When-Radio-Changed

The When-Radio-Changed trigger on CUSTOMERS.Credit_Limit: When the user changes the credit limit, this trigger immediately confirms whether the customer has outstanding orders exceeding the new credit limit. If so, a message warns the user.

```
DECLARE
    n NUMBER;
    v_unpaid_orders NUMBER;
BEGIN
    SELECT SUM(nvl(unit_price,0)*nvl(quantity,0))
        INTO v_unpaid_orders
        FROM orders o, order_items i
        WHERE o.customer_id = :customers.customer_id
        AND o.order_id = i.order_id
    -- Unpaid credit orders have status between 4 and 9
    AND (o.order_status > 3 AND o.order_status < 10);
    IF v_unpaid_orders > :customers.credit_limit THEN
        n := SHOW_ALERT('credit_limit_alert');
    END IF;
END;
```

Note: Displaying alerts is discussed in the lesson titled “Run-Time Messages and Alerts.”

Interacting with Check Boxes

The screenshot shows a user interface for searching customers. It includes input fields for 'First Name' containing 'Constantin' and 'Last Name'. Below these is a checkbox labeled 'Perform case-sensitive query on name?' which is checked. This visual context illustrates the 'checkbox' being interacted with.

When-Checkbox-Changed

```
IF CHECKBOX_CHECKED('control.case_sensitive') THEN
    SET_ITEM_PROPERTY('customers.cust_first_name',
        CASE_INSENSITIVE_QUERY, PROPERTY_FALSE);
    SET_ITEM_PROPERTY('customers.cust_last_name',
        CASE_INSENSITIVE_QUERY, PROPERTY_FALSE);
ELSE
    SET_ITEM_PROPERTY('customers.cust_first_name',
        CASE_INSENSITIVE_QUERY, PROPERTY_TRUE);
    SET_ITEM_PROPERTY('customers.cust_last_name',
        CASE_INSENSITIVE_QUERY, PROPERTY_TRUE);
END IF;
```

ORACLE®

16 - 7

Copyright © 2009, Oracle. All rights reserved.

Interacting with Check Boxes

You have already seen an example of adding functionality to radio groups; you now look at adding functionality to other items that accept user input.

Check Boxes

When the user selects or deselects a check box, such as the “Perform case-sensitive query on name?” check box shown in the slide screenshot, the associated value for the state is set. You may want to perform trigger actions based on this change. Note that the CHECKBOX_CHECKED function enables you to test the state of a check box without needing to know the associated values for the item.

Example

The When-Checkbox-Changed trigger (shown in the slide) on the CONTROL.Case_Sensitive item specifies that the query is not case-sensitive if the box is not selected.

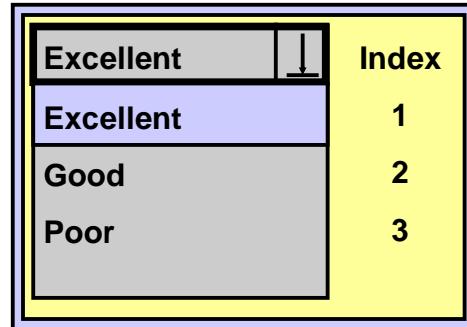
Changing List Items at Run Time

Triggers:

- When-List-Changed
- When-List-Activated

Built-ins:

- ADD_LIST_ELEMENT
- DELETE_LIST_ELEMENT



ORACLE®

16 - 8

Copyright © 2009, Oracle. All rights reserved.

Changing List Items at Run Time

You can use the When-List-Changed trigger to trap user selection of a list value. For Tlists, you can trap double-clicks with When-List-Activated.

With Forms Builder, you can also change the selectable elements in a list as follows:

- Periodically update the list from a two-column record group.
- Add or remove individual list elements through the ADD_LIST_ELEMENT and DELETE_LIST_ELEMENT built-ins, respectively:

```
ADD_LIST_ELEMENT('list_item_name', index, 'label',
    'value');

DELETE_LIST_ELEMENT('list_item_name', index);
```

Parameter	Description
Index	Number identifying the element position in the list (top is 1)
Label	The name of the element
Value	The new value for the element

Note: You can eliminate the Null list element of a list by changing Required to Yes.

At run time, when the block contains queried or changed records, Forms may not allow you to add or delete elements from a list item.

Using Dynamic LOVs

- You can use one list of values (LOV) for multiple items
- You can associate an LOV with the item at run time:

```
SET_ITEM_PROPERTY(LOV_NAME, 'my_lov') ;  
  
• You can use a dynamic record group:  
rg_id := POPULATE_GROUP_WITH_QUERY('my_rg',  
'SELECT ...');  
SET_LOV_PROPERTY('my_lov', GROUP_NAME,  
'my_rg');
```



Using Dynamic LOVs

LOVs can be made more flexible by assigning them to items at run time rather than at design time. You can use the SET_ITEM_PROPERTY built-in to accomplish this.

You can use a single LOV for multiple items by modifying the query of its underlying record group at run time by using the POPULATE_GROUP_WITH_QUERY built-in. For example, a Key-Listval trigger for an item can contain the following code:

```
DECLARE rg_id NUMBER;  
BEGIN  
    SET_LOV_PROPERTY('emp_lov',TITLE,'Sales Reps');  
    rg_id := POPULATE_GROUP_WITH_QUERY('EMP_RG','SELECT  
        employee_id ID, first_name || ' ' || last_name NAME  
        FROM employees WHERE job_id = ''SA_REP''');  
    SYNCHRONIZE;  
    LIST_VALUES;  
END;
```

Displaying LOVs from Buttons

- **Uses:**
 - Convenient alternative for accessing LOVs
 - Can display independently of text items
- **Needs:**
 - The When-Button-Pressed trigger
 - The LIST_VALUES or SHOW_LOV built-in

ORACLE®

16 - 10

Copyright © 2009, Oracle. All rights reserved.

Displaying LOVs from Buttons

If you have attached an LOV to a text item, the user can invoke the LOV from the text item by selecting Edit > Display List or by pressing the List Values key.

However, it is always useful if a button is available to display an LOV. The button has two advantages:

- It is a convenient alternative for accessing the LOV.
- It displays an LOV independently of a text item (using SHOW_LOV).

There are two built-ins that you can call to invoke an LOV from a trigger. These are LIST_VALUES and SHOW_LOV.

LIST_VALUES Procedure

This built-in procedure invokes the LOV that is attached to the current text item in the form. It has an optional argument, which may be set to RESTRICT, meaning that the current value of the text item is used as the initial search string on the LOV. The default for this argument is NO_RESTRICT.

Displaying LOVs from Buttons (continued)

SHOW_LOV Function

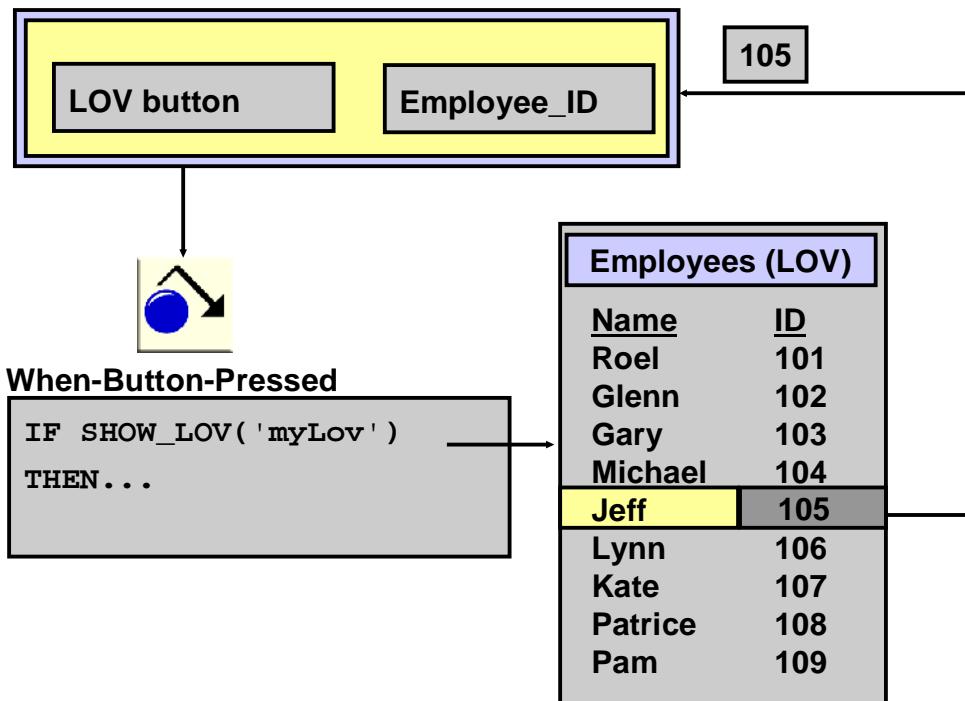
This built-in function, without arguments, invokes the LOV of the current item. However, there are arguments that you can use to define which LOV is to be displayed, what the x and y coordinates are, and where its window should appear:

```
SHOW_LOV('lov_name', x, y)  
SHOW_LOV(lov_id, x, y)
```

You should note that either the LOV name (in quotation marks) or the LOV ID (without quotation marks) can be supplied in the first argument.

Note: `lov_id` is a PL/SQL variable where the internal ID of the object is stored. Internal IDs are a more efficient way of identifying an object.

Using the SHOW_LOV Function



16 - 12

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Using the SHOW_LOV Function

The SHOW_LOV function returns a Boolean value:

- TRUE indicates that the user selected a record from the LOV.
- FALSE indicates that the user dismissed the LOV without choosing a record, or that the LOV returned 0 records from its Record Group.

Note

- You can use the FORM_SUCCESS function to differentiate between the two causes of SHOW_LOV returning FALSE.
- Create the LOV button with a suitable label, such as "Pick," and arrange it on the canvas where the user intuitively associates it with the items that the LOV supports (even though the button has no direct connection with text items). This is usually adjacent to the main text item to which the LOV returns a value.
- You can use the SHOW_LOV function to display an LOV that is not even attached to a text item if you identify the LOV in the first argument of the function. When called from a button, this invokes the LOV to be independent of cursor location. This is shown in the example in the slide, where a button calls a trigger that invokes the LOV. The LOV returns a value to Employee_Id, but the LOV is not attached to the Employee_Id item.

Using the SHOW_LOV Function (continued)

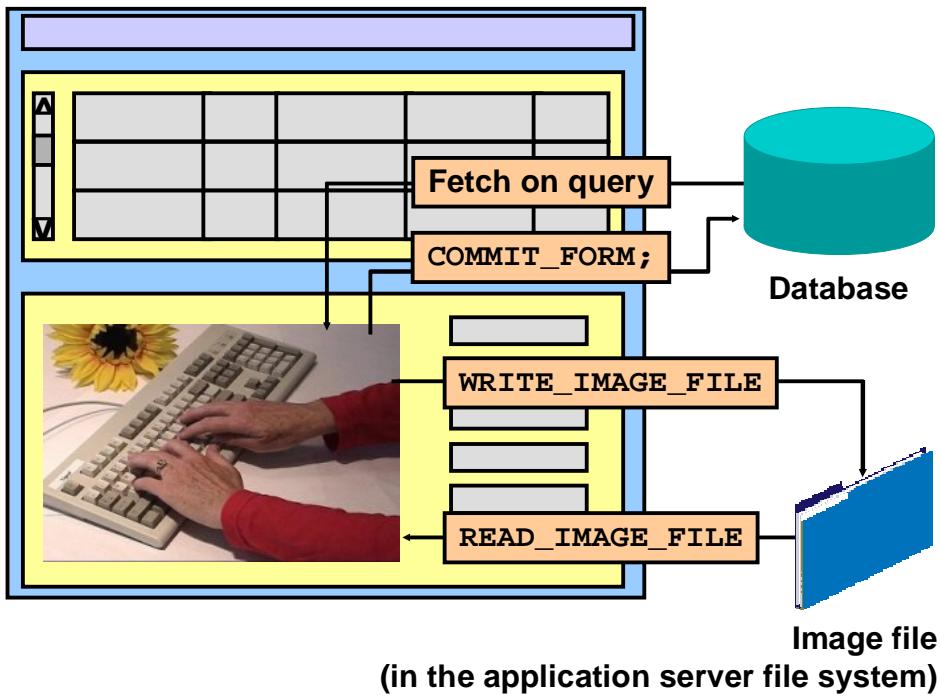
- Switch off the button's Mouse Navigate property. When using LIST_VALUES, the cursor must reside in the text item that is attached to the LOV. With SHOW_LOV, this also maintains the cursor in its original location after the LOV is closed, wherever that may be.

Example: Forcing the User to Select a Value from the LOV

A When-Button-Pressed trigger on Customer_Lov_Button invokes an LOV in a PL/SQL loop, until the function returns TRUE. Because SHOW_LOV returns TRUE when the user selects a record, the LOV continues to redisplay if the user does not select a record.

```
LOOP
    EXIT WHEN SHOW_LOV('customer_lov');
    MESSAGE('You must select a value from list');
END LOOP;
```

Populating Image Items



Populating Image Items

Image items that have the Database Item property set to Yes automatically populate in response to a query in the owning block (from a LONG RAW or BLOB column in the base table).

Nonbase table image items, however, need to be populated by other means. For example, the READ_IMAGE_FILE built-in procedure loads the image from the file system.

If you load an image into a base-table image item by using READ_IMAGE_FILE, its contents are committed to the database LONG RAW or BLOB column when you save the changes in the form. You can use this technique to populate a table with images.

There are two triggers that fire when the user interacts with an image item directly:

- When-Image-Pressed (fires for a click on an image item)
- When-Image-Activated (fires for a double-click on an image item)

The graphic shows an image item populated initially by a database query. The WRITE_IMAGE_FILE built-in is then used to write the image file to the file system. After the image is in the file system, READ_IMAGE_FILE can be used to repopulate the image item, possibly with a different image. In the example, the image item is a base table item, so any changed image is written to the database when the record is saved.

Populating Image Items (continued)

READ_IMAGE_FILE Procedure

You can use this built-in procedure to load an image file in a variety of formats, into an image item: `READ_IMAGE_FILE('filename', 'filetype', 'item_name');`

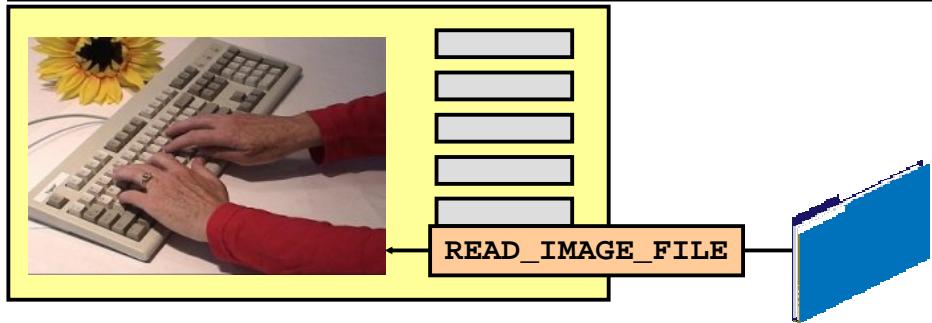
Parameter	Description
<code>filename</code>	Image file name (without a specified path, default path is assumed)
<code>filetype</code>	File type of the image (You can use ANY as a value, but it is recommended to set a specific file type for better performance. Refer to online Help for file types.)
<code>item_name</code>	Name of the image item (a variable holding Item_id is also valid for this argument.) (This parameter is optional.)

Note

- The `READ_IMAGE_FILE` built-in procedure loads an image file from the application server file system. If you need to load an image file from the file system on the client, use a JavaBean.
- The `filetype` parameter is optional in `READ_IMAGE_FILE`. If you omit `filetype`, you must explicitly identify the `item_name` parameter.
- The `WRITE_IMAGE_FILE` built-in procedure writes an image file to the application server file system. If you need to write an image file to the file system on the client, use a JavaBean.

Image Example

```
READ_IMAGE_FILE  
(TO_CHAR(:order_items.product_id)||'.JPG',  
'JPEG', 'order_items.product_image');
```



ORACLE®

16 - 16

Copyright © 2009, Oracle. All rights reserved.

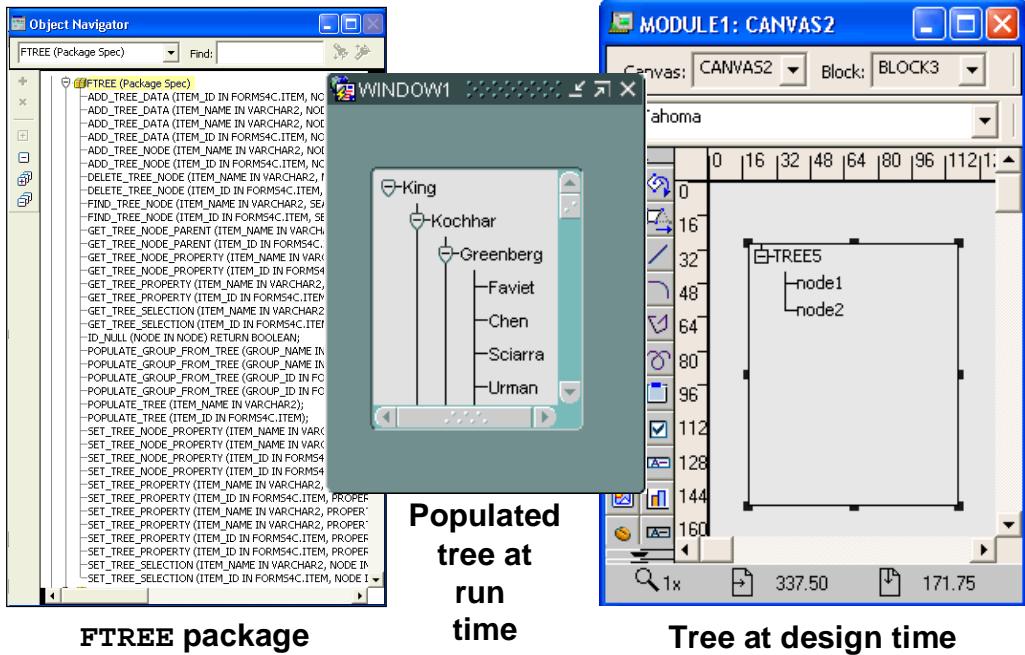
Image Example

The following When-Image-Pressed trigger on the Product_Image item displays a picture of the current product (in the ITEM block) when the user clicks the image item. This example assumes that the related file names have the format *<product_id>.jpg*.

```
READ_IMAGE_FILE(TO_CHAR(:order_items.product_id)||'.jpg',  
'JPEG', 'order_items.product_image');
```

Notice that the first argument to this built-in is data type CHAR. Because Product_Id is a NUMBER item, it must be first converted by using the TO_CHAR function.

Displaying Hierarchical Trees



Displaying Hierarchical Trees

The hierarchical tree displays data in the form of a standard navigator, similar to the Object Navigator used in Oracle Forms Builder. The middle and right screenshots show the appearance of a hierarchical tree at run time and at design time.

You can populate a hierarchical tree with values contained in a record group or query text. At run time, you can programmatically add, remove, modify, or evaluate elements in a hierarchical tree. You can also use the Property Palette to set the populate properties of the hierarchical tree, but you must still programmatically populate the tree at run time.

The FTREE package contains built-ins and constants to interact with hierarchical tree items in a form. To utilize the built-ins and constants, you must precede their names with the name of the package.

You can add data to a tree view by:

- Populating a tree with values contained in a record group or query by using the POPULATE_TREE built-in
- Adding data to a tree under a specific node by using the ADD_TREE_DATA built-in
- Modifying elements in a tree at run time by using built-in subprograms, such as SET_TREE_NODE_PROPERTY
- Adding or deleting nodes and the data elements under the nodes with ADD_TREE_NODE or DELETE_TREE_NODE

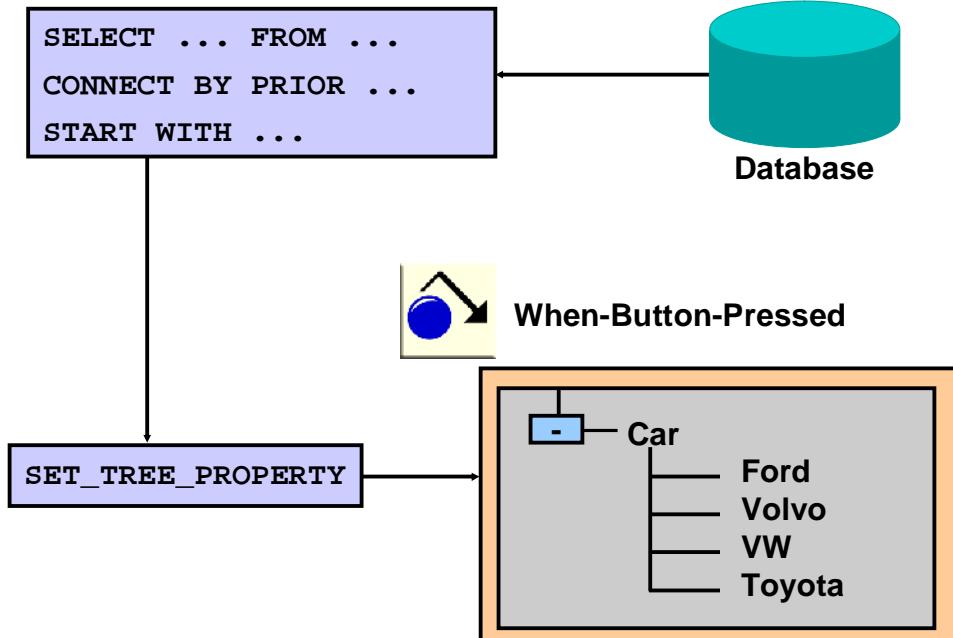
Displaying Hierarchical Trees (continued)

SET_TREE_PROPERTY Procedure

This built-in procedure can be used to change certain properties for the indicated hierarchical tree item. It can also be used to populate the indicated hierarchical tree item from a record group: FTREE.SET_TREE_PROPERTY(item_name, Ftree.property, value);.

Parameter	Description
item_name	Specifies the name of the object created at design time. The data type of the name is VARCHAR2. A variable holding Item_id is also valid for this argument.
property	Specifies one of the following properties: RECORD_GROUP: Replaces the data set of the hierarchical tree with a record group and causes it to display QUERY_TEXT: Replaces the data set of the hierarchical tree with a SQL query and causes it to display ALLOW_EMPTY_BRANCHES: Possible values are PROPERTY_TRUE and PROPERTY_FALSE
value	Specifies the value appropriate to the property you are setting

Populating Hierarchical Trees with Queries



Populating Hierarchical Trees with Queries

To use a query to populate a hierarchical tree, as illustrated by the slide graphics, perform the following steps:

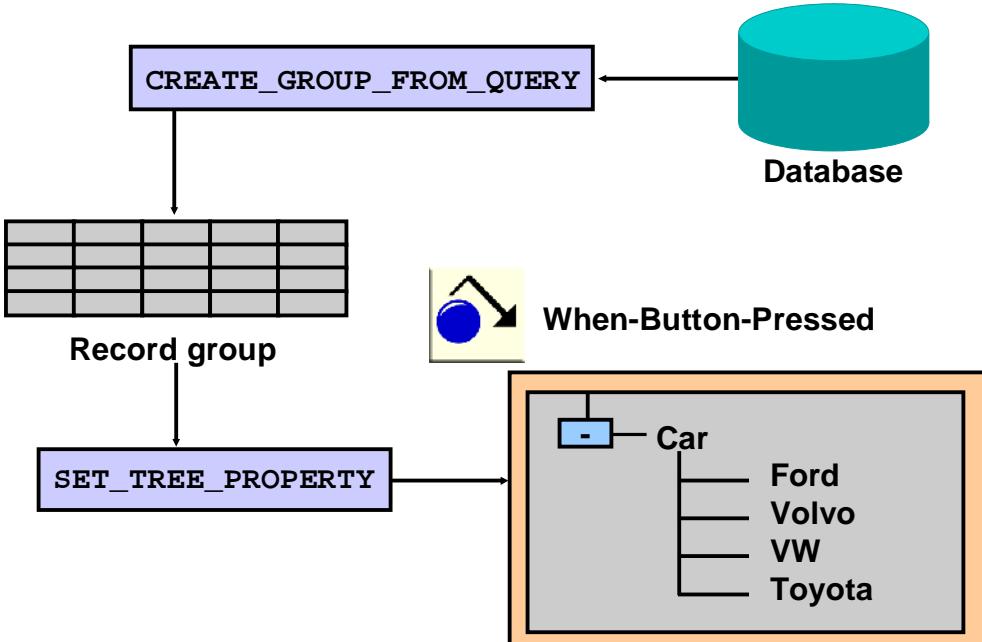
1. Set the Data Query property of the hierarchical tree item, either at design time or programmatically with `SET_TREE_PROPERTY`. If you do this programmatically, this also executes the query and displays the query results in the tree item.
2. If you set the Data Query property at design time, use `POPULATE_TREE` to execute the query and populate the tree with the query results.

Example

This code can be used in a When-Button-Pressed or When-New-Form-Instance trigger to initially populate the hierarchical tree with data.

```
DECLARE htree ITEM;
v_qt VARCHAR2(200) := 'SELECT 1, LEVEL, last_name, NULL,
    TO_CHAR(employee_id) FROM employees CONNECT BY PRIOR
    employee_id = manager_id START WITH job_id = ''AD_PRES'''';
BEGIN
    htree := FIND_ITEM('employee_block.emp_tree');
    FTREE.SET_TREE_PROPERTY(htree, Ftree.QUERY_TEXT,v_qt);
END;
```

Populating Hierarchical Trees with Record Groups



Populating Hierarchical Trees with Record Groups

To use a record group to populate a hierarchical tree, perform the following steps:

1. Create the record group, either at design time or programmatically, as illustrated in the slide graphic.
2. Populate the record group at run time with POPULATE_GROUP.
3. Populate and display the hierarchical tree data:
 - Use POPULATE_TREE if you have already set the Record Group property of the hierarchical tree item, either in the Property Palette or programmatically.
 - Use SET_TREE_PROPERTY as illustrated in the slide graphics if you have not set the Record Group property of the tree; this also populates and displays the tree data if the populated record group exists.

Populating Hierarchical Trees with Record Groups (continued)

Example

This code can be used in a When-Button-Pressed or When-New-Form-Instance trigger to initially populate the hierarchical tree with data. The example locates the hierarchical tree first. Then, a record group is created and the hierarchical tree is populated.

```
DECLARE
    htree ITEM;
    v_ignore NUMBER;
    rg_emps RECORDGROUP;
BEGIN
    htree := FIND_ITEM('tree_block.htree3');
    rg_emps := CREATE_GROUP_FROM_QUERY('rg_emps',
        'select 1, LEVEL, last_name, NULL, TO_CHAR(employee_id) '
        || ' from employees ' ||
        'CONNECT BY PRIOR employee_id = manager_id ' ||
        'START WITH job_id = ''AD_PRES'''');
    v_ignore := POPULATE_GROUP(rg_emps);
    FTREE.SET_TREE_PROPERTY(htree,FTREE.RECORD_GROUP,rg_emps);
END;
```

Using Hierarchical Queries

When-Button-Pressed

```
rg_emps := CREATE_GROUP_FROM_QUERY('rg_emps',
  'SELECT 1, LEVEL, last_name, NULL,
  TO_CHAR(employee_id) ' ||
  'from employees ' ||
  'CONNECT BY PRIOR employee_id = manager_id ' ||
  'START WITH job_id = ''AD_PRES'''');

v_ignore := POPULATE_GROUP(rg_emps);

FTREE.SET_TREE_PROPERTY('block4.tree5',
  FTREE.RECORD_GROUP, rg_emps);
```

ORACLE®

16 - 22

Copyright © 2009, Oracle. All rights reserved.

Using Hierarchical Queries

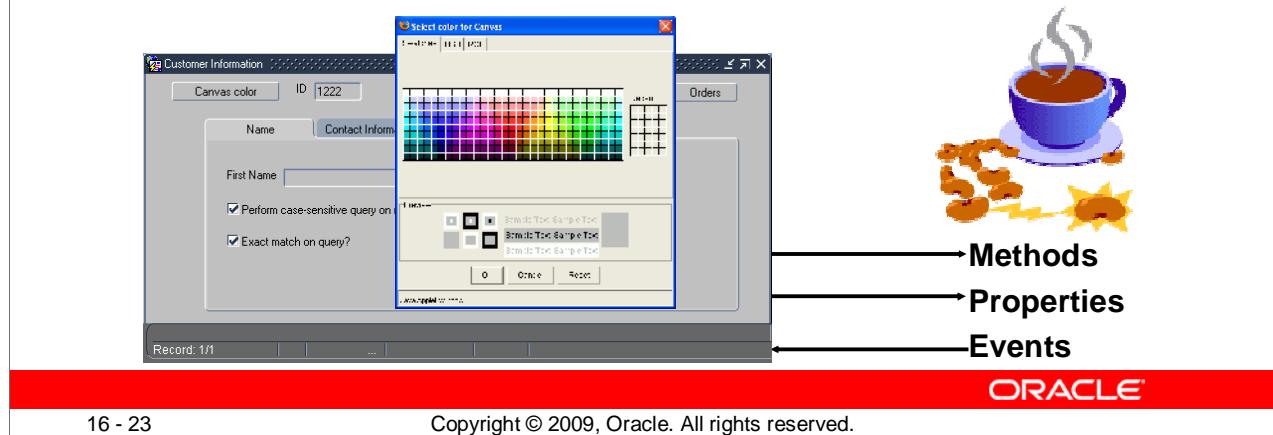
The columns in a record group or query that are used to populate a hierarchical tree are the following:

- **Initial state:** 0 (not expandable), 1 (expanded), or -1 (collapsed)
- **Node tree depth:** The LEVEL pseudocolumn
- **Label for the node:** What the user sees
- **Icon for the node:** Picture displayed, if any
- **Data:** The actual value of the node

The query itself uses standard SQL syntax for a hierarchical query, with a CONNECT BY clause to define the parent/child relationship and a START WITH clause to identify the root of the hierarchy.

Interacting with JavaBeans

- Tell Forms about the bean: Register
- Communication from Forms to JavaBean:
 - Invoke methods
 - Get/Set properties
- Communication from JavaBean to Forms: Events



16 - 23

Copyright © 2009, Oracle. All rights reserved.

Interacting with JavaBeans

In the lesson titled “Creating Noninput Items,” you learned how to add a JavaBean to a form using the Bean Area item. The bean that you add to a form may have a visible component on the form itself, such as a Calendar bean that has its own button to invoke the bean. However, JavaBeans (such as the ColorPicker bean shown in the slide) do not always have visible components, so you may need to create a button or some other way to invoke it.

Regardless of whether the bean is visible in the bean area, there must be some communication between the run-time form and the Java classes that comprise the bean. First, the form must be made aware of the bean, either by setting its Implementation Class property at design time or by registering the bean and its events at run time. When the form knows about the bean, it communicates to the bean by:

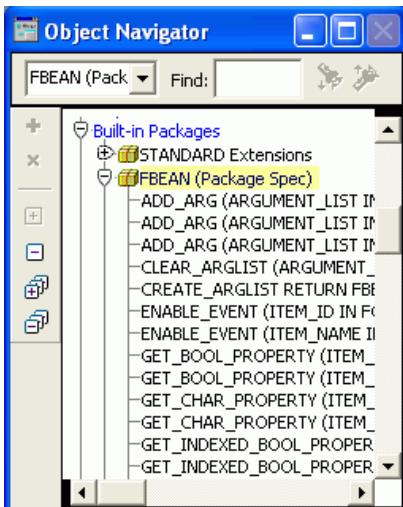
- Invoking the methods of the bean
- Getting and setting properties of the bean

The bean communicates to the form by:

- Sending an event, such as the fact that the user selected a date or color
- Sending a list containing information needed by the form, such as what date or color was selected
- Returning a value from an invoked method

Interacting with JavaBeans

The FBEAN package provides built-ins to:



- Register the bean
- Invoke methods of the bean
- Get and set properties on the bean
- Subscribe to bean events

ORACLE®

Interacting with JavaBeans (continued)

The FBEAN package, shown expanded in the Object Navigator in the slide, contains Forms built-ins that enable you to code interactions with JavaBeans in PL/SQL, thereby, eliminating the need to know Java in order to communicate with the bean.

Many of the built-ins take some of the same arguments:

- **Item Name or Item ID (obtained with the FIND_ITEM built-in):** The first argument for most of the FBEAN built-ins, referred to on the next page simply as ITEM
- **Item Instance:** A reference to which instance of the item should contain the bean. This is applicable where Bean Area is part of a multirow block and more than one instance of Bean Area is displayed. This is referred to on the next page as INSTANCE. You can use the ALL_ROWS (or FBEAN.ALL_ROWS) value for the Item Instance value to indicate that the command should apply to all of the instances of this Bean Area in the block.

Note: This refers to the UI instance of Bean Area, not the row number in the block. For example, in a block with 5 rows displayed and 100 rows queried, there will be 5 instances of the bean numbered 1 through 5, not 100 instances.

- **Value:** Can accept BOOLEAN, VARCHAR2, or NUMBER data types

Interacting with JavaBeans (continued)

FBEAN package (continued)

Some of the built-ins in the FBEAN package are:

- **GET_PROPERTY(ITEM , INSTANCE , PROPERTY_NAME)** (returns VARCHAR2):
Is a function that retrieves the value of the specified property
- **SET_PROPERTY(ITEM , INSTANCE , PROPERTY_NAME , VALUE)**:
Sets the specified property of the bean to the value indicated
- **INVOKE(ITEM , INSTANCE , METHOD_NAME [, ARGUMENTS])**:
Invokes a method on the bean, optionally passing arguments to the method
- **REGISTER_BEAN(ITEM , INSTANCE , BEAN_CLASS)**:
Registers the bean with the form at run time, making all its exposed attributes and methods available for the form's bean item (The last argument is the full class name of the bean, such as 'oracle.forms.demos.beans.ColorPicker').
- **ENABLE_EVENT(ITEM , INSTANCE , EVENT_LISTENER_NAME , SUBSCRIBE)**:
Is a BOOLEAN indicating whether to subscribe (TRUE) or unsubscribe (FALSE) to the event

Remember to precede calls to any of these built-ins with the package name and a dot, such as FBEAN.GET_PROPERTY(...). You can pass arguments to these built-ins as either a delimited string or as an argument list.

Deploying the Bean

Because the bean itself is a Java class or set of Java class files separate from the form module, you need to know where to put these files. You can locate these:

- On the middle-tier server, either in the directory structure referenced by the form applet's CODEBASE parameter or in the server's CLASSPATH. CODEBASE is by default the forms\java subdirectory of ORACLE_HOME.
- If using JInitiator, in a Java Archive (JAR) file in the middle-tier server's CODEBASE directory, and included in the ARCHIVE parameter so that the JAR file is downloaded to and cached on the client. For example:

```
archive_jini=frmall_jinit.jar,colorpicker.jar
```

(The CODEBASE and ARCHIVE parameters are set in the formsweb.cfg file.)

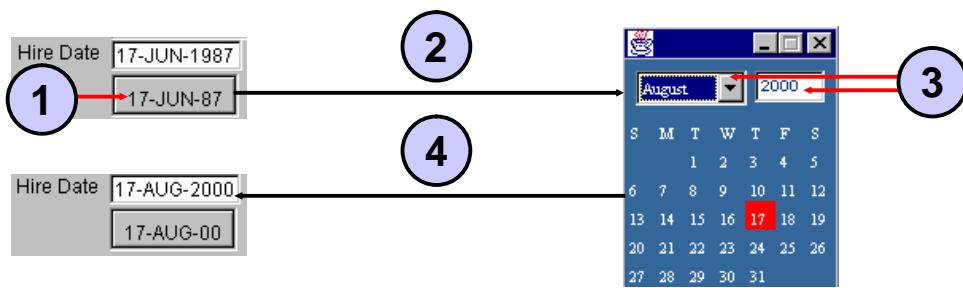
Interacting with JavaBeans

- Register a listener for the event:

```
FBEAN.ENABLE_EVENT( 'MyBeanArea' ,1 , 'mouseListener' ,  
true );
```

- When an event occurs on the bean:

- The When-Custom-Item-Event trigger fires
- The name and information are sent to Forms in:
 - SYSTEM.custom_item_event
 - SYSTEM.custom_item_event_parameters



ORACLE®

Interacting with JavaBeans (continued)

Responding to Events

When a user interacts with a JavaBean at run time, it usually causes an event to occur. You can use FBEAN.ENABLE_EVENT to register a listener for the event, so that when the event occurs, Forms will fire the When-Custom-Item-Event trigger. In this trigger, you can code a response to the event. The SYSTEM.custom_item_event and SYSTEM.custom_event_parameters variables contain the name of the event and information the bean is sending to the form.

A typical interaction that can occur includes the following steps:

1. The user clicks the bean area for a Calendar bean, as shown in the first screenshot in the slide. This bean area has a visible component on the form that looks like a button. The label is set to the hire date for an employee, which is 17-JUN-87 in the example.
2. The Calendar bean is invoked and displays a calendar initially set to the employee's hire date.
3. The user changes the date on the bean by picking a new month and year as in the screenshot at the right, and then clicking on a day, initiating the DateChanged event.
4. The When-Custom-Item-Event trigger obtains the changed date and assigns it back to the employee hire_date item, as shown in the screenshot at the bottom left. The label on the bean area "button" also changes to the new date.

Interacting with JavaBeans (continued)

Coding a When-Custom-Item-Event Trigger

To respond to JavaBeans events in a When-Custom-Item-Event trigger, you code the action that you want to take place when an event occurs.

For example, when a user selects a date from the Calendar bean, the DateChange event takes place and the When-Custom-Item-Event trigger fires. In the code for the When-Custom-Item-Event trigger on the Calendar bean area item, you need to obtain the name of the event. If it is the DateChange event, you must obtain the new date and assign it to a form item, such as the employee's hire date. You can use the system variables containing the event and parameter information:

```
declare
    hBeanEventDetails ParamList;
    eventName varchar2(80);
    paramType number;
    eventType varchar2(80);
    newDateVal varchar2(80);
    newDate date := null;
begin
    hBeanEventDetails := get_parameter_list
        (:SYSTEM.custom_item_event_parameters);
    eventName := :SYSTEM.custom_item_event;
    if(eventName = 'DateChange') then
        get_parameter_attr(hBeanEventDetails,
            'DateValue', ParamType, newDateVal);
        newDate := to_date(newDateVal, 'DD.MM.YYYY');
    end if;
    :employees.hire_date := newDate;
end;
```

The preceding example is for a bean that uses hand-coded integration. If you use the FBEAN package to integrate the bean, the name of the value passed back to the form is always called 'DATA'.

For example:

```
get_parameter_attr(:system.custom_item_event_parameters,
    'DATA', paramType, eventData);
```

(where paramType and eventData are PL/SQL variables you declare in the When-Custom-Item-Event trigger, such as paramType and newDateVal in the preceding example).

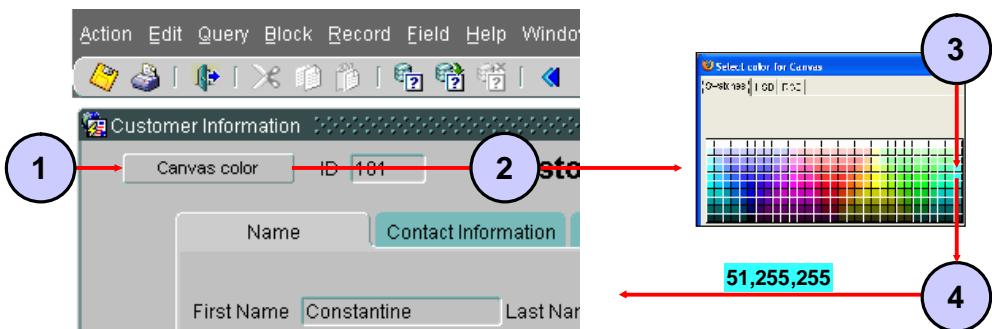
Note: There are many examples of JavaBeans in the Forms Demos that you can download from OTN:

http://otn.oracle.com/sample_code/products/forms/index.html

Interacting with JavaBeans

The JavaBean may:

- Not have a visible component
- Not communicate via events
- Return a value to the form when invoked (used in the same way as a function)



Interacting with JavaBeans (continued)

Getting Values from JavaBeans Without Events

Not all information from JavaBeans is obtained via events. For example, a JavaBean may return a value when one of its methods is invoked. This value may be assigned to a PL/SQL variable or Forms item, similar to the way a function returns a value.

An example of this is the ColorPicker bean that you added to the Customer form in the lesson titled “Creating Noninput Items.” It contains a single method that returns a value, and also has no visible component in the bean area of the form. To invoke the bean and obtain a value from it, you can use a Forms push button and trigger with code similar to the following:

```
vcNewColor := FBean.Invoke_char(hColorPicker,  
1,'showColorPicker','"Select color for canvas"');
```

The `INVOKE_CHAR` built-in is used to call a method that returns a `VARCHAR2` value.

1. The user clicks the button to invoke the bean, as illustrated by the left screenshot.
2. The Color Picker component appears, shown in the right screenshot.
3. The user selects a color.
4. The color value (RGB values in comma-separated list) is returned to the `vcNewColor` variable. The code can then use the color value to set the canvas color.

Summary

In this lesson, you should have learned that:

- You can use triggers to supplement the functionality of input and noninput items
- You can call useful built-ins from triggers:
 - CHECKBOX_CHECKED
 - [ADD | DELETE]_LIST_ELEMENT
 - SHOW_LOV
 - [READ | WRITE]_IMAGE_FILE
 - FTREE: POPULATE_TREE , ADD_TREE_DATA,
[GET | SET]_TREE_PROPERTY
 - FBEAN: [GET | SET]_PROPERTY , INVOKE ,
REGISTER_BEAN , ENABLE_EVENT



Summary

In this lesson, you should have learned to use triggers to provide functionality to the GUI items in form applications.

- The item interaction triggers accept SELECT statements and other standard PL/SQL constructs.

Triggers to supplement the functionality of input items include:

- When-[Checkbox | Radio]-Changed
- When-List-[Changed | Activated]

Triggers to supplement functionality of noninput items include:

- When-Button-Pressed
- When-Image-[Pressed | Activated]
- When-Tree-Node-[Activated | Expanded | Selected]
- When-Custom-Item-Event

There are built-ins for check boxes, LOV control, list item control, image file reading, hierarchical tree manipulation, interaction with JavaBeans, and so on.

Practice 16: Overview

This practice covers the following topics:

- Writing a trigger to check whether the customer's credit limit has been exceeded
- Creating a toolbar button to display and hide product images
- Coding a button to enable users to choose a canvas color for a form



Practice 16: Overview

In this practice, you create some additional functionality for a radio group. You also code interaction with a JavaBean. Finally, you add some triggers that enable interaction with buttons.

Run-Time Messages and Alerts

17

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

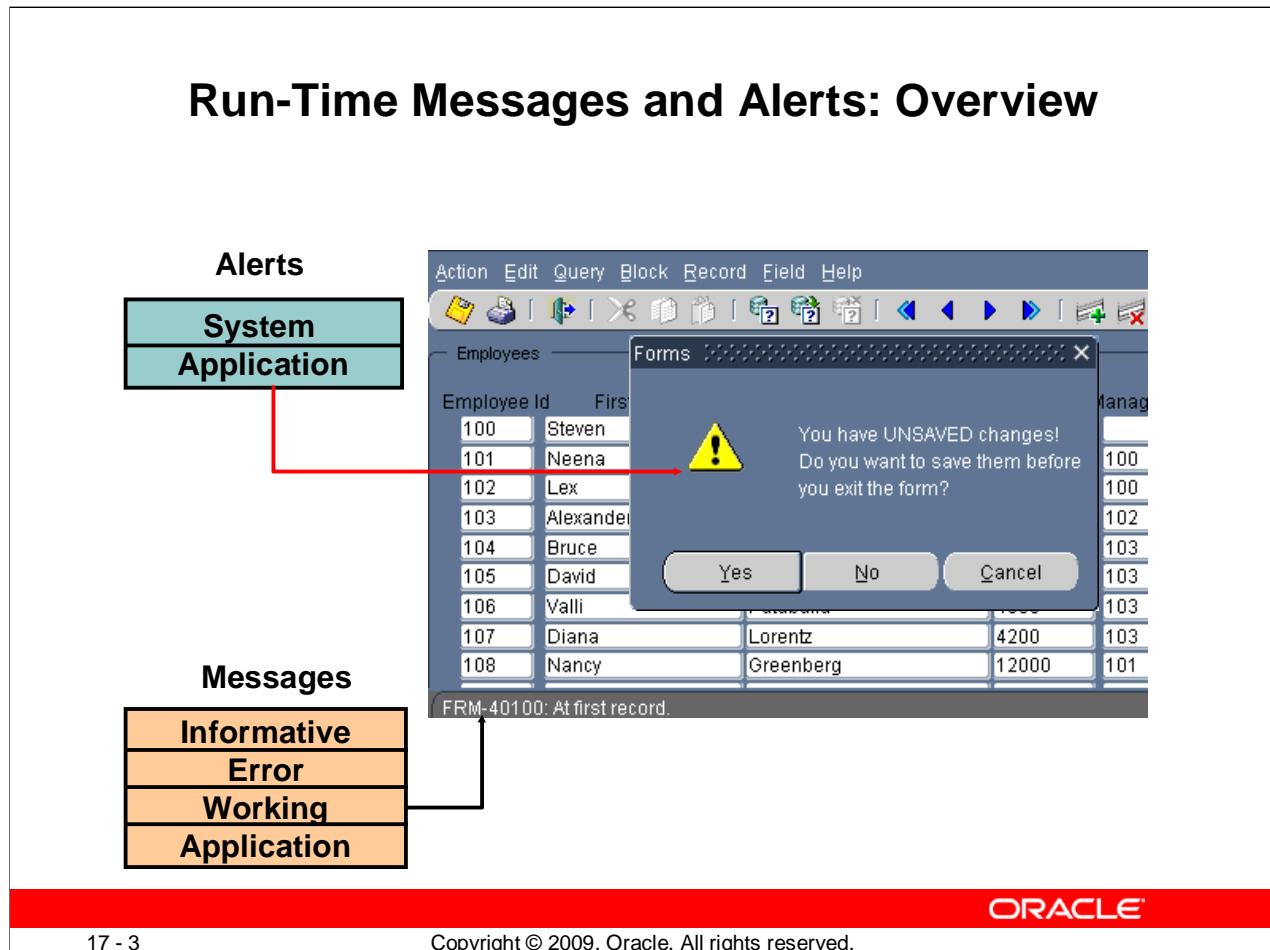
- Describe the default messaging behavior of a form
- Handle run-time failure of built-in subprograms
- Identify the different types of Forms messages
- Control system messages
- Create and control alerts
- Handle database server errors



Lesson Aim

This lesson shows you how to intercept system messages, and if desired, replace them with ones that are more suitable for your application. You will also learn how to handle errors by using built-in subprograms and how to build customized alerts for communicating with users.

Run-Time Messages and Alerts: Overview



Run-Time Messages and Alerts: Overview

Forms displays messages at run time to inform the operator of events that occur in the session. As the designer, you may want to either suppress or modify some of these messages, depending on the nature of the application.

Forms can communicate with the user in the following ways:

- **Informative message:** A message tells the user the current state of processing, or gives context-sensitive information. The default display is on the message line. You can suppress its appearance with an On-Message trigger.
- **Error message:** This informs the user of an error that prevents the current action. The default display is on the message line. You can suppress message line errors with an On-Error trigger.
- **Working message:** This tells the operator that the form is currently processing (for example, Working...). This is shown on the message line. This type of message can be suppressed by setting the suppress_working system variable to True:
 :SYSTEM.suppress_working := 'TRUE' ;

Run-Time Messages and Alerts: Overview (continued)

- **System alert:** Alerts give information to the operator that require either an acknowledgment or an answer to a question before processing can continue. This is displayed as a modal window. When more than one message is waiting to show on the message line, the current message is also displayed as an alert.

You can also build messages and alerts into your application:

- **Application message:** These are messages that you build into your application by using the MESSAGE built-in. The default display is on the message line.
- **Application alert:** These are alerts that you design as part of your application, and issue to the operator for a response by using the SHOW_ALERT built-in.

The screenshot shows two messages being displayed:

- The status line on the form displays an error message: FRM-40100: At first record
- An alert displays a message that the developer has substituted for the standard message: “Do you want to save the changes you have made?”
The new message is: “You have UNSAVED changes! Do you want to save them before you exit the form?” The alert has Yes, No, and Cancel buttons.

Built-ins and Handling Errors

- FORM_SUCCESS
 - TRUE: Action successful
 - FALSE: An error or fatal error occurred
- FORM_FAILURE
 - TRUE: A nonfatal error occurred
 - FALSE: Action successful or a fatal error occurred
- FORM_FATAL
 - TRUE: A fatal error occurred
 - FALSE: Action successful or a nonfatal error occurred

ORACLE®

17 - 5

Copyright © 2009, Oracle. All rights reserved.

Built-ins and Handling Errors

When a built-in subprogram fails, it does not directly cause an exception in the calling trigger or program unit. This means that subsequent code continues after a built-in fails, unless you take action to detect a failure.

Example

A button in the CONTROL block called Stock_Button is situated on the Toolbar canvas of the Orders form. When clicked, the When-Button-Pressed trigger navigates to the INVENTORIES block, and performs a query there:

```
GO_BLOCK( 'inventories' ) ;
EXECUTE_QUERY;
```

If the GO_BLOCK built-in procedure fails because the INVENTORIES block does not exist, or because it is nonenterable, the EXECUTE_QUERY procedure still executes, and attempts a query in the wrong block.

Built-ins and Handling Errors (continued)

Built-in Functions for Detecting Success and Failure

Forms Builder supplies some functions that indicate whether the latest action in the form was successful.

Built-In Function	Description of Returned Value
FORM_SUCCESS	TRUE: Action successful FALSE: Error or fatal error occurred
FORM_FAILURE	TRUE: A nonfatal error occurred FALSE: Either no error or a fatal error occurred
FORM_FATAL	TRUE: A fatal error occurred FALSE: Either no error or a nonfatal error occurred

Note: These built-in functions return success or failure of the latest action in the form. The failing action may occur in a trigger that fired as a result of a built-in from the first trigger. For example, the EXECUTE_QUERY procedure can cause a Pre-Query trigger to fire, which may itself fail.

Errors and Built-Ins

- Built-in failure does not cause an exception.
- Test built-in success with the FORM_SUCCESS function:
IF FORM_SUCCESS THEN . . .
or IF NOT FORM_SUCCESS THEN . . .
- What went wrong?
 - ERROR_CODE, ERROR_TEXT, ERROR_TYPE
 - MESSAGE_CODE, MESSAGE_TEXT, MESSAGE_TYPE

ORACLE®

17 - 7

Copyright © 2009, Oracle. All rights reserved.

Errors and Built-Ins

It is usually most practical to use FORM_SUCCESS because this returns FALSE if either a fatal or a nonfatal error occurs. You can then code the trigger to take appropriate action.

Example of FORM_SUCCESS

Here is the same trigger again. This time, the FORM_SUCCESS function is used in a condition to decide whether the query should be performed, depending on the success of the GO_BLOCK action.

```
GO_BLOCK( 'inventories' );
IF FORM_SUCCESS THEN
    EXECUTE_QUERY;
ELSE
    MESSAGE( 'An error occurred while navigating to
Stock' );
END IF;
```

Triggers fail only if there is an unhandled exception or you raise the FORM_TRIGGER_FAILURE exception to fail the trigger in a controlled manner.

Errors and Built-Ins (continued)

Note: The program unit CHECK_PACKAGE_FAILURE, which is written when you build master-detail blocks, may be called to fail a trigger if the last action was unsuccessful.

Built-in Functions to Determine the Error

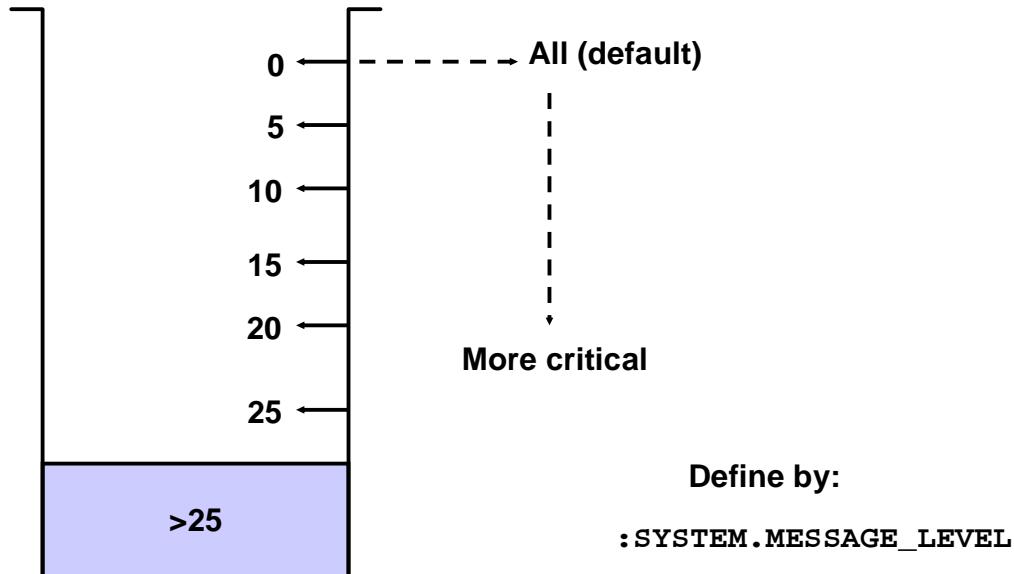
When you detect an error, you may need to identify it to take a specific action. Three more built-in functions provide this information:

Built-in Function	Description of Returned Value
ERROR_CODE	Error number (data type NUMBER)
ERROR_TEXT	Error description (data type CHAR)
ERROR_TYPE	FRM=Forms Builder error, ORA=Oracle error (data type CHAR)

These built-ins are explained in detail later in this lesson.

Controlling System Messages

Message Levels:



Controlling System Messages

You can prevent system messages from being issued, based on their severity level. Forms Builder classifies every message with a severity level that indicates how critical or trivial the information is; the higher the numbers, the more critical the message. There are six levels that you can affect.

Severity Level	Description
0	All messages
5	Reaffirms an obvious condition
10	User has made a procedural mistake
15	User attempting action for which the form is not designed
20	Cannot continue intended action due to a trigger problem or some other outstanding condition
25	A condition that could result in the form performing incorrectly
> 25	Messages that cannot be suppressed

The graphic depicts the concept that messages become more critical as their level number increases.

Controlling System Messages (continued)

Suppressing Messages According to Their Severity

In a trigger, you can specify that only messages above a specified severity level are to be issued by the form. You do this by assigning a value to the MESSAGE_LEVEL system variable. Forms then issues only those messages that are above the severity level defined in this variable.

The default value for MESSAGE_LEVEL (at form startup) is 0. This means that messages of all severities are displayed.

Suppressing Messages

```
:SYSTEM.message_level := '5';
UP;
IF NOT FORM_SUCCESS THEN
    MESSAGE('Already at the first Order');
END IF;
:SYSTEM.message_level := '0';
```

```
:SYSTEM.suppress_working := 'TRUE';
```



Suppressing Messages

Example

The following When-Button-Pressed trigger moves up one record, using the built-in procedure UP. If the cursor is already on the first record, the built-in fails and the following message usually displays: FRM-40100: At first record.

This is a severity level 5 message. However, the trigger suppresses this, and outputs its own application message instead. The trigger resets the message level to normal (0) afterwards.

```
:SYSTEM.message_level := '5';
UP;
IF NOT FORM_SUCCESS THEN
    MESSAGE('Already at the first Order');
END IF;
:SYSTEM.message_level := '0';
```

Suppressing Messages (continued)

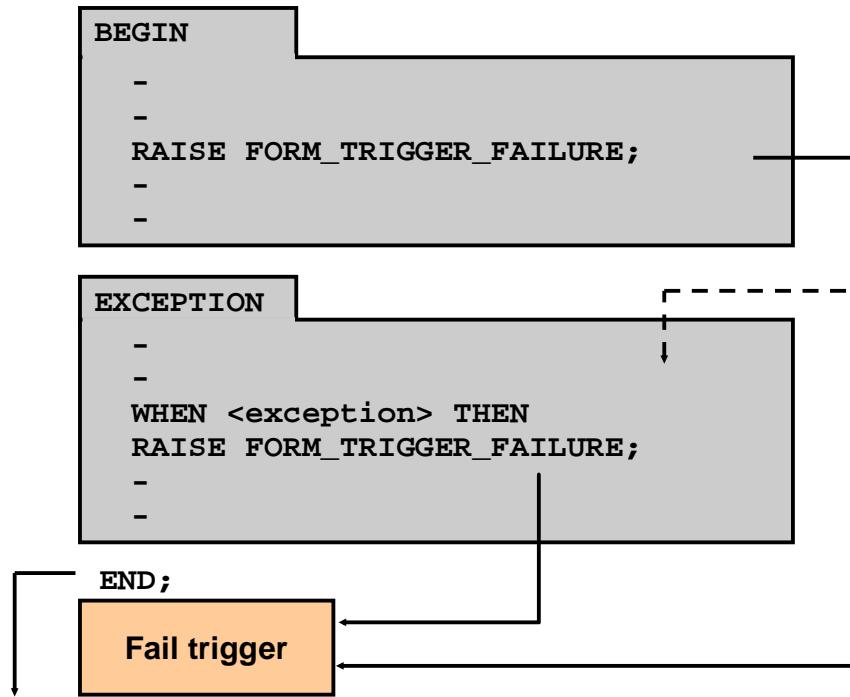
Suppressing Working Messages

Working messages are displayed when Forms is busy processing an action. For example, while querying you receive the message: Working You can suppress this message by setting the suppress_working system variable to True:

```
:SYSTEM.suppress_working := 'TRUE' ;
```

Note: You can set these system variables as soon as the form starts up, if required, by performing the assignments in a When-New-Form-Instance trigger.

FORM_TRIGGER_FAILURE Exception



ORACLE®

17 - 13

Copyright © 2009, Oracle. All rights reserved.

FORM_TRIGGER_FAILURE Exception

Triggers fail only when one of the following occurs:

- An unhandled exception
- When you request the trigger to fail by raising the built-in exception `FORM_TRIGGER_FAILURE`

This exception is defined and handled by Forms Builder, beyond the visible trigger text that you write. You can raise this exception:

- In the executable part of a trigger, to skip remaining actions and fail the trigger
- In an exception handler, to fail the trigger *after* your own exception-handling actions have been obeyed

In either case, Forms Builder has its own exception handler for `FORM_TRIGGER_FAILURE`, which fails the trigger but does *not* cause an unhandled exception. This means that you can fail the trigger in a controlled manner.

The arrows in the slide show that when the `FORM_TRIGGER_FAILURE` exception is raised, whether in the body of the code or in an exception handler, trigger execution stops and does not end gracefully—in other words, the trigger fails.

FORM_TRIGGER_FAILURE Exception (continued)

Example

The following example adds an action to the exception handler of the When-Validate-Item trigger for the Customer_Id item. It raises an exception to fail the trigger when the message is sent, and therefore, traps the user in the Customer_Id item:

```
SELECT cust_first_name || ' ' || cust_last_name
  INTO :orders.customer_name
    FROM customers
   WHERE customer_id = :orders.customer_id;
EXCEPTION
  WHEN no_data_found THEN
    MESSAGE('Customer with this ID not found');
    RAISE form_trigger_failure;
```

Triggers for Intercepting System Messages

- On-Error:
 - Fires when a system error message is issued
 - Is used to trap Forms and Oracle Server errors, and to customize error messages
- On-Message:
 - Fires when an informative system message is issued
 - Is used to suppress or customize specific messages

ORACLE®

17 - 15

Copyright © 2009, Oracle. All rights reserved.

Triggers for Intercepting System Messages

By writing triggers that fire on message events, you can intercept system messages before they are displayed on the screen. These triggers are:

- **On-Error:** Fires on issuance of a system error message
- **On-Message:** Fires on issuance of an informative system message

These triggers replace the display of a message, so that no message is seen by the operator unless you issue one from the trigger itself.

You can define these triggers at any level. For example, an On-Error trigger at item level intercepts only those error messages that occur while control is in that item. However, if you define one or both of these triggers at form level, all messages that cause them to fire will be intercepted regardless of which object in the current form causes the error or message.

On-Error Trigger

Use this trigger to:

- Detect Forms and Oracle Server errors. This trigger can perform corrective actions based on the error that occurred.
- Replace the default error message with a customized message for this application

Triggers for Intercepting System Messages (continued)

Remember that you can use the built-in functions ERROR_CODE, ERROR_TEXT, and ERROR_TYPE to identify the details of the error, and possibly use this information in your own message.

Example of an On-Error Trigger

This On-Error trigger sends a customized message for error 40202 (field must be entered), but reconstructs the standard system message for all other errors. Remember that the “ON-” triggers fire in place of the usual processing.

```
IF ERROR_CODE = 40202 THEN
    MESSAGE('You must fill in this field for an Order');
ELSE
    MESSAGE(ERROR_TYPE || '-' || TO_CHAR(ERROR_CODE) ||
           ': ' || ERROR_TEXT);
END IF;
RAISE FORM_TRIGGER_FAILURE;
```

Handling Informative Messages

- On-Message trigger
- Built-in functions:
 - MESSAGE_CODE
 - MESSAGE_TEXT
 - MESSAGE_TYPE

ORACLE®

17 - 17

Copyright © 2009, Oracle. All rights reserved.

Handling Informative Messages

Use an On-Message trigger to suppress informative messages, replacing them with customized application messages, as appropriate.

You can handle messages in On-Message in a similar way to On-Error. However, because this trigger fires due to informative messages, you will use different built-ins to determine the nature of the current message.

Built-in Function	Description of Returned Value
MESSAGE_CODE	Number of informative message that would have displayed (NUMBER data type)
MESSAGE_TEXT	Text of informative message that would have displayed (CHAR data type)
MESSAGE_TYPE	FRM=Forms Builder message ORA=Oracle server message NULL>No message issued yet in this session (CHAR data type)

Handling Informative Messages (continued)

Note: These functions return information about the most recent message that was issued. If your applications must be supported in more than one national language, use MESSAGE_CODE in preference to MESSAGE_TEXT when checking a message.

Example of an On-Message trigger

This On-Message trigger modifies the “Query caused no records to be retrieved” message (40350) and the “Query caused no records to be retrieved. Re-enter.” message (40301):

```
IF      MESSAGE_CODE in (40350,40301) THEN
    MESSAGE('No Orders found-check your search values');
ELSE
    MESSAGE(MESSAGE_TYPE || '-' || TO_CHAR(MESSAGE_CODE)
           || ':' || MESSAGE_TEXT);
END IF;
```

Setting Alert Properties

The screenshot illustrates the configuration of alert properties. On the left, a Property Palette displays seven items:

Title	This is the Title
Message	Alert Message (Maximum 200 characters) Can appear on multiple lines
Alert Style	Caution
Button 1 Label	Label 1
Button 2 Label	Label 2
Button 3 Label	Label 3
Default Alert Button	Button 1

On the right, a generic alert window is shown with numbered callouts pointing to its components:

- 1: Title bar (labeled "This is the Title")
- 2: Close button (top-right)
- 3: Caution icon (yellow exclamation mark)
- 4: Stop icon (red alarm bell)
- 5: Note icon (yellow note with thumbtack)
- 6: Buttons (labeled "Label 1", "Label 2", "Label 3")
- 7: Message area (labeled "Alert Message (Maximum 200 characters) Can appear on multiple lines")

At the bottom, a red bar contains the ORACLE logo and the copyright notice: "Copyright © 2009, Oracle. All rights reserved."

Setting Alert Properties

Example

The screenshot at the top of the slide shows the Property Palette with the alert properties that you can set. The screenshot at the right of the slide shows a generic example of an alert, with all the three icons and buttons that can be defined.

1	Title
2	Message
3	Alert Style (Caution, Stop, Note)
4	Button1 label
5	Button2 label
6	Button3 label
7	Default Alert Button

At the bottom-left of the slide, the icons that display on all three styles of alerts are depicted:

- **Caution alerts:** Yellow triangle containing an exclamation mark
- **Stop alerts:** Red alarm bell
- **Note alerts:** Note with a thumbtack

Setting Alert Properties (continued)

Creating and Controlling Alerts

Alerts are an alternative method for communicating with the operator. Because they are displayed in a modal window, alerts provide an effective way of drawing attention and forcing the operator to answer the message before processing can continue.

Use alerts when you need to perform the following:

- Display a message that the operator cannot ignore, and must acknowledge.
- Ask the operator a question where up to three answers are appropriate (typically Yes, No, or Cancel).

You handle the display and responses to an alert by using built-in subprograms. Alerts are, therefore, managed in two stages:

- Create the alert at design time, and define its properties in the Property Palette.
- Activate the alert at run time by using built-ins, and take action based on the operator's returned response.

How to Create an Alert

Like the other objects that you create at design time, alerts are created from the Object Navigator.

- Select the Alerts node in the Navigator, and then click Create.
- Define the properties of the alert in the Property Palette.

The properties that you can specify for an alert include the following:

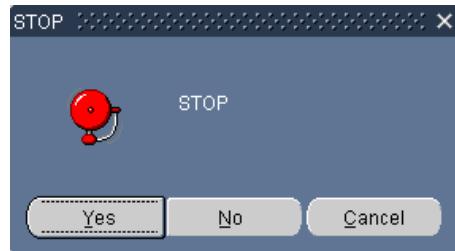
Property	Description
Name	Name for this object
Title	Title that displays on alert
Alert Style	Symbol that displays on alert: Stop, Caution, or Note
Button1, Button2, Button3 Labels	Labels for each of the three possible buttons (Null indicates that the button will not be displayed.)
Default Alert Button	Specifies which button is selected if the user presses Enter
Message	Message that will appear in the alert—can be multiple lines, but maximum of 200 characters

Planning Alerts: How Many Do You Need?

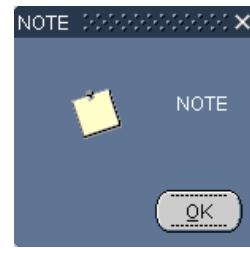
Yes/No
questions



Yes/No/Cancel
questions



Caution
messages



Informative
messages

ORACLE®

Planning Alerts: How Many Do You Need?

Potentially, you can create an alert for every separate alert message that you need to display, but this is usually unnecessary. You can define a message for an alert at run time, before it is displayed to the operator. A single alert can be used for displaying many messages if the available buttons are suitable for responding to the messages.

Create an alert for each combination of:

- Alert style required
- Set of available buttons (and labels) for operator response

For example, an application might require one Note-style alert with a single button (OK) for acknowledgment (depicted at bottom right of slide), one Caution alert with a similar button (depicted at the bottom-left of slide), and two Stop alerts that each provide a different combination of buttons for a reply – the top-left screenshot shows such an alert with Yes and No buttons, while the top-right screenshot shows it with Yes, No, and Cancel buttons. You can then assign a message to the appropriate alert before its display, through the SET_ALERT_PROPERTY built-in procedure.

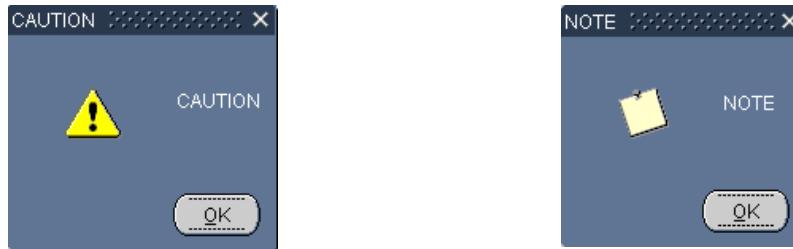
Because the properties for alert title, message, and labels for buttons can be set dynamically at run time, a module should require no more than three generic alerts.

Controlling Alerts at Run Time



SET_ALERT_PROPERTY

SET_ALERT_BUTTON_PROPERTY



ORACLE®

Controlling Alerts at Run Time

There are built-in subprograms to change an alert message, to change alert button labels, and to display the alert and return the operator's response to the calling trigger.

SET_ALERT_PROPERTY Procedure

At form startup, the default message (as defined in the Property Palette) is initially assigned. Use this built-in to change the message that is currently assigned to an alert:

```
SET_ALERT_PROPERTY( 'alert_name' , property , 'message' ) ;
```

Parameter	Description
Alert_name	The name of the alert, as defined in Forms Builder (You can alternatively specify an alert_id [unquoted] for this argument.)
Property	The property being set; use ALERT_MESSAGE_TEXT when defining a new message for this alert
Message	The character string that defines the message (You can give a character expression, instead of a single-quoted string, if required.)

Controlling Alerts at Run Time (continued)

SET_ALERT_BUTTON_PROPERTY Procedure

Use this built-in to change the label on one of the alert buttons:

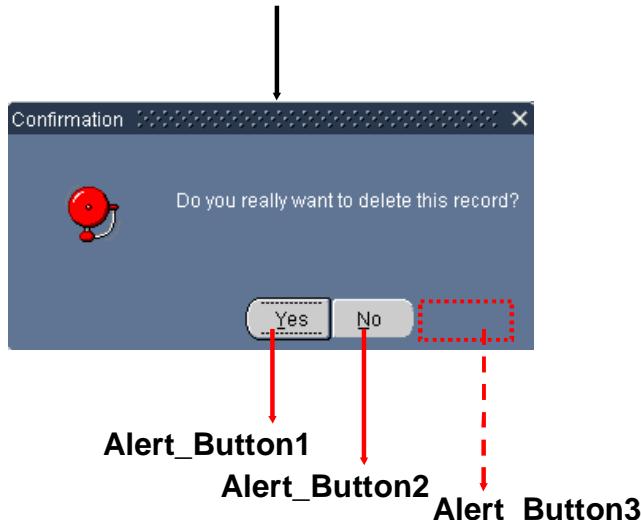
```
SET_ALERT_BUTTON_PROPERTY('alert_name', button, property,  
'value')
```

Parameter	Description
Alert_name	The name of the alert, as defined in Forms Builder (You can alternatively specify an alert_id [unquoted] for this argument.)
Button	The number that specifies the alert button (Use ALERT_BUTTON1, ALERT_BUTTON2, and ALERT_BUTTON3 constants.)
Property	The property being set; use LABEL
Value	The character string that defines the label

The screenshots show the same set of generic alerts as in the previous slide. The SET_ALERT_PROPERTY and SET_ALERT_BUTTON_PROPERTY built-ins enable you to reuse these alerts for different purposes.

Using the SHOW_ALERT Function

```
IF SHOW_ALERT( 'del_Check' )=ALERT_BUTTON1 THEN  
. . .
```



ORACLE®

17 - 24

Copyright © 2009, Oracle. All rights reserved.

Using the SHOW_ALERT Function

SHOW_ALERT is how you display an alert at run time and return the operator's response to the calling trigger:

```
selected_button := SHOW_ALERT( 'alert_name' );
```

Alert_Name is the name of the alert, as defined in the builder. You can alternatively specify an *Alert_Id* (unquoted) for this argument.

SHOW_ALERT returns a NUMBER constant, which indicates which of the three possible buttons the user clicked in response to the alert. These numbers correspond to the values of three PL/SQL constants, which are predefined by the Forms Builder:

If the number equals...	The operator selected...
ALERT_BUTTON1	Button 1
ALERT_BUTTON2	Button 2
ALERT_BUTTON3	Button 3

After displaying an alert that has more than one button, you can determine which button the operator clicked by comparing the returned value against the corresponding constants.

Using the SHOW_ALERT Function (continued)

Example

A trigger that fires when the user attempts to delete a record might invoke the alert shown in the slide to obtain confirmation. If the operator selects Yes, the DELETE_RECORD built-in is called to delete the current record from the block.

```
IF SHOW_ALERT( 'del_check' ) = ALERT_BUTTON1 THEN  
    DELETE_RECORD;  
END IF;
```

The alert shown in the screenshot has only two buttons, but the space where the third button would appear, if defined, is shown with dotted lines.

Directing Errors to an Alert

```
PROCEDURE Alert_On_Failure IS
  n NUMBER;
BEGIN
  SET_ALERT_PROPERTY('error_alert',
    ALERT_MESSAGE_TEXT, ERROR_TYPE || 
    '-' || TO_CHAR(ERROR_CODE) || 
    ': ' || ERROR_TEXT);
  n := SHOW_ALERT('error_alert');
END;
```



Directing Errors to an Alert

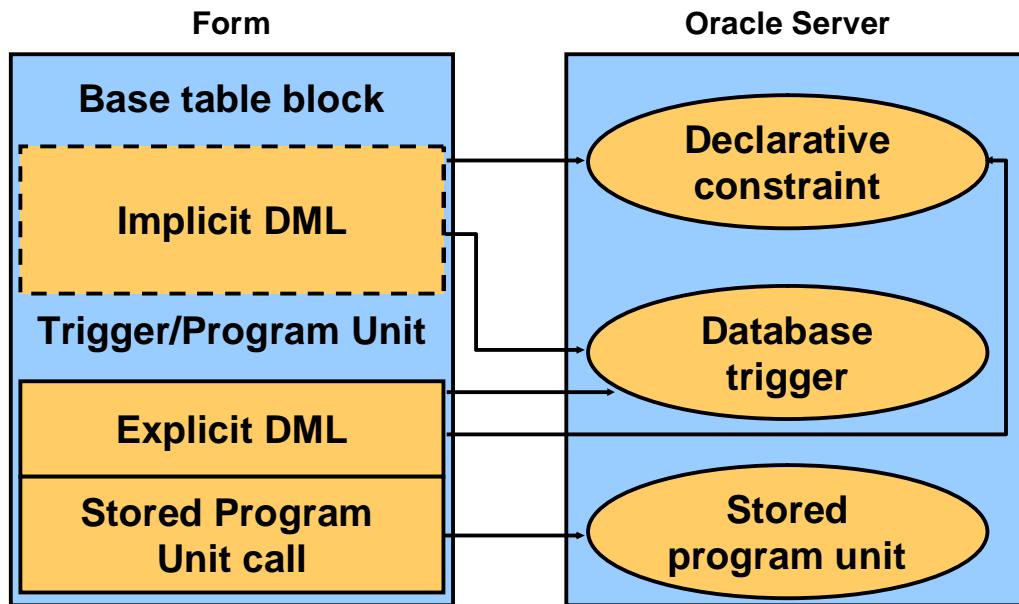
You may want to display errors automatically in an alert, through an On-Error trigger. The built-in functions that return error information, such as `ERROR_TEXT`, can be used in the `SET_ALERT_PROPERTY` procedure, to construct the alert message for display.

Example: The following user-named procedure can be called when the last form action was unsuccessful. The procedure fails the calling trigger and displays Error_Alert containing the error information.

```
PROCEDURE alert_on_failure IS
  n NUMBER;
BEGIN
  SET_ALERT_PROPERTY('error_alert',
    ALERT_MESSAGE_TEXT,
    ERROR_TYPE || '-' || TO_CHAR(ERROR_CODE) || ':' || 
    ERROR_TEXT);
  n := SHOW_ALERT('error_alert');
END;
```

Note: If you want the trigger to fail, include a call to `RAISE FORM_TRIGGER_FAILURE`.

Handling Errors Raised by the Oracle Database Server



Handling Errors Raised by the Oracle Database Server

Oracle Server errors can occur for many different reasons, such as violating a declarative constraint or encountering a stored program unit error. You should know how to handle errors that may occur in different situations. Causes of Oracle Server errors include:

Cause	Error Message
Declarative constraint	Causes predefined error message
Database trigger	Error message specified in RAISE_APPLICATION_ERROR
Stored program unit	Error message specified in RAISE_APPLICATION_ERROR

The arrows in the slide indicate that:

- Implicit data manipulation language (DML) in a form's base table block can encounter errors that are caused by database constraints or triggers
- In triggers or program units in a form:
 - Explicit DML can encounter errors that are caused by database constraints or triggers
 - Stored program unit calls can result in errors from the database stored programs.

Handling Errors Raised by the Oracle Database Server (continued)

Types of DML Statements

Declarative-constraint violations and firing of database triggers are in turn caused by DML statements. For error-handling purposes, you must distinguish between the following two types of DML statements:

Type	Description
Implicit DML	DML statements that are associated with base table blocks. Implicit DML is also called base table DML. By default, Forms constructs and issues these DML statements.
Explicit DML	DML statements that a developer explicitly codes in triggers or program units

FRM-Error Messages Caused by Implicit DML Errors

If an implicit DML statement causes an Oracle Server error, Forms displays one of these FRM-error messages:

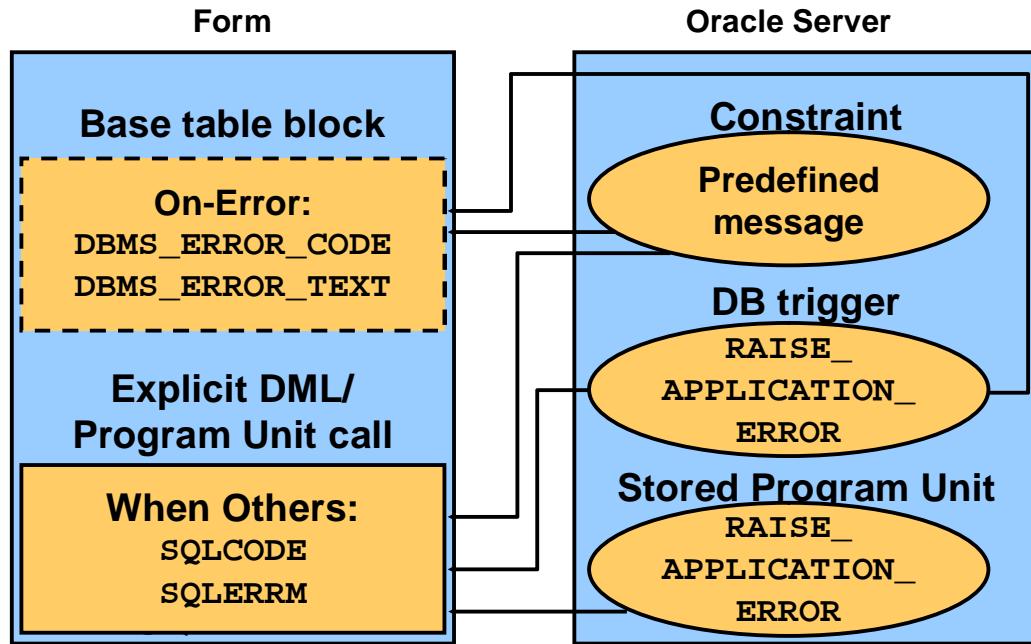
- FRM-40508: ORACLE error: unable to INSERT record.
- FRM-40509: ORACLE error: unable to UPDATE record.
- FRM-40510: ORACLE error: unable to DELETE record.

You can use ERROR_CODE to trap these errors in an On-Error trigger and then use DBMS_ERROR_CODE and DBMS_ERROR_TEXT to determine the ORA-error code and message.

FRM-Error Messages with Web-Deployed Forms

Users may receive a generic FRM-99999 error. You can obtain meaningful information about this error from the Java Control Panel.

Trapping Oracle Database Server Errors



Trapping Oracle Database Server Errors

As shown in the graphics in the slide, Oracle database server errors can be initiated by either implicit DML in a base table block, or explicit DML in a trigger or program unit. How you handle the errors depends on the initiator of the error, as shown in the following table:

Initiator Type	Error Handling
Implicit DML	Use the Forms built-ins DBMS_ERROR_CODE and DBMS_ERROR_TEXT in an On-Error trigger.
Explicit DML	Use the PL/SQL functions SQLCODE and SQLERRM in a WHEN OTHERS exception handler of the trigger or program that issued the DML statements.

As illustrated by the arrows in the slide, declarative-constraint violations and database trigger errors may be caused by both implicit DML and explicit DML. Stored program units are always called explicitly from a trigger or program unit.

Technical Note

The values of DBMS_ERROR_CODE and DBMS_ERROR_TEXT are the same as what a user would see after selecting Help > Display Error; the values are not automatically reset following successful execution.

Summary

In this lesson, you should have learned:

- Forms displays messages at run time to inform the operator of events that occur in the session
- You can use FORM_SUCCESS to test for run-time failure of built-ins
- There are four types of Forms messages:
 - Informative
 - Error
 - Working
 - Application



Summary

In this lesson, you should have learned how to intercept system messages, and how to replace them with the ones that are more suitable for your application. You also learned how to build customized alerts for communicating with operators.

- Test for failure of built-ins by using the FORM_SUCCESS built-in function or other built-in functions.
- Forms messages can be Informative, Error, Working, or Application messages.

Summary

- You can control system messages with built-ins and triggers:
 - MESSAGE_LEVEL
 - SUPPRESS_WORKING
 - On-[Error | Message] triggers
 - [ERROR | MESSAGE]_[CODE | TEXT | TYPE]
- There are different types of alerts: Stop, Caution, and Note
- There are different alert built-ins:
 - SHOW_ALERT
 - SET_ALERT_PROPERTY
 - SET_ALERT_BUTTON_PROPERTY

ORACLE®

17 - 31

Copyright © 2009, Oracle. All rights reserved.

Summary (continued)

- Set system variables to suppress system messages:
 - Assign a value to message_level to specify that only messages above a specific severity level are to be used by the form.
 - Assign a value of True to suppress_working to suppress all working messages.
- On-Error and On-Message triggers intercept system error messages and informative system messages.
- You can use the built-ins ERROR_CODE, ERROR_TEXT, ERROR_TYPE, MESSAGE_CODE, MESSAGE_TEXT, or MESSAGE_TYPE to obtain information about the number, text, and type of errors and messages.
- Alert types: Stop, Caution, and Note
- Up to three buttons are available for response (NULL indicates no button).
- Display alerts and change alert messages at run time with SHOW_ALERT and SET_ALERT_PROPERTY.

Summary

- How to handle database server errors:
 - Implicit DML: Use DBMS_ERROR_CODE and DBMS_ERROR_TEXT in the On-Error trigger
 - Explicit DML: Use SQLCODE and SQLERRM in the WHEN OTHERS exception handler

ORACLE®

17 - 32

Copyright © 2009, Oracle. All rights reserved.

Summary (continued)

- Intercept and handle server-side errors.

Practice 17: Overview

This practice covers the following topics:

- Using a customized alert to inform the operator that the customer's credit limit has been exceeded
- Using a generic alert to ask the operator to confirm that the form should terminate



Practice 17: Overview

In this practice, you create some alerts. These include a general alert for questions and a specific alert that is customized for credit limit.

- Using a customized alert to inform the operator that the customer's credit limit has been exceeded
- Using a generic alert to ask the operator to confirm that the form should terminate; involves setting alert properties programmatically

18

Query Triggers

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

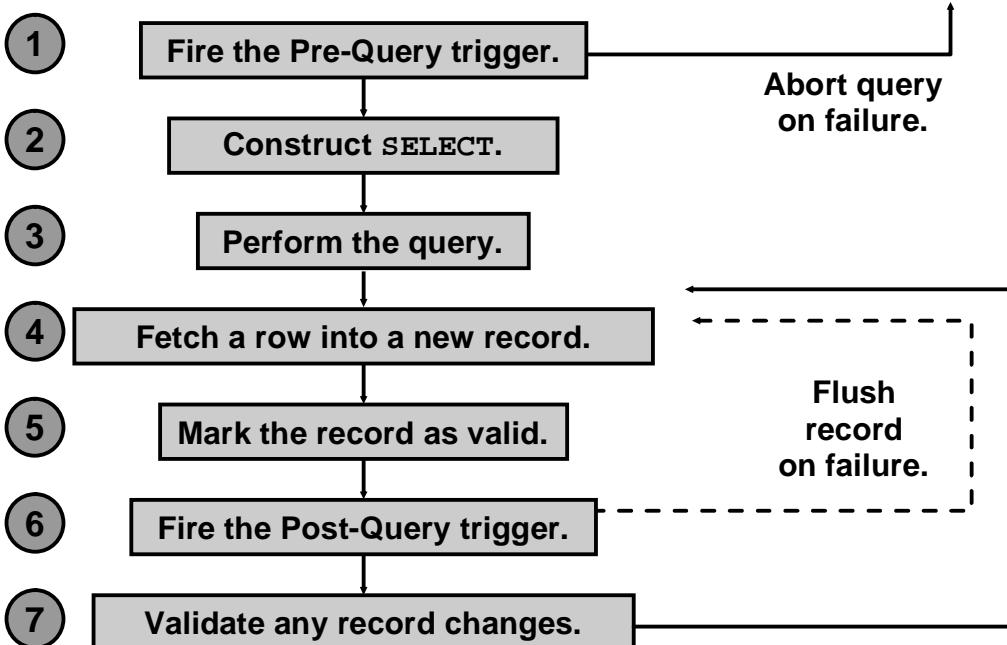
- Explain the processes involved in querying a data block
- Describe query triggers and their scope
- Write triggers to screen query conditions
- Write triggers to supplement query results
- Control trigger action based on the form's query status



Lesson Aim

In this lesson, you learn how to control events associated with queries on base-table data blocks. You can customize the query process as necessary and supplement the results returned by a query.

Query Processing: Overview



ORACLE®

18 - 3

Copyright © 2009, Oracle. All rights reserved.

Query Processing: Overview

Generally, triggers are associated with a query in one of two ways:

- A trigger fires due to the query process itself (for example, Pre-Query and Post-Query).
- An event can fire a trigger in Enter-Query mode, if the “Fire in Enter-Query Mode” property of the associated trigger is enabled.

The query triggers, Pre-Query and Post-Query, fire due to the query process itself, and are usually defined on the block where the query takes place.

With these triggers, you can add to the normal Forms processing of records, or possibly abandon a query before it is even executed, if the required conditions are not suitable.

Forms Query Processing

When a query is initiated on a data block, either by the operator or by a built-in subprogram, the following major events take place:

1. Forms fires the Pre-Query trigger if defined. As indicated by the arrow in the slide, the query is aborted if the Pre-Query trigger fails.
2. If the Pre-Query succeeds, Forms constructs the query SELECT statement, based on any existing criteria in the block (either entered by the operator or by Pre-Query).

Query Processing: Overview (continued)

3. The query is performed.
4. Forms fetches the column values of a row into the base-table items of a new record in the block.
5. The record is marked Valid.
6. Forms fires the Post-Query trigger. If it fails, this record is flushed from the block, as indicated by the arrow in the slide, and then steps 4 through 7 are repeated for any remaining records of this query.
7. Forms performs item and record validation if the record has changed (due to a trigger).
8. As indicated by the arrow in the slide, steps 4 through 7 are repeated for any remaining records of this query.

SELECT Statements Issued During Query Processing

```
SELECT      base_column, ..., ROWID
INTO        :base_item, ..., :ROWID
FROM        base_table
WHERE       (default_where_clause
              OR onetime_where_clause)
AND         (example_record_conditions)
AND         (query_where_conditions)
ORDER BY    default_order_by_clause |
              query_where_order_by
```

Slightly different for COUNT

ORACLE®

18 - 5

Copyright © 2009, Oracle. All rights reserved.

SELECT Statements Issued During Query Processing

If you have not altered default query processing, Forms issues a SELECT statement when you want to retrieve or count records.

```
SELECT      base_column,      base_column, ..., ROWID
INTO        :base_item,       :base_item, ..., :ROWID
FROM        base_table
WHERE       (default_where_clause OR onetime_where_clause)
AND         (example_record_conditions)
AND         (query_where_conditions)
ORDER BY    default_order_by_clause | query_where_order_by

SELECT      COUNT(*)
FROM        base_table
WHERE       (default_where_clause OR onetime_where_clause)
AND         (example_record_conditions)
AND         (query_where_conditions)
ORDER BY    default_order_by_clause | query_where_order_by
```

SELECT Statements Issued During Query Processing (continued)

Note: The vertical bar (|) in the ORDER BY clause indicates that either of the two possibilities can be present. Forms retrieves the ROWID only when the Key Mode block property is set to Unique (the default). The entire WHERE clause is optional. The ORDER BY clause is also optional.

If you want to count records that satisfy criteria specified in the Query/Where dialog box, enter one or more variables in the example record and click Count Query.

Using a WHERE Clause

- Four sources for the WHERE clause:
 - WHERE Clause block property
 - ONETIME_WHERE block property
 - Example Record
 - Query/Where dialog box
- WHERE clauses are combined by the AND operator, except that WHERE and ONETIME_WHERE are mutually exclusive:
 - ONETIME_WHERE is used the first time the query executes.
 - WHERE is used for subsequent executions of the query.

ORACLE®

18 - 7

Copyright © 2009, Oracle. All rights reserved.

Using a WHERE Clause

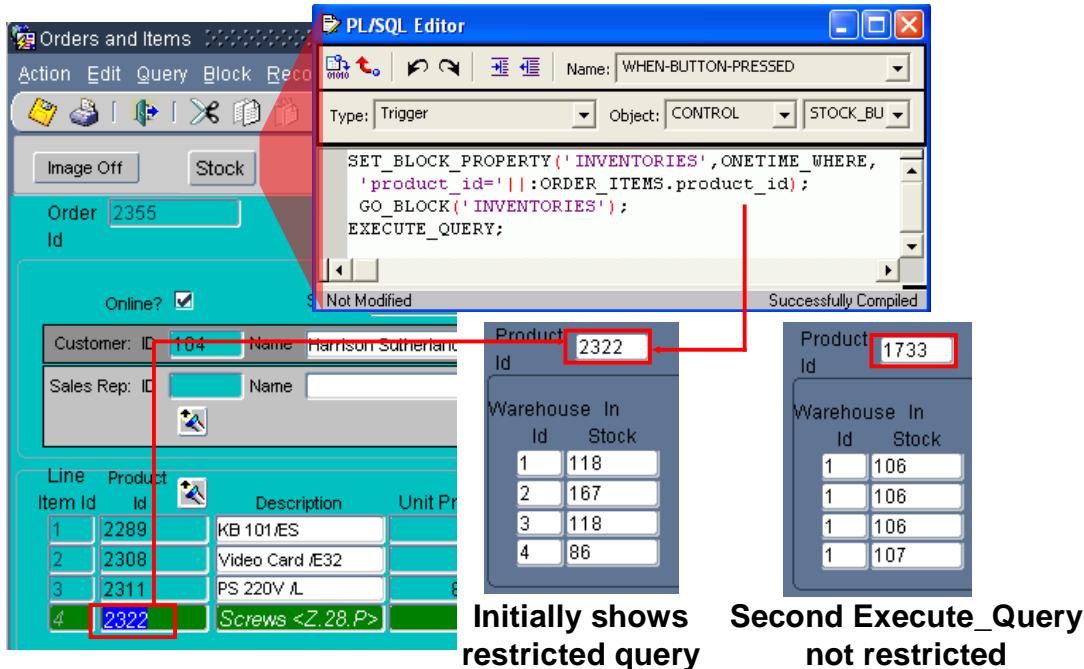
The WHERE and ORDER BY clauses of a default base table SELECT statement are derived from several sources. It is important to know how different sources interact.

Four Sources for the WHERE Clause

- The WHERE Clause block property (set in Forms Builder, or by setting the DEFAULT_WHERE_CLAUSE property programmatically)
- The ONETIME_WHERE block property (set programmatically)
- Example Record
- Query/Where dialog box

If more than one source is present, the different conditions will all be used and linked with an AND operator. If the WHERE clause and the ONETIME_WHERE clause are present, only one is used: the ONETIME_WHERE clause for the first execution of the query, and the WHERE clause for subsequent executions.

Setting the ONETIME_WHERE Property



Setting the ONETIME_WHERE Property

For instances where you want to restrict the query only once, you can programmatically set the ONETIME_WHERE property on a block. This specifies a WHERE clause for the block that will be in effect only for the first query issued on the block after setting that property.

Example

From the ORDER_ITEMS block, you want to display the INVENTORIES block, which is in a separate window. When the block is initially displayed, it should present information about the stock of the product selected in the ORDER_ITEMS block. However, after this initial information is displayed, you want users to be able to query the stock of any product. This functionality is illustrated by the three screenshots at the bottom of the slide that show:

- The Orders form with product 2322 selected in the order items block
- The INVENTORIES window initially displayed, showing inventory for product 2322
- INVENTORIES after a second query, showing inventory for a different product

You accomplish this by coding the Stock button to set the ONETIME_WHERE property, as shown in the following code in the slide:

```
SET_BLOCK_PROPERTY('inventories', ONETIME_WHERE,
'product_id = ||:order_items.product_id');
Go_BLOCK('inventories');
EXECUTE_QUERY;
```

Using an ORDER BY Clause

- Two sources for the ORDER BY clause are:
 - The ORDER BY Clause block property
 - The Query/Where dialog box
- The second source for the ORDER BY clause overrides the first one.

ORACLE®

18 - 9

Copyright © 2009, Oracle. All rights reserved.

Using an ORDER BY Clause

Two Sources for the ORDER BY Clause

- The ORDER BY Clause block property
- The Query/Where dialog box

An ORDER BY clause specified in the Query/Where dialog box overrides the value of the ORDER BY Clause block property.

Note: You can use the SET_BLOCK_PROPERTY built-in to change the WHERE Clause and ORDER BY Clause block properties at run time.

Writing Query Triggers: Pre-Query Trigger

- Defined at block level
- Fires once, before the query is performed

```
IF    TO_CHAR(:orders.order_id) ||
      TO_CHAR(:orders.customer_id)
IS NULL THEN
  MESSAGE('You must query by
          Order ID or Customer ID');
  RAISE form_trigger_failure;
END IF;
```



Writing Query Triggers: Pre-Query Trigger

You must define this trigger at block level or above. It fires for either a global or restricted query, just before Forms executes the query. You can use Pre-Query to:

- Test the operator's query conditions, and to fail the query process if the conditions are not satisfactory for the application
- Add criteria for the query by assigning values to base-table items

Example

The Pre-Query trigger on the ORDERS block shown in the slide permits queries only if there is a restriction on either the Order_Id or Customer_Id. This prevents very large queries.

Note: Pre-Query is useful for assigning values passed from other Oracle Forms Developer modules, so that the query is related to data elsewhere in the session. You will learn how to do this in the lesson titled "Introducing Multiple Form Applications."

Writing Query Triggers: Post-Query Trigger

- Fires for each fetched record (except during array processing)
- Is used to populate nondatabase items and calculate statistics

```
SELECT COUNT(order_id)
INTO :orders.lineitem_count
FROM order_items
WHERE order_id = :orders.order_id;
```



Writing Query Triggers: Post-Query Trigger

This trigger is defined at block level or above. Post-Query fires for each record that is fetched into the block as a result of a query. Note that the trigger fires only on the initial fetch of a record, not when a record is subsequently scrolled back into view a second or third time.

Use Post-Query as follows:

- To populate nondatabase items as records are returned from a query
- To calculate statistics

Example

The Post-Query trigger on the ORDERS block, shown in the slide, selects the total count of line items for the current Order, and displays this number as a summary value in the non-base-table item Lineitem_Count.

Writing Query Triggers: Using SELECT Statements in Triggers

- Forms Builder variables are preceded by a colon.
- The query must return one row for success.
- Code exception handlers.
- The INTO clause is mandatory, with a variable for each selected column or expression.
- ORDER BY is not relevant.

ORACLE®

18 - 12

Copyright © 2009, Oracle. All rights reserved.

Writing Query Triggers: Using SELECT Statements in Triggers

The previous trigger example populates the Lineitem_Count item through the INTO clause. Again, colons are required in front of Forms Builder variables to distinguish them from PL/SQL variables and database columns.

Here is a reminder of some other rules regarding SELECT statements in PL/SQL:

- A single row must be returned from the query, or else an exception is raised that terminates the normal executable part of the PL/SQL block. You usually want to match a form value with a unique column value in your restriction.
- Code exception handlers in your PL/SQL block to deal with possible exceptions raised by SELECT statements.
- The INTO clause is mandatory, and must define a receiving variable for each selected column or expression. You can use PL/SQL variables, form items, or global variables in the INTO clause.
- ORDER BY and other clauses that control multiple-row queries are not relevant (unless they are part of an Explicit Cursor definition).

Query Array Processing

- Reduces network traffic
- Enables Query Array processing:
 - Enable the Array Processing option.
 - Set the Query Array Size property.
- The Query Array Size property
- The Query All Records property

ORACLE®

18 - 13

Copyright © 2009, Oracle. All rights reserved.

Query Array Processing

The default behavior of Forms is to process records one at a time. With array processing, a structure (array) containing multiple records is sent to or returned from the server for processing.

Forms supports both array fetch processing and array data manipulation language (DML) processing. For both querying and DML operations, you can determine the array size to optimize performance for your needs. This lesson focuses on array processing of queries rather than DML.

You can enable array processing for queries by:

- Setting preferences:
 - Select Edit > Preferences.
 - Click the Runtime tab.
 - Select the Array Processing check box.
- Setting block properties:
 - In the Object Navigator, double-click the node of the block to display the Property Palette.
 - Under the Records category, set the Query Array Size property to a number that represents the number of records in the array for array processing.

Query Array Processing (continued)

Query Array Size Property: This property specifies the maximum number of records that Forms should fetch from the database at one time. If set to zero, the query array size defaults to the number of records displayed in the block.

A size of 1 provides the fastest perceived response time because Forms fetches and displays only one record at a time. By contrast, a size of 10 fetches up to ten records before displaying any of them; however, the larger size reduces overall processing time by making fewer calls to the database for records.

Query All Records Property: This property specifies whether all the records matching the query criteria should be fetched into the data block when a query is executed.

- **Yes:** Fetches all records from query
- **No:** Fetches the number of records specified by the Query Array Size block property

Coding Triggers for Enter-Query Mode

- Some triggers may fire in Enter-Query mode.
- Set the “Fire in Enter-Query Mode” property.
- Test mode during execution with :SYSTEM.mode
 - NORMAL
 - ENTER-QUERY
 - QUERY

ORACLE®

18 - 15

Copyright © 2009, Oracle. All rights reserved.

Coding Triggers for Enter-Query Mode

Some triggers that fire when the form is in Normal mode (during data entry and saving) may also be fired in Enter-Query mode. You need to consider the trigger type and actions in these cases.

“Fire in Enter-Query Mode” Property

To create a trigger that fires in Enter-Query mode, in its Property Palette set the “Fire in Enter-Query Mode” property to Yes. This property determines whether Forms fires a trigger if the associated event occurs in Enter-Query mode. Not all triggers can do this; consult Forms Builder online Help, which lists each trigger and whether this property can be set.

By default, the “Fire in Enter-Query Mode” property is set to Yes for triggers that accept this. Set it to No in the Property Palette if you want the trigger to fire only in Normal mode.

Coding Triggers for Enter-Query Mode (continued)

Example

If you provide a button for the operator to invoke an LOV, and the LOV is required to help with query criteria as well as data entry, the When-Button-Pressed trigger should fire in both modes. This trigger has “Fire in Enter-Query Mode” set to Yes (default for this trigger type):

```
IF SHOW_LOV( 'customers' ) THEN  
    MESSAGE( 'Selection successful' );  
END IF;
```

Note: The following triggers may fire in Enter-Query mode:

- Key-
- On-Error
- On-Message
- When- triggers, *except*:
 - When-Database-Record
 - When-Image-Activated
 - When-New-Block-Instance
 - When-New-Form-Instance
 - When>Create-Record
 - When-Remove-Record
 - When-Validate-Record
 - When-Validate-Item

Using Built-ins in Enter-Query Mode

Some built-in subprograms are illegal if a trigger is executed in Enter-Query mode. Again, consult the Forms Builder online Help, which specifies whether an individual built-in can be used in this mode.

One general restriction is that in Enter-Query mode you cannot navigate to another record in the current form. So any built-in that would potentially enable this is illegal. These include GO_BLOCK, NEXT_BLOCK, PREVIOUS_BLOCK, GO_RECORD, NEXT_RECORD, PREVIOUS_RECORD, UP, DOWN, OPEN_FORM, and so on.

Determining the Current Mode

- Example:

```
IF :SYSTEM.mode = 'NORMAL'  
THEN ENTER_QUERY;  
ELSE EXECUTE_QUERY;  
END IF;
```

- Some built-ins are illegal.
- Consult online Help.
- You cannot navigate to another record in the current form.

ORACLE®

18 - 17

Copyright © 2009, Oracle. All rights reserved.

Determining the Current Mode

When a trigger fires in both Enter-Query mode and Normal modes, you may need to know the current mode at execution time for the following reasons:

- Your trigger needs to perform different actions depending on the mode.
- Some built-in subprograms cannot be used in Enter-Query mode.

The read-only system variable, mode, stores the current mode of the form. Its value (always in uppercase) is one of the following:

Value of :SYSTEM.MODE	Definition
NORMAL	Form is in Normal processing mode.
ENTER-QUERY	Form is in Enter-Query mode.
QUERY	Form is in Fetch-processing mode, meaning that Forms is currently performing a fetch. (For example, this value always occurs in a Post-Query trigger.)

Determining the Current Mode (continued)

Example

Consider the following When-Button-Pressed trigger for the Query button.

If the operator clicks the button in Normal mode, the trigger places the form in Enter-Query mode (using the ENTER_QUERY built-in). Otherwise, if already in Enter-Query mode, the button executes the query (using the EXECUTE_QUERY built-in).

```
IF :SYSTEM.MODE = 'NORMAL' THEN
    ENTER_QUERY;
ELSE
    EXECUTE_QUERY;
END IF;
```

Overriding Default Query Processing

Additional transactional triggers for query processing

Trigger	Do-the-right-thing built-in
On-Close	
On-Count	COUNT_QUERY
On-Fetch	FETCH_RECORDS
Pre-Select	
On-Select	SELECT_RECORDS
Post-Select	

ORACLE®

18 - 19

Copyright © 2009, Oracle. All rights reserved.

Overriding Default Query Processing

You can use certain transactional triggers to replace default commit processing. Some of the transactional triggers can also be used to replace default query processing.

You can call “Do-the-right-thing” built-ins from transactional triggers to augment default query processing. The “Do-the-right-thing” built-ins perform the same actions as the default processing. You can then supplement the default processing with your own code.

Using Transactional Triggers for Query Processing

Transactional triggers for query processing are primarily intended to access certain data sources other than Oracle. However, you can also use these triggers to implement special functionality by augmenting default query processing against an Oracle database.

Overriding Default Query Processing (continued)

Transactional Triggers for query processing have the following characteristics:

Trigger	Characteristic
On-Close	Fires when Forms closes a query (It augments, rather than replaces, default processing.)
On-Count	Fires when Forms would usually perform default Count Query processing to determine the number of rows that match the query conditions
On-Fetch	Fires when Forms performs a fetch for a set of rows (You can use the CREATE_QUERIED_RECORD built-in to create queried records if you want to replace default fetch processing.) The trigger continues to fire until: <ul style="list-style-type: none">• No queried records are created during a single execution of the trigger• The query is closed by the user or by the ABORT_QUERY built-in executed from another trigger• The trigger raises FORM_TRIGGER_FAILURE
Pre-Select	Fires after Forms has constructed the block SELECT statement based on the query conditions, but before it issues this statement
On-Select	Fires when Forms would usually issue the block SELECT statement (The trigger replaces the open cursor, parse, and execute phases of a query.)
Post-Select	Fires after Forms has constructed and issued the block SELECT statement, but before it fetches the records

Obtaining Query Information at Run Time

- SYSTEM.mode
- SYSTEM.last_query
 - Contains bind variables (ord_id = :1) before SELECT_RECORDS
 - Contains actual values (ord_id = 102) after SELECT_RECORDS
- GET/SET_ITEM_PROPERTY
 - Get and set:
 - CASE_INSENSITIVE_QUERY
 - QUERYABLE
 - QUERY_ONLY
 - Get only:
 - QUERY_LENGTH
- GET/SET_BLOCK_PROPERTY
 - Get and set:
 - DEFAULT_WHERE
 - ONETIME_WHERE
 - ORDER_BY
 - QUERY_ALLOWED
 - QUERY_HITS
 - Get only:
 - QUERY_OPTIONS
 - RECORDS_TO_FETCH

ORACLE®

18 - 21

Copyright © 2009, Oracle. All rights reserved.

Obtaining Query Information at Run Time

You can use system variables and built-ins to obtain information about queries.

Using **SYSTEM.mode**

Use the SYSTEM.mode system variable to obtain the form mode. The three values are NORMAL, ENTER_QUERY, and QUERY.

Using **SYSTEM.last_query**

Use SYSTEM.last_query to obtain the text of the base-table SELECT statement that was last executed by Forms. If a user has entered query conditions in the Example Record, the exact form of the SELECT statement depends on when this system variable is used.

If the system variable is used before Forms has implicitly executed the SELECT_RECORDS built-in, the SELECT statement contains bind variables (for example, order_id=:1). If used after Forms has implicitly executed the SELECT_RECORDS built-in, the SELECT statement contains the actual search values (for example, order_id=102). The system variable contains bind variables during the Pre-Select trigger and actual search values during the Post-Select trigger.

Unlike most system variables, SYSTEM.last_query may contain a mixture of uppercase and lowercase letters.

Obtaining Query Information at Run Time (continued)

Using `GET_ITEM_PROPERTY` and `SET_ITEM_PROPERTY`

The following item properties may be useful for getting query information. The property marked with an asterisk cannot be set.

- `CASE_INSENSITIVE_QUERY`
- `QUERYABLE`: Determines whether the item can be included in a query against the base table of the block to which the item belongs
- `QUERY_ONLY`: Specifies that an item can be queried but that it should not be included in any `INSERT` or `UPDATE` statement that Forms issues for the block at run time
- `QUERY_LENGTH *`

Using `GET_BLOCK_PROPERTY` and `SET_BLOCK_PROPERTY`

The following block properties may be useful for obtaining query information. The properties marked with an asterisk cannot be set.

- `DEFAULT_WHERE`
- `ONETIME_WHERE`
- `ORDER_BY`
- `QUERY_ALLOWED`
- `QUERY_HITS`
- `QUERY_OPTIONS *`
- `RECORDS_TO_FETCH *`

Summary

In this lesson, you should have learned that:

- Query processing includes the following steps:
 1. Pre-Query trigger fires
 2. SELECT statement constructed
 3. Query performed
 4. Record fetched into block
 5. Record marked Valid
 6. Post-Query trigger fires
 7. Item and record validation if the record has changed (due to a trigger)
 8. Steps 4 through 7 repeat till all fetched



Summary

In this lesson, you learned how to control the events associated with queries on base-table blocks.

- **The query process:** Before beginning the query, the Pre-Query trigger fires once for each query. Then the query statement is constructed and executed. For each record retrieved by the query, the record is fetched into the block and marked as valid, the Post-Query trigger fires for that record, and item and record validation occurs if a trigger has changed the record.

Summary

- The query triggers, which must be defined at block or form level, are:
 - Pre-Query: Use to screen query conditions (set ONETIME_WHERE or DEFAULT_WHERE properties, or assign values to use as query criteria)
 - Post-Query: Use to supplement query results (populate non-base-table items, perform calculations)
- You can use transactional triggers to override default query processing
- You can control trigger action based on the form's query status by checking SYSTEM.MODE values: NORMAL, ENTER-QUERY, or QUERY

ORACLE®

18 - 24

Copyright © 2009, Oracle. All rights reserved.

Summary (continued)

- The Pre-Query trigger fires before the query executes. This trigger is defined at the block level or above. Use the Pre-Query trigger to check or modify query conditions.
- The Post-Query trigger fires as each record is fetched (except array processing). This trigger is defined at the block level or above. Use the Post-Query trigger to perform calculations and populate additional items.
- Some triggers can fire in both Normal and Enter-Query modes.
 - Use SYSTEM.MODE to test the current mode.
 - Some built-ins are illegal in Enter-Query mode.
- Override default query processing by using transactional triggers; to replace the default functionally, use “Do-the-right-thing” built-ins.
- Obtain query information at run-time by using:
 - SYSTEM.mode, SYSTEM.last_query
 - Some properties of GET/SET_BLOCK_PROPERTY and GET/SET_ITEM_PROPERTY

Practice 18: Overview

This practice covers the following topics:

- Populating customer names and sales representative names for each row of the ORDERS block
- Populating descriptions for each row of the ORDER_ITEMS block
- Restricting the query on the INVENTORIES block for only the first query on that block
- Disabling the effects of the Exit button and changing a radio group in Enter-Query mode
- Adding two check boxes to enable case-sensitive and exact match query

ORACLE®

18 - 25

Copyright © 2009, Oracle. All rights reserved.

Practice 18: Overview

In this practice, you create two query triggers to populate non-base-table items. You also change the default query interface to enable case-sensitive and exact match queries.

19

Validation

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Explain the effects of the validation unit upon a form
- Control validation using:
 - Object properties
 - Triggers
 - Pluggable Java Components
- Describe how Forms tracks validation status
- Control when validation occurs

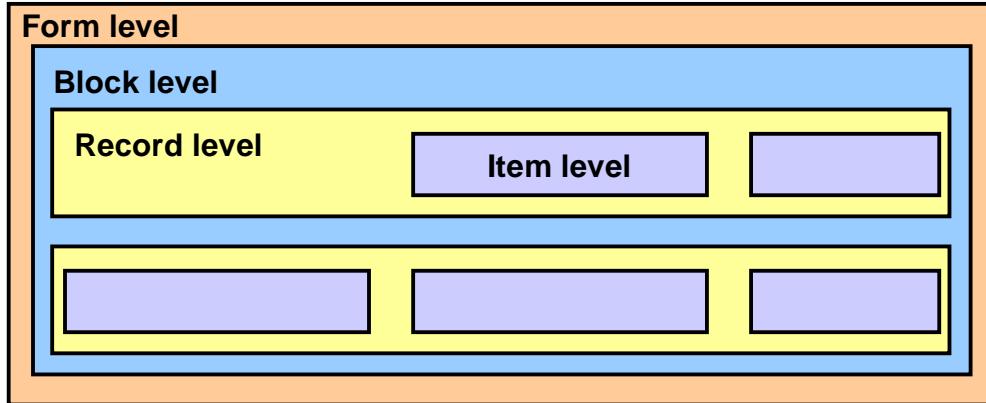


Lesson Aim

In this lesson, you learn how to supplement item validation by using both object properties and triggers. You also learn to control when validation occurs.

Validation Process

- Forms validates at the following levels:



- Validation occurs when:
 - The Enter key is pressed or the ENTER built-in procedure is run
 - The operator or trigger leaves the validation unit

ORACLE®

19 - 3

Copyright © 2009, Oracle. All rights reserved.

Validation Process

Validation Levels

Forms performs a validation process at several levels to ensure that records and individual values follow appropriate rules. If validation fails, control is passed back to the appropriate level, so that the operator can make corrections. Validation occurs at:

- Item level:** Forms records a status for each item to determine whether it is currently valid. If an item has been changed and is not yet marked as valid, Forms first performs standard validation checks to ensure that the value conforms to the item's properties. These checks are carried out before firing any When-Validate-Item triggers that you have defined. Standard checks include the following:
 - Format mask
 - Required (if so, is the item null?)
 - Data type
 - Range (Lowest-Highest Allowed Value)
 - Validate from List (see later in this lesson)

Validation Process (continued)

- **Record level:** After leaving a record, Forms checks to see whether the record is valid. If not, the status of each item in the record is checked, and a When-Validate-Record trigger is then fired, if present. When the record passes these checks, it is set to valid.
- **Block or form level:** At the block or form level, all records below that level are validated. For example, if you commit (save) changes in the form, all records in the form are validated, unless you have suppressed this action.

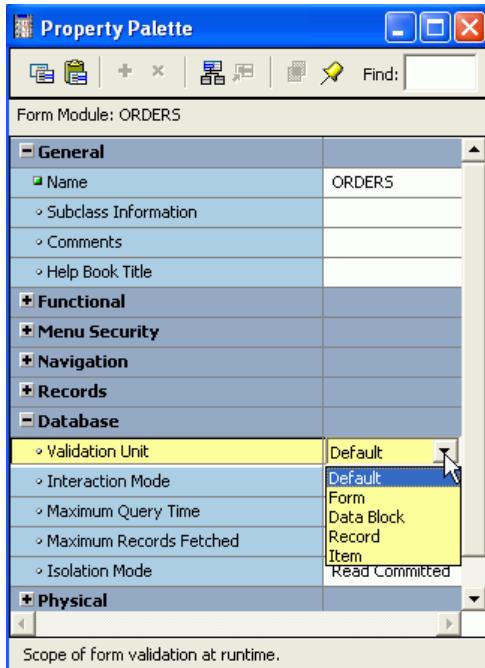
The graphics in the slide illustrate that validation occurs at the form, block, record, and item levels. A box representing a form contains another box representing a block. The block contains two boxes representing records, each of which contains smaller boxes representing items.

When Does Validation Occur?

Forms carries out validation for the validation unit under the following conditions:

- The Enter key is pressed or the ENTER built-in procedure is run. The purpose of the ENTER built-in is to force validation immediately.
Note: The ENTER action is not necessarily mapped to the key that is physically labeled Enter.
- The operator or a trigger navigates out of the validation unit. This includes when changes are committed. The default validation unit is item, but can also be set to record, block, or form by the designer. The validation unit is discussed in the next slide.

Using Object Properties to Control Validation



ORACLE®

19 - 5

Copyright © 2009, Oracle. All rights reserved.

Using Object Properties to Control Validation

You can control when and how validation occurs in a form, even without triggers. Do this by setting properties for the form and for individual items within it.

Validation Unit

The validation unit defines the maximum amount of data an operator can enter in the form before Forms initiates validation. Validation unit is a property of the form module, and it can be set in the Property Palette (as shown in the slide) to:

- Default
- Item
- Record
- Data Block
- Form

The default setting is item level. The default setting is usually chosen.

In practice, an item-level validation unit means that Forms validates changes when an operator navigates out of a changed item. This way, standard validation checks and firing the When-Validate-Item trigger of that item can be done immediately. As a result, operators are aware of validation failure as soon as they attempt to leave the item.

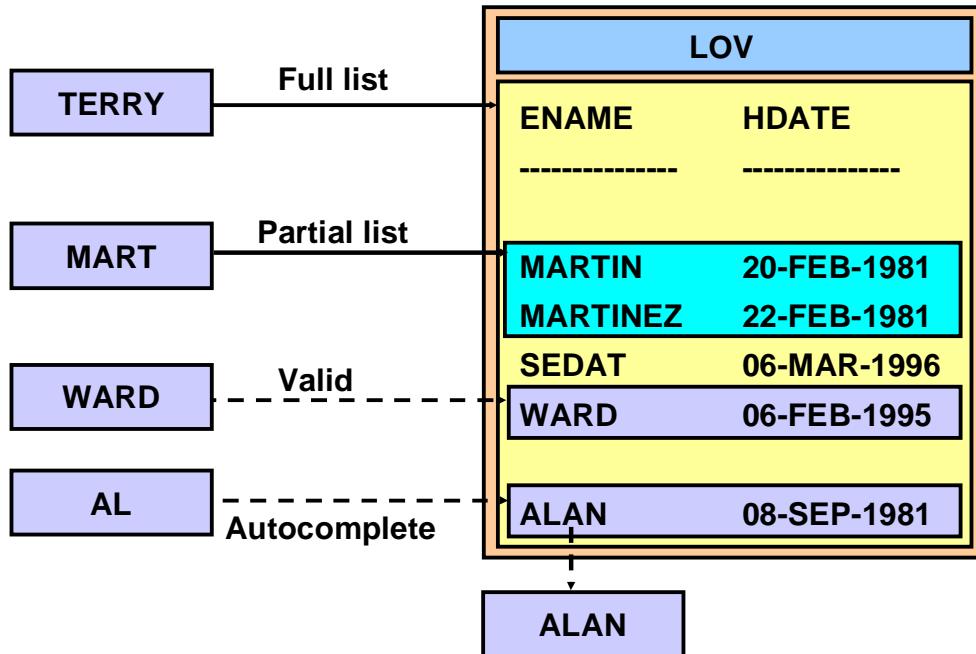
Using Object Properties to Control Validation (continued)

At higher validation units (record, block, or form level), the above checks are postponed until navigation moves out of that unit. All outstanding items and records are validated together, including the firing of the When-Validate-Item and When-Validate-Record triggers.

You might set a validation unit above item level under one of the following conditions:

- Validation involves database references, and you want to postpone traffic until the operator has completed a record (record level).
- You want to improve performance by reducing round trips to the application server.

Using LOVs for Validation



Using LOVs for Validation

When you attach a list of values (LOV) to a text item by setting the LOV property of the item, you can optionally use the LOV contents to validate data entered in the item.

The “Validate from List” Property

You can use the LOV for validation by setting the “Validate from List” property to Yes for the item. During validation, Forms then automatically uses the item value as a non-case-sensitive search string on the LOV contents. The following events then occur, depending on the circumstances:

- If the item’s value causes a single record to be found in the LOV, but is a partial value of the LOV value, the full LOV column value is returned to the item (provided the item is defined as the return item in the LOV). The item then passes this validation phase.
- If the value in the text item matches one of the values in the first column of the LOV, validation succeeds, the LOV is not displayed, and processing continues normally.
- If the item value causes multiple records to be found in the LOV, Forms displays the LOV and uses the text item value as the search criteria to automatically reduce the list, so that the operator must choose.
- If no match is found, the full LOV contents are displayed to the operator.

Using LOVs for Validation (continued)

Example

The LOV in the slide graphic contains five names: MARTIN, MARTINEZ, SEDAT, WARD, and ALAN. The graphic illustrates the four possible scenarios for LOV validation:

- When the item's value is AL, Forms autocompletes the value in the text item because a single value matching the text item's partial value was found in the LOV.
- When the item's value is WARD, it matches one of the values in the first column of the LOV. So validation succeeds, the LOV is not displayed, and processing continues normally.
- When the item's value is MART, it matches multiple records in the LOV. Forms displays the LOV with only MARTIN and MARTINEZ in the list, and the operator can then choose the correct value.
- When the item's value is TERRY, no match is found, and the full LOV contents are displayed to the operator.

Note: Make sure that the LOVs you create for validation purposes have the validation column defined first, with a display width greater than 0. You also need to define the Return Item for the LOV column as the item being validated.

For performance reasons, do not use the “LOV for Validation” property for large LOVs.

Using Triggers to Control Validation

- Item level:
When-Validate-Item
- Block level:
When-Validate-Record

```
IF :orders.order_date > SYSDATE THEN
    MESSAGE('Order Date is later than today!');
    RAISE FORM_TRIGGER_FAILURE;
END IF;
```

ORACLE®

19 - 9

Copyright © 2009, Oracle. All rights reserved.

Using Triggers to Control Validation

There are triggers that fire due to validation, which let you add your own customized actions. There are also some built-in subprograms that you can call from triggers that affect validation.

The When-Validate-Item Trigger

This trigger fires after standard item validation. Input focus is returned to the item if the trigger fails.

Example

The When-Validate-Item trigger on `orders.order_date` shown in the slide ensures that the Order_Date is not later than the current (database) date.

Using Triggers to Control Validation (continued)

The When-Validate-Record Trigger

This trigger fires after standard record-level validation, when the operator has left a new or changed record. Because Forms has already checked that required items for the record are valid, you can use this trigger to perform additional checks that may involve more than one of the record's items, in the order in which they were entered.

When-Validate-Record must be defined at block level or above.

Example

This When-Validate-Record trigger on the ORDER_ITEMS block warns the operator when a line item for a new credit order causes the customer's credit limit to be exceeded. This involves a cross-check of two items in the record, Order_Status and Customer_Id, so a When-Validate-Item trigger would not be a good choice for this type of validation.

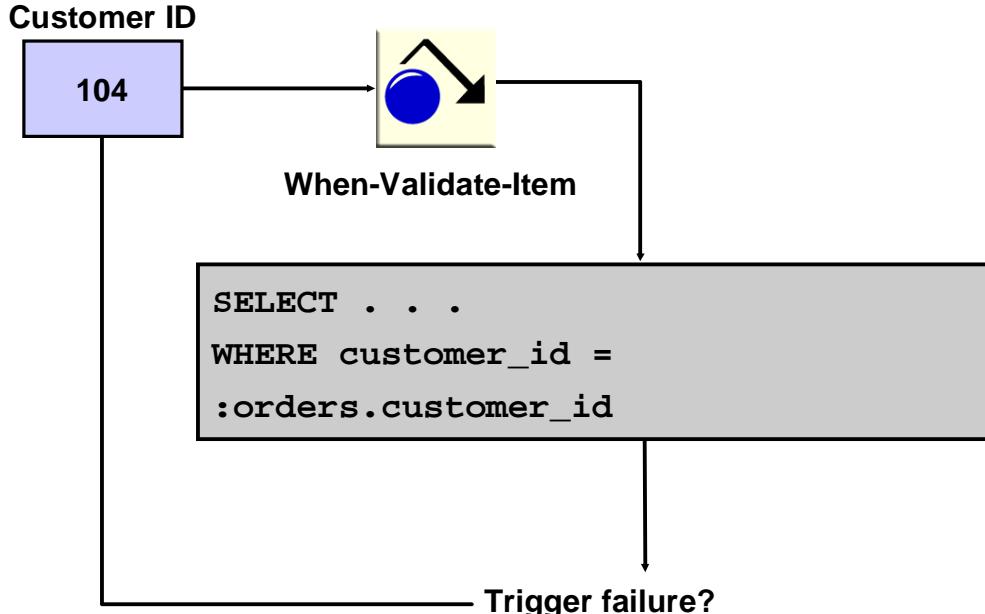
```
DECLARE
    cred_limit number;
    n number;
BEGIN
    -- Order status of 4 is new credit order
    IF :orders.order_status = 4 THEN
        SELECT credit_limit INTO cred_limit FROM customers
            WHERE customer_id = :orders.customer_id;
        IF :control.total > cred_limit THEN
            n := show_alert('credit_limit_alert');
        END IF;
    END IF;
END;
```

Note: If you want to stop the operator from navigating out of the item when validation fails, you can raise an exception to fail the trigger:

```
RAISE FORM_TRIGGER_FAILURE;
```

In the example above, you include this code immediately after displaying the alert.

Example: Validating User Input



ORACLE®

19 - 11

Copyright © 2009, Oracle. All rights reserved.

Example: Validating User Input

The slide example shows the following When-Validate-Item trigger that is placed on the Customer_ID item. It returns the name corresponding to the Customer ID entered by the user.

```
SELECT cust_first_name || ' ' || cust_last_name
INTO :orders.customer_name
FROM customers
WHERE customer_id = :orders.customer_id;
```

If the user enters an invalid value in the item, a matching row is not found, and the SELECT statement causes an exception. The success or failure of the query can, therefore, be used to validate user input.

The exceptions that can occur when a single row is not returned from a SELECT statement in a trigger are:

- NO_DATA_FOUND: No rows are returned from the query.
- TOO_MANY_ROWS: More than one row is returned from the query.

Example: Validating User Input (continued)

If the Customer_ID item contains a value that is not found in the table, the NO_DATA_FOUND exception is raised, and the trigger fails because there is no exception handler to prevent the exception from propagating to the end of the trigger.

In this example, the SELECT statement returns a value to a non-base-table item in the form. However, it can simply return the value to a variable that is declared in the trigger if there is no need to return a value to a text item.

Note: A failing When-Validate-Item trigger prevents the cursor from leaving the item.

For an unhandled exception, as above, the user receives the following message:

```
FRM-40735: <trigger type> trigger raised unhandled exception  
<exception>
```

Using Client-Side Validation

- Forms validation:
 - Occurs on the middle tier
 - Involves network traffic
- Client-side validation:
 - Improves performance
 - Implemented with PJC

The image contains two screenshots of an Oracle Forms application. Both screenshots show a grid with columns: Line, Product Id, Description, Unit Price, and Quantity. The Quantity column is highlighted with a red border.

Top Screenshot: A message box at the bottom says "FRM-50016: Legal characters are 0-9 - + E.". A red box highlights the Quantity column. Below the grid, a message box says "Using number datatype".

Bottom Screenshot: A message box at the bottom says "Enter a numeric value". A red box highlights the Quantity column. Below the grid, a message box says "Using KeyFilter PJC".

Using Client-Side Validation

When you use a When-Validate-Item trigger for validation, the trigger itself is processed on the WebLogic Server. Even validation that occurs with a format mask on an item involves a trip to the middle tier. In the first example in the slide, alphabetic characters have been entered in the Quantity item. The number data type on the item is not checked until the operator presses Enter to send the input to the Forms Services machine, which returns the error FRM-50016: Legal characters are 0-9.

You should consider using Pluggable Java Components (PJC)s to replace the default functionality of standard client items, such as text boxes. Then validation of items, such as the date or maximum or minimum values, is contained within an item. This technique opens up opportunities for more complex, application-specific validation, such as automatic formatting of input—for example, telephone numbers with the format (XXX) XXX-XXXX. Even a simple numeric format is enforced instantly, not allowing alphabetic keystrokes to be entered into the item.

This validation is performed on the client without involving a network round trip, thus improving performance. In the second example in the slide, the KeyFilter PJC does not allow the operator to enter an alphabetic character into the Quantity item. The only message that is displayed on the message line is the item's Hint: Enter a numeric value.

Using Client-Side Validation (continued)

PJCs are similar to JavaBeans, and in fact, the two terms are often used interchangeably.

Although both are Java components that you can use in a form, there are the following differences between them:

- JavaBeans are implemented in a bean area item, whereas PJCs are implemented in a native Forms item, such as a text item or check box.
- PJCs must always have the implementation class specified in the Property Palette, but JavaBeans may be registered at run time with the FBean package.

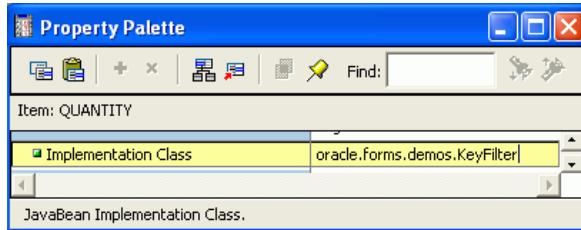
Several JavaBeans and PJCs that have been written for Oracle Forms are available at <http://forms.pjc.bean.over-blog.com>.

Note: Non-input items, such as Chart Item, Display Item, Hierarchical Tree, and Image Item, do not have the Implementation Class property and cannot be extended by a PJC.

Using a PJC

To use a PJC:

- Set the item's Implementation Class property



- Set properties for the PJC

```
SET_CUSTOM_PROPERTY('order_items.quantity',
1,'FILTER_TYPE','NUMERIC');
```

ORACLE®

19 - 15

Copyright © 2009, Oracle. All rights reserved.

Using a PJC

You implement a PJC to replace an item by setting the item's Implementation Class property to the class of the PJC. You may use the SET_CUSTOM_PROPERTY built-in to set properties of the PJC that restrict input or otherwise validate the item. At run time, Forms looks for the Java class contained on the middle tier or in the archive files with the path specified in the Implementation Class for the item.

The screenshot shows setting the Implementation Class in the Property Palette of the Quantity item to oracle.forms.demos.KeyFilter. If you open keyfilter.jar in WinZip, you find that the path to KeyFilter.class is oracle\forms\demos.

You deploy the PJC as you would a JavaBean, which was discussed in the lesson titled “Adding Functionality to Items.” You can locate the Java class file:

- On the middle-tier server, either in the directory structure referenced by the form applet's CODEBASE parameter or in the server's CLASSPATH. CODEBASE is by default the forms\java subdirectory of ORACLE_HOME.
- In a Java Archive (JAR) file in the middle-tier server's CODEBASE directory, and included in the ARCHIVE parameter so that the JAR file is downloaded to and cached on the client. For example: archive=frmall.jar,keyfilter.jar

(The CODEBASE and ARCHIVE parameters are set in the formsweb.cfg file.)

Tracking Validation Status

- NEW:
 - When a record is created
 - Also for “Copy Value from Item” or Initial Value
- CHANGED:
 - When changed by the user or trigger
 - When any item in the new record is changed
- VALID:
 - When validation has been successful
 - After the records are fetched from database
 - After a successful post or commit
 - Duplicated record inherits the status of source

ORACLE®

19 - 16

Copyright © 2009, Oracle. All rights reserved.

Tracking Validation Status

When Forms leaves an object, it usually validates any changes that were made to the contents of the object. To determine whether validation must be performed, Forms tracks the validation status of items and records.

Tracking Validation Status (continued)

Item Validation Status

Status	Definition
NEW	When a record is created, Forms marks every item in that record as new. This is true even if the item is populated by the “Copy Value from Item” or Initial Value item properties, or by the When-Create-Record trigger.
CHANGED	Forms marks an item as changed under the following conditions: <ul style="list-style-type: none">• When the item is changed by the user or a trigger• When any item in a new record is changed, all of the items in the record are marked as changed.
VALID	Forms marks an item as valid under the following conditions: <ul style="list-style-type: none">• All the items in the record that are fetched from the database are marked as valid.• If validation of the item has been successful• After successful post or commit• Each item in a duplicated record inherits the status of its source.

Record Validation Status

Status	Definition
NEW	When a record is created, Forms marks that record as new. This is true even if the item is populated by the “Copy Value from Item” or Initial Value item properties, or by the When-Create-Record trigger.
CHANGED	Whenever an item in a record is marked as changed, Forms marks that record as changed.
VALID	Forms marks a record as valid under the following conditions: <ul style="list-style-type: none">• After all items in the record have been successfully validated• All records that are fetched from the database are marked as valid.• After successful post or commit• A duplicate record inherits the status of its source.

Using Built-Ins to Control When Validation Occurs

- CLEAR_BLOCK, CLEAR_FORM, EXIT_FORM
- ENTER
- SET_FORM_PROPERTY
 - (..., VALIDATION)
 - (..., VALIDATION_UNIT)
- ITEM_IS_VALID item property
- VALIDATE (scope)

ORACLE®

19 - 18

Copyright © 2009, Oracle. All rights reserved.

Using Built-Ins to Control When Validation Occurs

You can use the following built-in subprograms in triggers to affect validation.

CLEAR_BLOCK, CLEAR_FORM, and EXIT_FORM

The first parameter to these built-ins, COMMIT_MODE, controls what will be done with unapplied changes when a block is cleared, the form is cleared, or the form is exited, respectively. When the parameter is set to NO_VALIDATE, changes are neither validated nor committed (by default, the operator is prompted for the action).

ITEM_IS_VALID Item Property

You can use GET_ITEM_PROPERTY and SET_ITEM_PROPERTY

built-ins with the ITEM_IS_VALID parameter to get or set the validation status of an item.

You cannot directly get and set the validation status of a record. However, you can get or set the validation status of all the items in a record.

ENTER

The ENTER built-in performs the same action as the Enter key. That is, it forces validation of data in the current validation unit.

Using Built-Ins to Control When Validation Occurs (continued)

SET_FORM_PROPERTY

You can use this to disable Forms validation. For example, suppose you are testing a form, and you need to bypass normal validation. Set the Validation property to Property_False for this purpose:

```
SET_FORM_PROPERTY( 'form_name' , VALIDATION , PROPERTY_FALSE );
```

You can also use this built-in to change the validation unit programmatically:

```
SET_FORM_PROPERTY( 'form_name' , VALIDATION_UNIT , scope );
```

VALIDATE

VALIDATE(scope) forces Forms to immediately execute validation processing for the indicated scope.

Note: Scope is one of DEFAULT_SCOPE, BLOCK_SCOPE, RECORD_SCOPE, or ITEM_SCOPE.

Summary

In this lesson, you should have learned that:

- The validation unit specifies how much data is entered before validation occurs
- You can control validation using:
 - Object properties: Validation Unit (form); “Validate from List” (item)
 - Triggers: When-Validate-Item (item level); When-Validate-Record (block level)
 - PJC for client-side validation
- Forms tracks validation status of items and records
- You can use built-ins to control when validation occurs

ORACLE®

19 - 20

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you learned to use additional validation features in Forms Builder, and to control when validation occurs.

- Validation occurs at several levels: Item, Record, Block, and Form
- Validation happens when:
 - The Enter key is pressed or the ENTER built-in procedure is run (to force validation immediately)
 - Control leaves the validation unit due to navigation or Commit
- Standard validation occurs before trigger validation.
- The Default validation unit is item level.
- The When-Validate-“*object*” triggers supplement standard validation.
- You can use PJC to perform client-side validation.
- Forms tracks validation status internally: NEW, CHANGED, or VALID
- You can use built-ins to control when validation occurs.

- CLEAR_BLOCK	- ENTER
- CLEAR_FORM	- ITEM_IS_VALID
- EXIT_FORM	- VALIDATE

Practice 19: Overview

This practice covers the following topics:

- Validating the Sales Representative item value by using an LOV
- Writing a validation trigger to check that online orders are credit orders
- Populating customer names, sales representative names, and IDs when a customer ID is changed
- Writing a validation trigger to populate the name and the price of the product when the product ID is changed
- Restricting user input to numeric characters using a PJC



Practice 19: Overview

In this practice, you introduce additional validation to the Customers and Orders form modules.

20

Navigation

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Distinguish between internal and external navigation
- Control navigation with properties
- Describe and use navigation triggers to control navigation
- Use navigation built-ins in triggers



Lesson Aim

Forms Builder offers a variety of ways to control cursor movement. This lesson looks at the different methods of forcing navigation both visibly and invisibly.

Navigation: Overview

- What is the navigational unit?
 - Outside the form
 - Form
 - Block
 - Record
 - Item
- Entering and leaving objects
- What happens if navigation fails?

ORACLE®

20 - 3

Copyright © 2009, Oracle. All rights reserved.

Navigation: Overview

The following sections introduce a number of navigational concepts to help you understand the navigation process.

What Is the Navigational Unit?

The navigational unit is an invisible, internal object that determines the navigational state of a form. Forms uses the navigational unit to keep track of the object that is currently the focus of a navigational process. The navigational unit can be one of the objects in the following hierarchy:

- Outside the form
- Form
- Block
- Record
- Item

When Forms navigates, it changes the navigational unit moving through this object hierarchy until the target item is reached.

Navigation: Overview (continued)

Entering and Leaving Objects

During navigation, Forms leaves and enters objects. Entering an object means changing the navigational unit from the object above in the hierarchy. Leaving an object means changing the navigational unit to the object above.

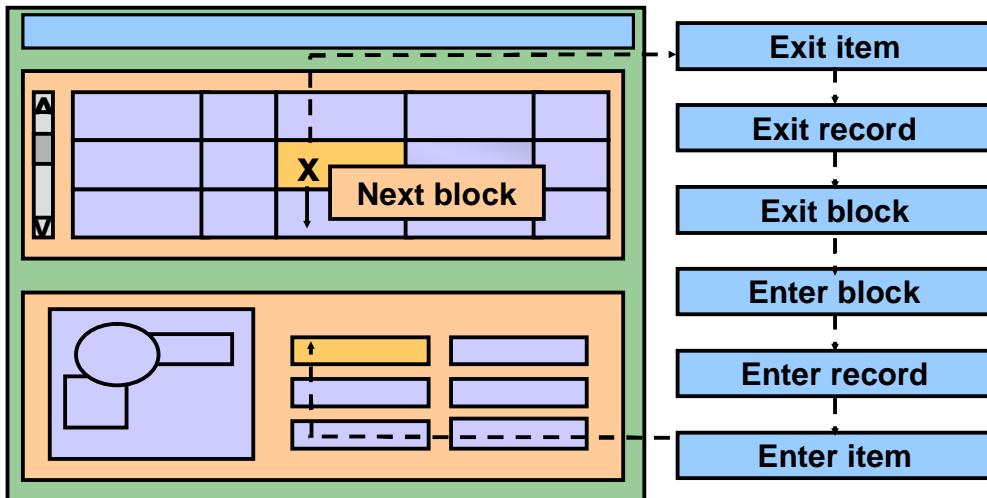
The Cursor and How It Relates to the Navigational Unit

The cursor is a visible, external object that indicates the current input focus. Forms will not move the cursor until the navigational unit has successfully become the target item. In this sense, the navigational unit acts as a probe.

What Happens if Navigation Fails?

If navigation fails, Forms reverses the navigation path and attempts to move the navigational unit back to its initial location. Note that the cursor is still at its initial position. If Forms cannot move the navigational unit back to its initial location, it exits the form.

Internal Navigation



Internal Navigation

Navigation occurs when the user or a trigger causes the input focus to move to another object. You have seen that navigation involves changing the location of the input focus on the screen. In addition to the visible navigation that occurs, some logical navigation takes place. This logical navigation is also known as internal navigation.

Internal Navigation at Form Startup

When you enter a form module, you see the input focus in the first enterable item of the first navigation block. You do not see the internal navigation events that must occur for the input focus to enter the first item. These internal navigation events are as follows:

- Entry to form
- Entry to block
- Entry to record
- Entry to item

Internal Navigation (continued)

Internal Navigation During Commit

When you commit your inserts, updates, and deletes to the database, you do not see the input focus moving. However, the following navigation events must occur internally before commit processing begins:

- Exit current item
- Exit current record
- Exit current block

Example

The example in the slide shows a form with two multirecord blocks. The cursor is initially in the second item in the second record of the first block. When the user selects to navigate to the next block, internal navigation occurs in addition to the visible navigation of moving the cursor to the first item in the next block, as shown in the slide.

Performance Note

Forms uses smart event bundling: All the events that are triggered by navigation between two objects are delivered as one packet to Forms Services on the middle tier for subsequent processing.

Using Object Properties to Control Navigation

- Block:
 - Navigation Style
 - Previous Navigation Data Block
 - Next Navigation Data Block
- Item:
 - Enabled
 - Keyboard Navigable
 - Mouse Navigate
 - Previous Navigation Item
 - Next Navigation Item

ORACLE®

20 - 7

Copyright © 2009, Oracle. All rights reserved.

Using Object Properties to Control Navigation

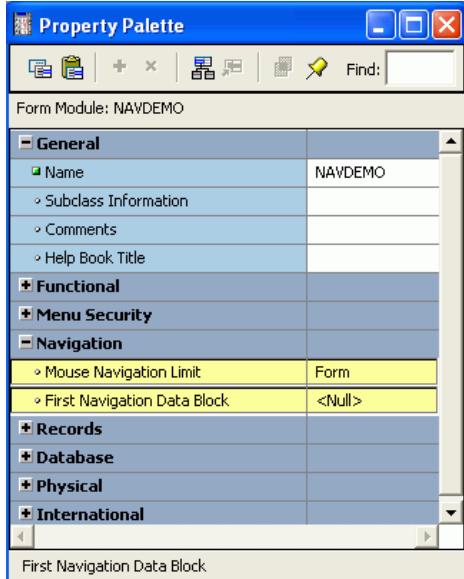
You can control the path through an application by controlling the order in which the user navigates to objects. You have seen navigation properties for blocks and items:

Object	Property
Block	Navigation Style Previous Navigation Data Block Next Navigation Data Block
Item	Enabled Keyboard Navigable Mouse Navigate Previous Navigation Item Next Navigation Item

Note: You can use the mouse to navigate to any enabled item regardless of its position in the navigational order.

Using Object Properties to Control Navigation

- Form module:
 - Mouse Navigation Limit
 - First Navigation Data Block



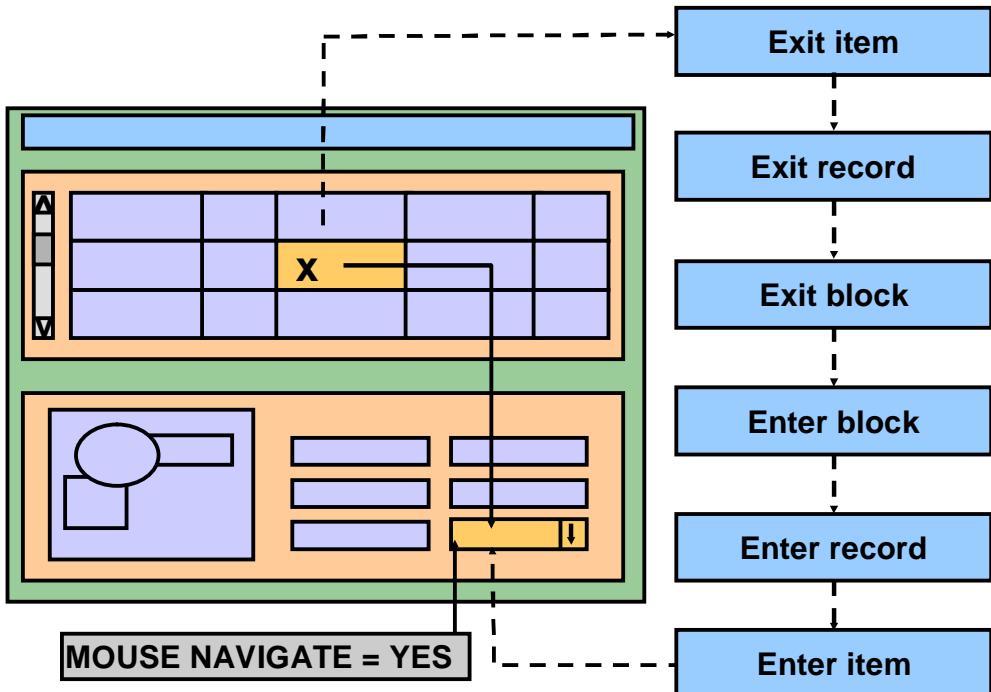
ORACLE®

Using Object Properties to Control Navigation (continued)

There are two other navigation properties, which you can set for the form module: Mouse Navigation Limit and First Navigation Data Block.

Form Module Property	Function
Mouse Navigation Limit	Determines how far outside the current item the user can navigate with the mouse
First Navigation Block	Specifies the name of the block to which Forms should navigate on form startup (Setting this property does not override the order used for committing.)

Using the Mouse Navigate Property



Using the Mouse Navigate Property

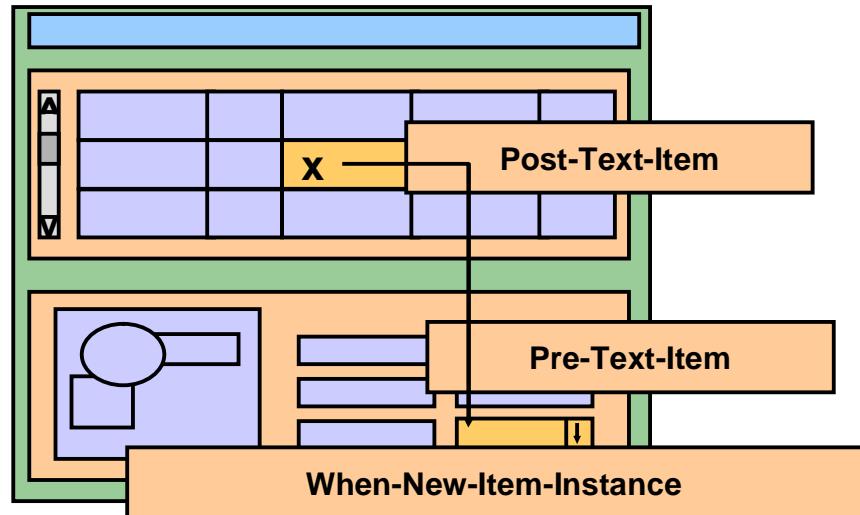
The Mouse Navigate property is valid for the following items:

- Push button
- Check box
- List item
- Radio group
- Hierarchical tree item
- Bean Area Item

Note: The default setting for the Mouse Navigate property is Yes.

Setting	Use to Ensure That:
Yes	Forms navigates to the new item. (This causes the relevant navigational and validation triggers to fire.) This is the situation shown in the slide graphic – the user is in block 1 and clicks an item in block 2. When Mouse Navigate is Yes, Forms performs all the internal navigation necessary to place the cursor in the item.
No	Forms does not navigate to the new item or validate the current item when the user activates the new item with the mouse.

Navigation Triggers Behavior



ORACLE®

Navigation Triggers Behavior

The navigation triggers can be subdivided into two general groups:

- Pre- and Post- navigation triggers
- When-New-<object>-Instance triggers

When Do Pre- and Post-Navigation Triggers Fire?

The Pre- and Post- navigation triggers fire during navigation, just before entry to or just after exit from the object specified as part of the trigger name.

Example

The example in the slide shows that when navigating from one text item to another:

- The Post-Text-Item trigger fires for the item being left
- The Pre-Text-Item trigger fires just before entering the new text item
- The When-New-Item-Instance trigger fires once navigation to the new item is complete

When Do Navigation Triggers Not Fire?

The Pre- and Post-navigation triggers do not fire if they belong to a unit that is lower in the hierarchy than the current validation unit. For instance, if the validation unit is Record, Pre- and Post-Text-Item triggers do not fire.

Navigation Triggers Behavior

Pre- and Post-	When-New-<object>-Instance
Fire during navigation	Fire after navigation
Do not fire if validation unit is higher than trigger object	Fire even when validation unit is higher than trigger object
Allow unrestricted built-ins	Allow restricted and unrestricted built-ins
Handle failure by returning to initial object	Are not affected by failure

ORACLE®

20 - 11

Copyright © 2009, Oracle. All rights reserved.

Navigation Triggers Behavior (continued)

When Do When-New-<object>-Instance Triggers Fire?

The When-New-<object>-Instance triggers fire immediately after navigation to the object specified as part of the trigger name. For example, the When-New-Item-Instance trigger fires immediately after navigation to a new instance of an item.

What Happens when a Navigation Trigger Fails?

If a Pre- or Post-navigation trigger fails, the input focus returns to its initial location (where it was prior to the trigger firing). To the user, it appears that the input focus has not moved at all.

Note: Be sure that Pre- and Post-navigation triggers display a message on failure. Failure of a navigation trigger can cause a fatal error to your form. For example, failure of Pre-Form, Pre-Block, Pre-Record, or Pre-Text-Item on entry to the form will cancel the execution of the form.

Using When-New-<object>Instance Triggers

- When-New-Form-Instance
- When-New-Block-Instance
- When-New-Record-Instance
- When-New-Item-Instance

ORACLE®

20 - 12

Copyright © 2009, Oracle. All rights reserved.

Using When-New-<object>Instance Triggers

If you include complex navigation paths through your application, you may want to check or set initial conditions when the input focus arrives in a particular block, record, or item. Use the following triggers to do this:

Trigger	Fires
When-New-Form-Instance	Whenever a form is run, after successful navigation into the form
When-New-Block-Instance	After successful navigation into a block
When-New-Record-Instance	After successful navigation into the record
When-New-Item-Instance	After successful navigation to a new instance of the item

Initializing Forms Object Instances

```
SET_FORM_PROPERTY(FIRST_NAVIGATION_BLOCK,  
'order_items');
```

```
SET_BLOCK_PROPERTY('orders', ORDER_BY,  
'customer_id');
```

```
SET_RECORD_PROPERTY(3, 'order_items', STATUS,  
QUERY_STATUS);
```

```
SET_ITEM_PROPERTY('control.stock_button',  
ICON_NAME, 'stock');
```

ORACLE®

20 - 13

Copyright © 2009, Oracle. All rights reserved.

Initializing Forms Object Instances

Use the When-New-<object>-Instance triggers, along with the SET_<object>_PROPERTY built-in subprograms to initialize Forms Builder objects. These triggers are particularly useful if you conditionally require a default setting.

Example

The following example of a When-New-Block-Instance trigger conditionally sets the DELETE_ALLOWED property to FALSE.

```
IF GET_APPLICATION_PROPERTY(username) = 'SCOTT' THEN  
SET_BLOCK_PROPERTY('order_items', DELETE_ALLOWED,  
PROPERTY_FALSE);  
END IF;
```

Example

Perform a query of all orders, when the Orders form is run, by including the following code in your When-New-Form-Instance trigger:

```
EXECUTE_QUERY;
```

Initializing Forms Object Instances (continued)

Example

Register the Color Picker JavaBean into the Control.Colorpicker bean area item when the CUSTOMERS form is run by including the following code in your When-New-Form-Instance trigger:

```
FBean.Register_Bean('control.colorpicker',1,  
'oracle.forms.demos.beans.ColorPicker');
```

At run time, Forms looks for the Java class contained on the middle tier or in the archive files with the path specified in the code. (If you open colorpicker.jar in WinZip, you find that the path to ColorPicker.class is oracle\forms\demos\beans.)

Using Pre- and Post-Triggers

- Pre/Post-Form
- Pre/Post-Block
- Pre/Post-Record
- Pre/Post-Text-Item

Post-Block Trigger Example:

```
SET_ITEM_PROPERTY('control.stock_button',  
enabled, property_false);
```

Disabling the Stock button when leaving the ORDER_ITEMS block



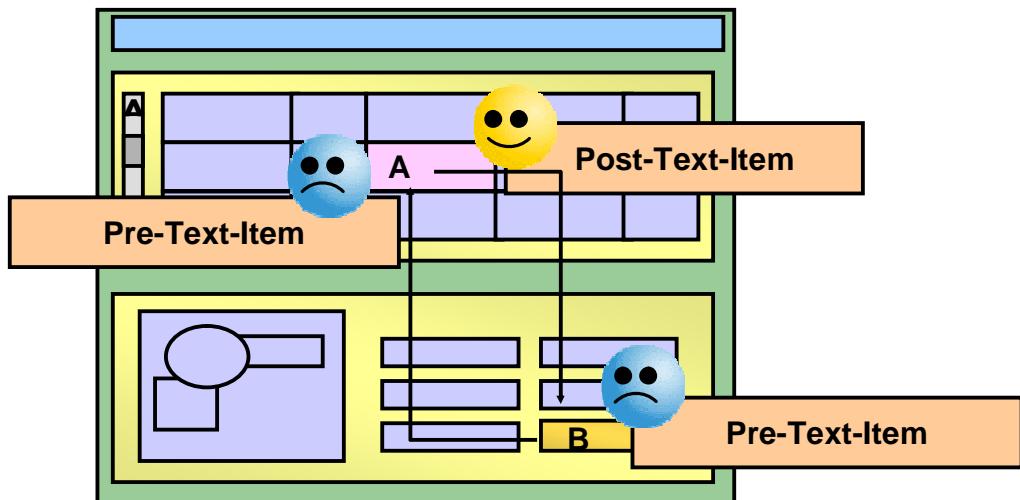
Using Pre- and Post-Triggers

Define Pre- and Post-Text-Item triggers at the item level, Pre- and Post-Block at the block level, and Pre- and Post-Form at the form level. Pre- and Post-Text-Item triggers fire only for text items.

Using Pre- and Post-Triggers (continued)

Trigger Type	Use to
Pre-Form	<ul style="list-style-type: none">Validate<ul style="list-style-type: none">UserTime of dayInitialize control blocksCall another form to display messages
Post-Form	<ul style="list-style-type: none">Perform housekeeping, such as erasing global variablesDisplay messages to the user before exit
Pre-Block	<ul style="list-style-type: none">Authorize access to the blockSet global variables
Post-Block	<ul style="list-style-type: none">Validate the last record that had input focusTest a condition and prevent the user from leaving the block
Pre-Record	<ul style="list-style-type: none">Set global variables
Post-Record	<ul style="list-style-type: none">Clear global variablesSet a visual attribute for an item as the user scrolls through a set of recordsPerform cross-field validation
Pre-Text-Item	<ul style="list-style-type: none">Derive a complex default valueRecord the previous value of a text item
Post-Text-Item	<ul style="list-style-type: none">Calculate or change item values

Avoiding the Navigation Trap



ORACLE®

20 - 17

Copyright © 2009, Oracle. All rights reserved.

Avoiding the Navigation Trap

You have seen that the Pre- and Post-navigation triggers fire during navigation, and when they fail, the internal cursor attempts to return to the current item (SYSTEM.cursor_item).

The graphic in the slide illustrates the navigation trap. This can occur when a Pre-navigation trigger (on the item to which the user is navigating) fails and attempts to return the logical cursor to its initial item. However, if the initial item has a Pre-Text-Item trigger that also fails, the cursor has nowhere to go, and a fatal error occurs.

Note: Be sure to code against navigation trigger failure.

Using Navigation Built-ins in Triggers

GO_FORM
GO_BLOCK
GO_ITEM
GO_RECORD
NEXT_BLOCK
NEXT_ITEM
NEXT_KEY
NEXT_RECORD

NEXT_SET
UP
DOWN
PREVIOUS_BLOCK
PREVIOUS_ITEM
PREVIOUS_RECORD
SCROLL_UP
SCROLL_DOWN



Using Navigation Built-ins in Triggers

You can initiate navigation programmatically by calling the built-in subprograms, such as GO_ITEM and PREVIOUS_BLOCK, from triggers.

Built-ins for Navigation	Function
GO_FORM	Navigates to an open form in a multiple form application
GO_BLOCK / ITEM / RECORD	Navigates to the indicated block, item, or record
NEXT_BLOCK / ITEM / KEY	Navigates to the next enterable block, item, or primary key item
NEXT / PREVIOUS_RECORD	Navigates to the first enterable item in the next or previous record
NEXT_SET	Fetches another set of records from the database and navigates to the first record that the fetch retrieves
UP , DOWN	Navigates to the instance of the current item in the previous/next record
PREVIOUS_BLOCK / ITEM	Navigates to the previous enterable block or item
SCROLL_UP / DOWN	Scrolls the block so that the records above the top visible one or below the bottom visible one are displayed

Calling Built-ins from Navigational Triggers

- When-New-Item-Instance

```
IF CHECKBOX_CHECKED('orders.order_mode') --Online  
THEN  
    orders.order_status := 4; --Credit order  
    GO_ITEM('orders.order_status');   
END IF;
```

- Pre-Text-Item

```
IF CHECKBOX_CHECKED('ORDERS.order_mode') --Online  
THEN  
    orders.order_status := 4; --Credit order  
    GO_ITEM('orders.order_status');   
END IF;
```

ORACLE®

20 - 19

Copyright © 2009, Oracle. All rights reserved.

Calling Built-ins from Navigational Triggers

You are not allowed to use a restricted built-in from within a trigger that fires during the navigation process (the Pre- and Post- triggers). This is because restricted built-ins perform some sort of navigation, and therefore, cannot be called until Forms navigation is complete. You can call restricted built-ins from triggers such as When-New-Item-Instance because that trigger fires after Forms has moved input focus to the new item.

The first example in the slide correctly uses the When-New-Item-Instance trigger, whereas the second incorrectly uses the Pre-Text-Item trigger and contains a restricted built-in.

Summary

In this lesson, you should have learned that:

- External navigation is visible to the user, whereas internal navigation occurs behind the scenes
- You can control navigation with properties of the form, block, or item
- Navigation triggers are those that fire before or after navigation
- You can use navigation built-ins in triggers (except for triggers that fire during navigation)

ORACLE®

20 - 20

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you learned the different methods of forcing visible (external) navigation and also the invisible events (internal).

- You can control navigation through the following properties, which you can set in the Property Palette, or programmatically by using `SET_[FORM | BLOCK | ITEM]_PROPERTY`:
 - Form module properties
 - Data block properties
 - Item properties
- Navigation triggers:
 - Pre- and Post- (fire during navigation—watch out for navigation trap)
 - When-New-<*object*>-Instance (fires after navigation)
- Navigation built-ins are available.
 - `GO_[FORM | BLOCK | RECORD | ITEM]`
 - `NEXT_[BLOCK | RECORD | ITEM | KEY | SET]`
 - `UP`
 - `DOWN`
 - `PREVIOUS_[BLOCK | RECORD | ITEM]`
 - `SCROLL_[UP | DOWN]`

Practice 20: Overview

This practice covers the following topics:

- Registering the bean area's JavaBean at form startup
- Setting properties on a Pluggable Java Component at form startup
- Executing a query at form startup
- Populating product images when navigating to each record of the ORDER_ITEMS block



Practice 20: Overview

In this practice, you provide a trigger to automatically perform a query, register a JavaBean, and set properties on a PJC at form startup. Also, you use When-New-<object>-Instance triggers to populate the Product_Image item as the operator navigates between records in the Orders form.

21

Transaction Processing

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

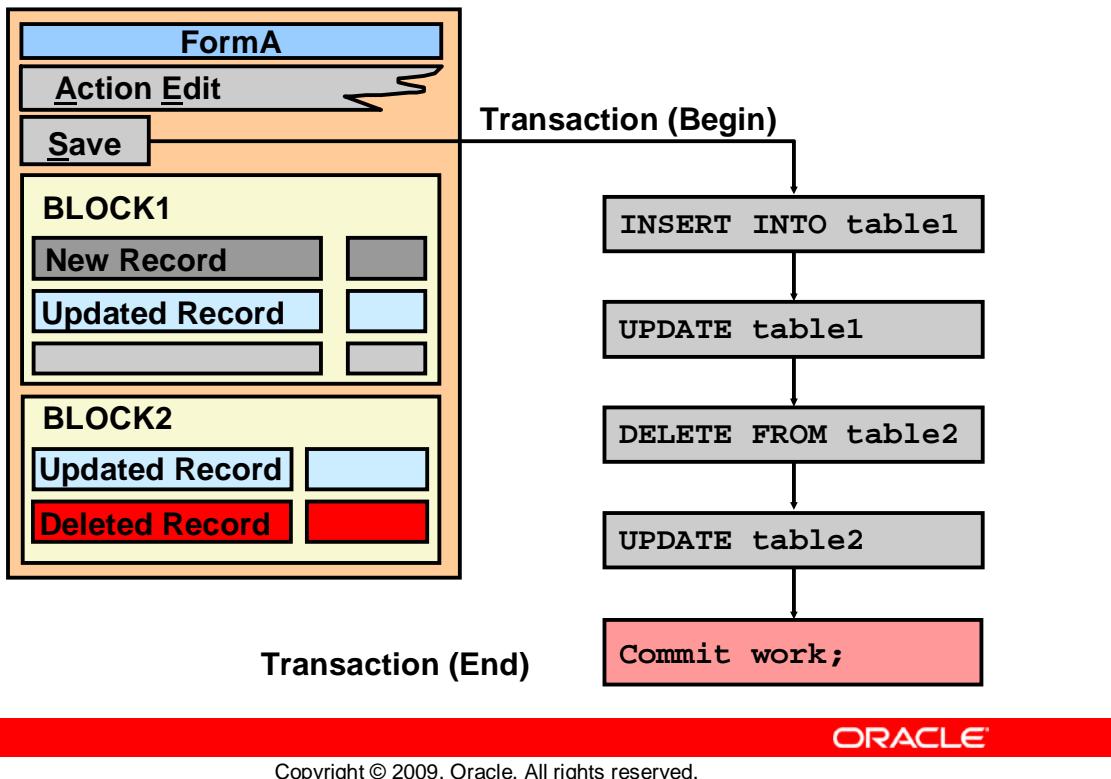
- Explain the process used by Forms to apply changes to the database
- Describe the commit sequence of events
- Supplement transaction processing
- Allocate sequence numbers to records as they are applied to tables
- Implement array data manipulation language (DML)



Lesson Aim

While applying a user's changes to the database, Forms Builder enables you to make triggers fire in order to alter or add to the default behavior. This lesson shows you how to build triggers that can perform additional tasks during a transaction.

Transaction Processing: Overview



21 - 3

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Transaction Processing: Overview

When Forms is asked to save the changes made in a form by the user, a process takes place involving events in the current database transaction. This process includes:

- **Default Forms transaction processing:** Applies the user's changes to the base tables
- **Firing transactional triggers:** Are needed to perform additional or modified actions in the saving process defined by the designer

When all of these actions are successfully completed, Forms commits the transaction, making the changes permanent. The scope of the transaction is the entire form module; you cannot separately commit blocks.

The graphics in the slide illustrate a form with two blocks. BLOCK1 contains a new record and an updated record. BLOCK2 contains an updated records and a deleted record. When the user clicks Save, the transaction proceeds as follows:

- The new record is inserted into the base table of BLOCK1 (called table1).
- The update on the updated row is performed on table1.
- The row is deleted from the base table of BLOCK2 (called table2).
- The update on the updated row is performed on table2.
- A commit is issued, committing the entire transaction.

Transaction Processing: Overview

Transaction processing includes two phases:

- Post:
 - Writes record changes to base tables
 - Fires transactional triggers
- Commit: Performs database commit

Errors result in:

- Rollback of the database changes
- Error message



Transaction Processing: Overview (continued)

The transaction process occurs as a result of either of the following actions:

- The user clicks Save or selects Action > Save from the menu, or clicks Save on the default Forms toolbar.
- The COMMIT_FORM built-in procedure is called from a trigger.

In either case, the process involves two phases, posting and committing:

Post: Posting writes the user's changes to the base tables, using implicit INSERT, UPDATE, and DELETE statements generated by Forms. The changes are applied in block sequence order as they appear in the Object Navigator at design time. For each block, deletes are performed first, followed by inserts, and updates. Transactional triggers fire during this cycle if defined by the designer.

The built-in procedure POST alone can invoke this posting process.

Commit: This performs the database commit, making the applied changes permanent and releasing locks.

Transaction Processing: Overview (continued)

Other events related to transactions include rollbacks, savepoints, and locking.

Rollbacks

Forms will roll back applied changes to a savepoint if an error occurs in its default processing, or when a transactional trigger fails.

By default, the user is informed of the error through a message, and a failing insert or update results in the record being redisplayed. The user can then attempt to correct the error before trying to save again.

Savepoints

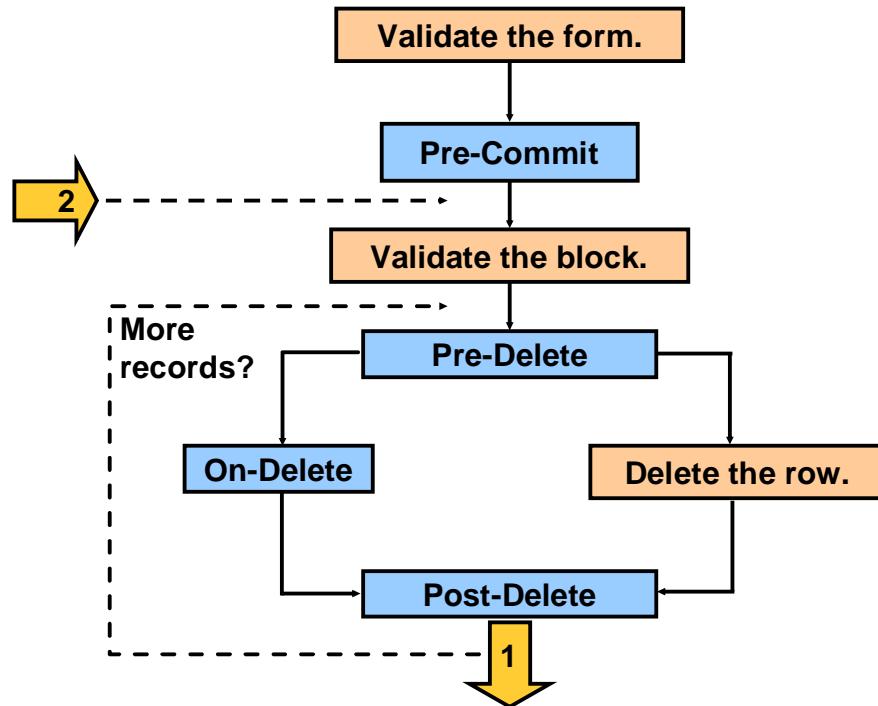
Forms issues savepoints in a transaction automatically, and will roll back to the latest savepoint if certain events occur. Generally, these savepoints are for Forms internal use, but certain built-ins, such as the `EXIT_FORM` built-in procedure, can request a rollback to the latest savepoint by using the `TO_SAVEPOINT` option.

Locking

When you update or delete base table records in a form application, database locks are automatically applied. Locks also apply during the posting phase of a transaction, and for DML statements that you explicitly use in your code.

Note: The SQL statements `COMMIT`, `ROLLBACK`, and `SAVEPOINT` cannot be called from a trigger directly. If encountered in a Forms program unit, Forms treats `COMMIT` as the `COMMIT_FORM` built-in, and `ROLLBACK` as the `CLEAR_FORM` built-in.

Commit Sequence of Events



ORACLE®

Commit Sequence of Events

The commit sequence of events (when the DML Array size is 1) is as follows, illustrated by the flowchart in the slide:

1. Validate the form.
2. Process the savepoint.
3. Fire the Pre-Commit trigger.
4. Validate the block (for all blocks in sequential order).
5. Perform the DML:

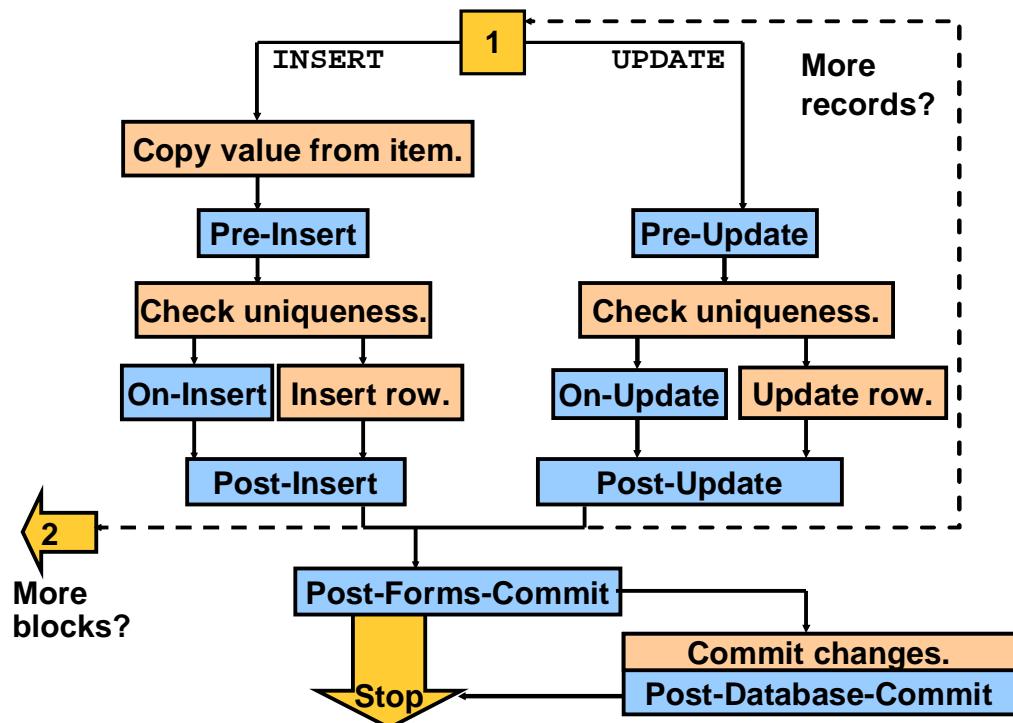
For all deleted records of the block (in reverse order of deletion):

- Fire the Pre-Delete trigger
- Delete the row from the base table or fire the On-Delete trigger
- Fire the Post-Delete trigger

For all inserted records of the block in sequential order:

- If it is an inserted record:
- Copy Value From Item
- Fire the Pre-Insert trigger
- Check the record uniqueness
- Insert the row into the base table or fire the On-Insert trigger
- Fire the Post-Insert trigger

Commit Sequence of Events



21 - 7

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Commit Sequence of Events (continued)

For all updated records of the block in sequential order:

- Fire the Pre-Update trigger
 - Check the record uniqueness
 - Update the row in the base table or fire the On-Update trigger
 - Fire the Post-Update trigger
6. Fire the Post-Forms-Commit trigger.

If the current operation is COMMIT, do the following:

7. Issue a SQL-COMMIT statement.
8. Fire the Post-Database-Commit trigger.

Characteristics of Commit Triggers

- Pre-Commit: It fires once if form changes are made or uncommitted changes are posted.
- Pre- and Post-DML
- On-DML: It fires per record, replacing the default DML on the row. Use the DELETE_RECORD, INSERT_RECORD, and UPDATE_RECORD built-ins.



Characteristics of Commit Triggers

You have already seen when commit triggers fire during the normal flow of commit processing. The table on the next page gives more detailed information about the conditions under which these triggers fire.

It is usually unnecessary to code commit triggers, and the potential for coding errors is high. Because of this, use commit triggers only if your application requires special processing at commit time.

One valid use of commit triggers is to distribute data manipulation language (DML) statements to underlying tables when you are performing DML on a block based on a join view. However, using a database instead of trigger may eliminate the need to define specialized Forms commit triggers for this purpose.

Note: If a commit trigger—except for the Post-Database-Commit trigger—fails, the transaction is rolled back to the savepoint that was set at the beginning of the current commit processing. This also means that uncommitted posts issued before the savepoint are not rolled back.

Characteristics of Commit Triggers

- Post-Forms-Commit: Fires once even if no changes are made
- Post-Database-Commit: Fires once even if no changes are made

Note: A commit-trigger failure causes a rollback to the savepoint.



Characteristics of Commit Triggers (continued)

Trigger	Characteristic
Pre-Commit	Fires once during commit processing, before base table blocks are processed; fires if there are changes to base table items in the form or if changes have been posted but not yet committed (always fires in case of uncommitted posts, even if there are no changes to post)
Pre- and Post-DML	Fire for each record that is marked for insert, update, or delete, just before or after the row is inserted, updated, or deleted in the database
On-DML	Fires for each record marked for insert, update, or delete when Forms would typically issue its INSERT, UPDATE, or DELETE statement, replacing the default DML statements (Include a call to the INSERT_RECORD, UPDATE_RECORD, or DELETE_RECORD built-in to perform default processing for these triggers.)
Post-Forms-Commit	Fires once during commit processing, after base table blocks are processed but before the SQL-COMMIT statement is issued; even fires if there are no changes to post or commit
Post-Database-Commit	Fires once during commit processing, after the SQL-COMMIT statement is issued; even fires if there are no changes to post or commit (This is also true for the SQL-COMMIT statement.)

Common Uses for Commit Triggers

Pre-Commit	Check user authorization; set up special locking
Pre-Delete	Journaling; implement foreign key delete rule
Pre-Insert	Generate sequence numbers; journaling; automatically generated columns; check constraints
Pre-Update	Journaling; implement foreign key update rule; autogenerated columns; check constraints

ORACLE®

21 - 10

Copyright © 2009, Oracle. All rights reserved.

Common Uses for Commit Triggers

After you know when a commit trigger fires, you should be able to choose the right commit trigger for the functionality that you want. To help you with this, the most common uses for commit triggers are mentioned in the table on the next page.

Where possible, implement functionality such as writing to a journal table, automatically supplying column values, and checking constraints in the server.

Note: Locking is also needed for transaction processing. You can use the On-Lock trigger if you want to amend the default locking of Forms.

Use DML statements only in commit triggers; otherwise, the DML statements are not included in the administration kept by Forms concerning commit processing. This may lead to unexpected and unwanted results.

Common Uses for Commit Triggers

On-Insert/Update/Delete	Replace default block DML statements
Post-Forms-Commit	Check complex multirow constraints
Post-Database-Commit	Test commit success; test uncommitted posts

ORACLE®

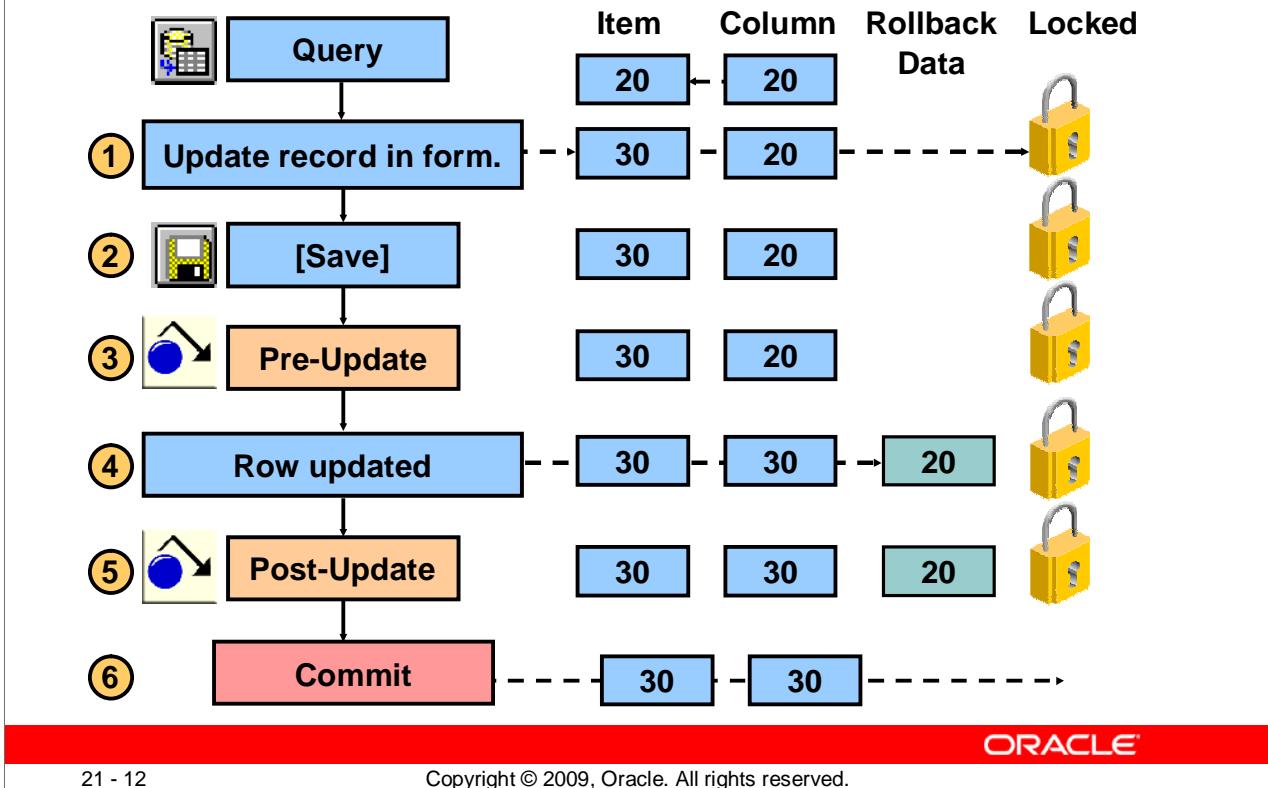
21 - 11

Copyright © 2009, Oracle. All rights reserved.

Common Uses for Commit Triggers (continued)

Trigger	Common Use
Pre-Commit	Checks user authorization; sets up special locking requirements
Pre-Delete	Writes to journal table; implements restricted or cascade delete
Pre-Insert	Writes to journal table; fills automatically generated columns; generates sequence numbers; checks constraints
Pre-Update	Writes to journal table; fills automatically generated columns; checks constraints; implements restricted or cascade update
Post-Delete, Post-Insert, Post-Update	Is seldom used
On-Delete, On-Insert, On-Update	Replaces default block DML statements; for example, to implement a pseudodelete or to update a join view
Post-Forms-Commit	Checks complex multirow constraints
Post-Database-Commit	Determines if the commit was successful; determines if there are posted uncommitted changes

Life of an Update



21 - 12

Copyright © 2009, Oracle. All rights reserved.

Life of an Update

To help you decide where certain trigger actions can be performed, consider an update operation as an example.

Example

The graphics in the slide show the price of a product being updated in a form. After the user queries the record, the following events occur:

1. The user updates the Price item from 20 to 30. This is now different from the corresponding database column (still 20.) By default, the row is locked on the base table.
2. The user saves the change, initiating the transaction process.
3. The Pre-Update trigger fires (if present). At this stage, the item and column are still different, because the update has not been applied to the base table. The trigger could compare the two values (for example, to make sure the new price is not lower than the existing one).
4. Forms applies the user's change to the database row. The item and column are now the same.

Life of an Update (continued)

5. The Post-Update trigger fires (if present). It is too late to compare the item against the column because the update has already been applied. However, the Oracle database retains the old column value as rollback data, so that a failure of this trigger reinstates the original value.
6. Forms issues the database commit, thus discarding the rollback data, releasing the lock, and making the changes permanent. The user receives the message Transaction Completed....

Performing Delete Validation

Pre-Delete trigger: Final checks before row deletion

```
DECLARE
    CURSOR c1 IS
        SELECT 'anything' FROM orders
        WHERE customer_id = :customers.customer_id;
BEGIN
    OPEN c1;
    FETCH c1 INTO :GLOBAL.dummy;
    IF c1%FOUND THEN
        CLOSE c1;
        MESSAGE('There are orders for this customer! ');
        RAISE form_trigger_failure;
    ELSE
        CLOSE c1;
    END IF;
END;
```

ORACLE®

21 - 14

Copyright © 2009, Oracle. All rights reserved.

Performing Delete Validation

Master detail blocks that are linked by a relation with the nonisolated deletion rule automatically prevent master records from being deleted in the form if matching detail rows exist.

You may, however, want to implement a similar check, as follows, when a deletion is applied to the database:

- A final check to ensure that no dependent detail rows have been inserted by another user since the master record was marked for deletion in the form (In an Oracle database, this is usually performed by a constraint or a database trigger.)
- A final check against form data, or checks that involve actions within the application

Note: If you select the Enforce data integrity check box in the Data Block Wizard, Forms Builder automatically creates the related triggers to implement constraints.

Performing Delete Validation (continued)

Example

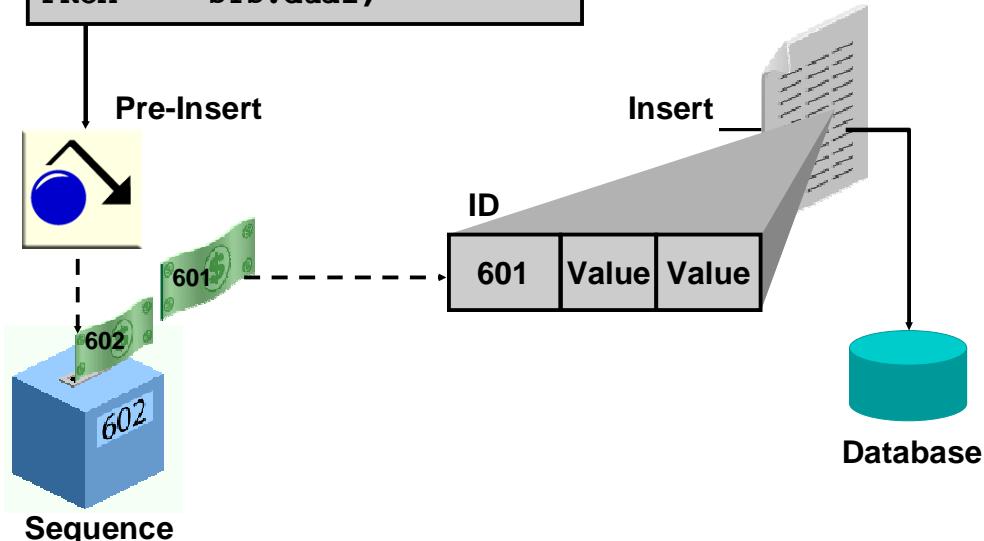
This Pre-Delete trigger on the CUSTOMER block of the CUSTOMERS form prevents deletion of rows if there are existing orders for the customer.

```
DECLARE
    CURSOR c1 IS
        SELECT 'anything' FROM orders
        WHERE customer_id = :customers.customer_id;
BEGIN
    OPEN c1;
    FETCH c1 INTO :GLOBAL.dummy;
    IF c1%FOUND THEN
        CLOSE c1;
        MESSAGE('There are orders for this customer!');
        RAISE form_trigger_failure;
    ELSE
        CLOSE c1;
    END IF;
END;
```

In the above example, a local PL/SQL variable can be used rather than a global variable. When the block terminates, a PL/SQL variable ceases to exist. A global variable exists for the session and retains any values that are set in code.

Assigning Sequence Numbers to Records

```
SELECT      orders_seq.NEXTVAL
INTO        :orders.order_id
FROM        SYS.dual;
```



Assigning Sequence Numbers to Records

You can assign default values for items from an Oracle sequence to automatically provide unique keys for records on their creation. However, if the user does not complete a record, the assigned sequence number is “wasted.”

An alternative method, illustrated in the slide graphic, is to assign unique keys to records from a Pre-Insert trigger, just before their insertion in the base table, by which time the user has completed the record and issued the Save.

Assigning unique keys in the posting phase can reduce:

- Gaps in the assigned numbers
- Data traffic on record creation, especially if records are discarded before saving

Assigning Sequence Numbers to Records (continued)

Example

The slide illustrates a Pre-Insert trigger on the ORDERS block that assigns an Order ID from the ORDERS_SEQ sequence. This Order ID is written to the Order_Id column when the row is subsequently inserted.

```
SELECT orders_seq.NEXTVAL  
INTO :orders.order_id  
FROM SYS.dual;
```

Note: The Insert Allowed and Keyboard Navigable properties on :orders.order_id should be set to No, so that the user does not enter an ID manually. The Required property should also be set to No, so that the user does not get a validation error when the value remains blank.

You can also assign sequence numbers from a table. If you use this method, two transactional triggers are usually involved:

- Use Pre-Insert to select the next available number from the sequence table (locking the row to prevent other users from selecting the same value), assign it to the Forms item, and increment the value in the sequence table by the required amount.
- Use Post-Insert to update the sequence table, recording the new upper value for the sequence.

Keeping an Audit Trail

- Write changes to nonbase tables.
- Gather statistics on applied changes.

Post-Insert example:

```
:GLOBAL.insert_tot :=  
TO_CHAR(TO_NUMBER(:GLOBAL.insert_tot)+1);
```



Keeping an Audit Trail

You may want to use the Post-<event> transactional triggers to record audit information about the changes applied to base tables. In some cases, this may involve duplicating inserts or updates in backup history tables, or recording statistics each time a DML operation occurs.

If the base table changes are committed at the end of the transaction, the audit information will also be committed.

Example

This Post-Update trigger writes the current record ID to the UPDATE_AUDIT table, along with a time stamp and the user who performed the update.

```
INSERT INTO update_audit (id, timestamp, who_did_it)  
VALUES ( :orders.order_id, SYSDATE, USER );
```

Example

This Post-Insert trigger adds to a running total of Inserts for the transaction, which is recorded in the INSERT_TOT global variable.

```
:GLOBAL.insert_tot :=  
TO_CHAR(TO_NUMBER(:GLOBAL.insert_tot)+1);
```

Testing the Results of Trigger DML

- SQL%FOUND
- SQL%NOTFOUND
- SQL%ROWCOUNT

```
UPDATE orders
   SET order_date = SYSDATE
 WHERE order_id = :orders.order_id;
IF SQL%NOTFOUND THEN
  MESSAGE('Record not found in database');
  RAISE form_trigger_failure;
END IF;
```

ORACLE®

21 - 19

Copyright © 2009, Oracle. All rights reserved.

Testing the Results of Trigger DML

When you perform DML in transactional triggers, you may need to test the results.

Unlike SELECT statements, DML statements do not raise exceptions when zero or multiple rows are processed. PL/SQL provides some useful attributes for obtaining results from the implicit cursor used to process the latest SQL statement (in this case, DML).

Obtaining Cursor Information in PL/SQL

PL/SQL Cursor Attribute	Values
SQL%FOUND	TRUE: Indicates > 0 rows processed FALSE: Indicates 0 rows processed
SQL%NOTFOUND	TRUE: Indicates 0 rows processed FALSE: Indicates > 0 rows processed
SQL%ROWCOUNT	Integer indicating the number of rows processed

Testing the Results of Trigger DML (continued)

Obtaining Cursor Information in PL/SQL (continued)

Example

This When-Button-Pressed trigger records the date of posting as the date ordered for the current Order record. If a row is not found by the UPDATE statement, an error is reported.

```
UPDATE orders
   SET order_date = SYSDATE
 WHERE order_id = :orders.order_id;
IF SQL%NOTFOUND THEN
   MESSAGE('Record not found in database');
   RAISE form_trigger_failure;
END IF;
```

Note: Triggers containing base table DML can adversely affect the usual behavior of your form because DML statements can cause some of the rows in the database to lock.

DML Statements Issued During Commit Processing

```
INSERT INTO base_table  (base_column, base_column,...)
VALUES                  (:base_item, :base_item, ...)
```

```
UPDATE    base_table
SET        base_column = :base_item, base_column =
           :base_item, ...
WHERE      ROWID = :ROWID
```

```
DELETE    FROM base_table
WHERE      ROWID = :ROWID
```

ORACLE®

21 - 21

Copyright © 2009, Oracle. All rights reserved.

DML Statements Issued During Commit Processing

If you have not altered default commit processing, Forms issues DML statements at commit time for each database record that is inserted, updated, or deleted.

```
INSERT INTO base_table  (base_column, base_column, ...)
VALUES      (:base_item,   :base_item,   ...)

UPDATE base_table
SET    base_column = :base_item, base_column = :base_item, ...
      WHERE ROWID = :ROWID

DELETE      FROM base_table
      WHERE ROWID = :ROWID
```

DML Statements Issued During Commit Processing

Rules:

- DML statements may fire database triggers.
- Forms uses and retrieves ROWID.
- The Update Changed Columns Only and Enforce Column Security properties affect UPDATE statements.
- Locking statements are not issued.

ORACLE®

21 - 22

Copyright © 2009, Oracle. All rights reserved.

DML Statements Issued During Commit Processing (continued)

Rules

- These DML statements may fire associated database triggers.
- Forms uses the ROWID construct only when the Key mode block property is set to Unique (or Automatic, the default). Otherwise, the primary key is used to construct the WHERE clause.
- If Forms successfully inserts a row in the database, it also retrieves the ROWID for that row.
- If the Update Changed Columns Only block property is set to Yes, only base columns with changed values are included in the UPDATE statement.
- If the Enforce Column Security block property is set to Yes, all base columns for which the current user has no update privileges are excluded from the UPDATE statement.
- Locking statements are not issued by Forms during default commit processing; they are issued as soon as a user updates or deletes a record in the form. If you set the Locking mode block property to Delayed, Forms waits to lock the corresponding row until commit time.

Overriding Default Transaction Processing

Additional transactional triggers:

Trigger	Do-the-Right-Thing Built-in
On-Check-Unique	CHECK_RECORD_UNIQUENESS
On-Column-Security	ENFORCE_COLUMN_SECURITY
On-Commit	COMMIT_FORM
On-Rollback	ISSUE_ROLLBACK
On-Savepoint	ISSUE_SAVEPOINT
On-Sequence-Number	GENERATE_SEQUENCE_NUMBER

Note: These triggers are meant to be used when connecting to data sources other than Oracle.

ORACLE®

21 - 23

Copyright © 2009, Oracle. All rights reserved.

Overriding Default Transaction Processing

You have already seen that some commit triggers can be used to replace the default DML statements that Forms issues during commit processing. You can use several other triggers to override the default transaction processing of Forms.

Transactional Triggers

All triggers that are related to accessing a data source are called *transactional triggers*. Commit triggers form a subset of these triggers. Other examples include triggers that fire during logon and logoff or during queries performed on the data source.

Overriding Default Transaction Processing

Transactional triggers for logging on and off:

Trigger	Do-the-Right-Thing Built-in
Pre-Logon	-
Pre-Logout	-
On-Logon	LOGON
On-Logout	LOGOUT
Post-Logon	-
Post-Logout	-

ORACLE®

21 - 24

Copyright © 2009, Oracle. All rights reserved.

Overriding Default Transaction Processing (continued)

Transactional Triggers for Logging On and Off

Trigger	Do-the-Right-Thing Built-In
Pre-Logon	-
Pre-Logout	-
On-Logon	LOGON
On-Logout	LOGOUT
Post-Logon	-
Post-Logout	-

Uses of Transactional Triggers

- Transactional triggers, except for the commit triggers, are primarily intended to access certain data sources other than Oracle.
- The logon and logoff transactional triggers can also be used with Oracle databases to change connections at run time.

Running Against Data Sources Other than Oracle

- Two ways to run against data sources other than Oracle:
 - Connecting with Open Gateway:
 - Cursor and Savepoint mode form module properties
 - Key mode and Locking mode block properties
 - Using transactional triggers:
 - Call 3GL programs
 - Database data block property

ORACLE®

21 - 25

Copyright © 2009, Oracle. All rights reserved.

Running Against Data Sources Other than Oracle

Two Ways to Run Against Data Sources Other than Oracle

- Use Oracle Transparent Gateway products.
- Write the appropriate set of transactional triggers.

Connecting with Open Gateway

When you connect to a data source other than Oracle with an Open Gateway product, you should be aware of these transactional properties:

- Cursor mode form module property
- Savepoint mode form module property
- Key mode block property
- Locking mode block property

You can set these properties to specify how Forms should interact with your data source. The specific settings depend on the capabilities of the data source.

Using Transactional Triggers

If no Open Gateway drivers exist for your data source, you must define transactional triggers. From these triggers, you must call 3GL programs that implement the access to the data source.

Running Against Data Sources Other than Oracle (continued)

Database Data Block Property

This block property identifies a block as a transactional control block; that is, a control block that should be treated as a base-table block. Setting this property to Yes ensures that transactional triggers will fire for the block, even though it is not a base-table block. If you set this property to Yes, you must define all On-Event transactional triggers, otherwise you will get an error during form generation.

Getting and Setting the Commit Status

- Commit status: Determines how the record will be processed
- SYSTEM.record_status:
 - NEW
 - INSERT (also caused by control items)
 - QUERY
 - CHANGED
- SYSTEM.block_status:
 - NEW (may contain records with status INSERT)
 - QUERY (also possible for control block)
 - CHANGED (block will be committed)
- SYSTEM.form_status: NEW, QUERY, CHANGED

ORACLE®

21 - 27

Copyright © 2009, Oracle. All rights reserved.

Getting and Setting the Commit Status

If you want to process a record in your form, it is often useful to know if the record is in the database or if it has been changed, and so on. You can use system variables and built-ins to obtain this information.

What Is the Commit Status of a Record?

The commit status of a record of a base table block determines how the record will be processed during the next commit process. For example, the record can be inserted, updated, or not processed at all.

Getting and Setting the Commit Status (continued)

The Four Values of **SYSTEM.record_status**

Value	Description
NEW	Indicates that the record has been created, but that none of its items have been changed yet (The record may have been populated by default values.)
INSERT	Indicates that one or more of the items in a newly created record have been changed (The record will be processed as an insert during the next commit process if its block has the CHANGED status; see below. Note that when you change a control item of a NEW record, the record status also becomes INSERT.)
QUERY	Indicates that the record corresponds to a row in the database, but that none of its base table items have been changed
CHANGED	Indicates that one or more base table items in a database record have been changed (The record will be processed as an update (or delete) during the next commit process.)

The Three Values of **SYSTEM.block_status**

Value	Description
NEW	Indicates that all records of the block have the status NEW (Note that a base table block with the status NEW may also contain records with the status INSERT caused by changing control items.)
QUERY	Indicates that all records of the block have the status QUERY if the block is a base table block (A control block has the status QUERY if it contains at least one record with the status INSERT.)
CHANGED	Indicates that the block contains at least one record with the status INSERT or CHANGED if the block is a base table block (The block will be processed during the next commit process. Note that a control block cannot have the status CHANGED.)

The Three Values of **SYSTEM.form_status**

Value	Description
NEW	Indicates that all blocks of the form have the status NEW
QUERY	Indicates that at least one block of the form has the status QUERY and all other blocks have the status NEW
CHANGED	Indicates that at least one block of the form has the status CHANGED

Getting and Setting the Commit Status

- System variables versus built-ins for commit status
- Built-ins for getting and setting commit status:
 - GET_BLOCK_PROPERTY
 - GET_RECORD_PROPERTY
 - SET_RECORD_PROPERTY

ORACLE®

21 - 29

Copyright © 2009, Oracle. All rights reserved.

Getting and Setting the Commit Status (continued)

The system variables SYSTEM.record_status and SYSTEM.block_status apply to the record and block where the cursor is located. You can use built-ins to obtain the status of other blocks and records.

Built-in	Description
GET_BLOCK_PROPERTY	Use the STATUS property to obtain the block status of the specified block.
GET_RECORD_PROPERTY	Use the STATUS property to obtain the record status of the specified record in the specified block.
SET_RECORD_PROPERTY	Set the STATUS property of the specified record in the specified block to one of the following constants: <ul style="list-style-type: none">• NEW_STATUS• INSERT_STATUS• QUERY_STATUS• CHANGED_STATUS

Getting and Setting the Commit Status

- Example: If the third record of the ORDERS block is a changed database record, set the status back to QUERY.
- Warnings:
 - Do not confuse commit status with validation status.
 - The commit status is updated during validation.

ORACLE®

21 - 30

Copyright © 2009, Oracle. All rights reserved.

Getting and Setting the Commit Status (continued)

Example

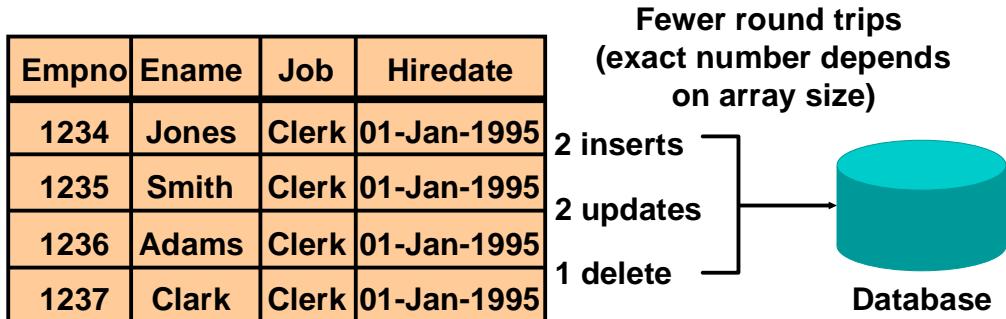
If the third record of the ORDERS block is a changed database record, set the status back to QUERY:

```
BEGIN
    IF GET_RECORD_PROPERTY( 3 , 'orders' , STATUS ) = 'CHANGED'
    THEN
        SET_RECORD_PROPERTY( 3 , 'orders' , STATUS ,
                             QUERY_STATUS );
    END IF;
END;
```

Array Processing

Array DML:

- Performs array inserts, updates, and deletes
- Vastly reduces network traffic



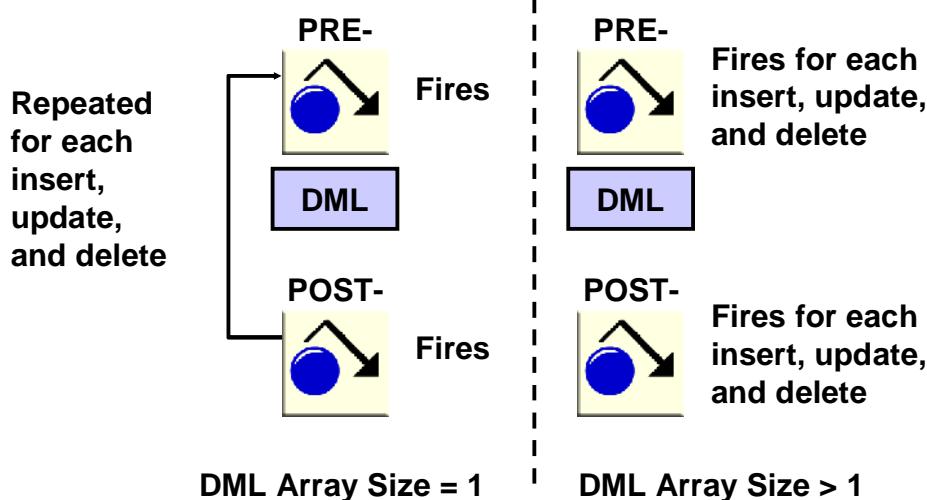
Array Processing

Array processing is an option in Forms Builder that alters the way records are processed. The default behavior of Forms is to process records one at a time. By enabling array processing, you can process groups of records at a time, reducing network traffic and thereby increasing performance. This is especially important in Web applications. With array processing, a structure (an array) containing multiple records is sent to or returned from the server for processing. This is illustrated in the slide, where five records are processed simultaneously.

Forms Builder supports both array fetch processing and array DML processing. For both querying and DML operations, you can determine the array size to optimize performance for your needs. This lesson focuses on array DML processing.

Array processing is available for query and DML operations for blocks based on tables, views, procedures, and subqueries; it is not supported for blocks based on transactional triggers.

Effect of Array DML on Transactional Triggers



Effect of Array DML on Transactional Triggers

With DML Array Size set to 1, the Pre-Insert, Pre-Update, and Pre-Delete triggers fire for each new, changed, and deleted record; the DML is issued, and the Post- trigger for that record fires. This is illustrated by the loop shown in the graphics at the left of the slide.

With DML Array Size set to greater than 1, the appropriate Pre- triggers fire for all of the new, changed, and deleted rows; all of the DML statements are issued, and all of the Post- triggers fire. This is illustrated by the graphics at the right of the slide.

If you change 100 rows and the DML Array Size is 20, you get 100 Pre- triggers, 5 arrays of 20 DML statements, and 100 Post- triggers.

Implementing Array DML

1. Enable the Array Processing option.
2. Specify a DML Array Size of greater than 1.
3. Specify block primary keys.

ORACLE®

21 - 33

Copyright © 2009, Oracle. All rights reserved.

Implementing Array DML

1. To set preferences:
 - Select Edit > Preferences.
 - Click the Runtime tab.
 - Select the Array Processing check box.
2. To set properties:
 - In the Object Navigator, select the Data Blocks node.
 - Double-click the Data Blocks icon to display the Property Palette.
 - Under the Advanced Database category, set the DML Array Size property to a number that represents the number of records in the array for array processing. You can also set this property programmatically.
3. When the DML Array Size property is greater than 1, you must specify the primary key. Key mode can still be unique.

The Oracle server uses the ROWID to identify the row, except after an array insert. If you update a record in the same session that you inserted it, the server locks the record by using the primary key.

Summary

In this lesson, you should have learned that:

- To apply changes to the database, Forms issues post and commit
- The commit is performed in a certain sequence
- You can supplement transaction processing with triggers



Summary

This lesson showed you how to build triggers that can perform additional tasks during the save stage of a current database transaction.

- Transactions are processed in two phases:
 - **Post:** Applies form changes to the base tables and fires transactional triggers
 - **Commit:** Commits the database transaction
- The commit sequence of events is as follows:
 1. Validate the form.
 2. Process savepoint.
 3. Fire Pre-Commit.
 4. Validate the block (performed for all blocks in sequential order).
 5. Perform the DML:
 - Delete records:** Fire Pre-Delete, delete row or fire On-Delete, fire Post-Delete trigger
 - Insert records:** Copy Value From Item, fire Pre-Insert, check record uniqueness, insert row or fire On-Insert, fire Post-Insert
 - Update records:** Fire Pre-Update, check record uniqueness, update row or fire On-Update, fire Post-Update
 6. Fire the Post-Forms-Commit trigger.

Summary

- You use the Pre-Insert trigger to allocate sequence numbers to records as they are applied to tables
- You check or change commit status:
 - GET_BLOCK_PROPERTY, [GET | SET]_RECORD_STATUS
 - :SYSTEM.[form | block | record]_status
- You use transactional triggers to override or augment default commit processing
- You reduce network roundtrips by setting the DML Array Size block property to implement array DML

ORACLE®

21 - 35

Copyright © 2009, Oracle. All rights reserved.

Summary (continued)

If the current operation is COMMIT, perform the following:

7. Issue a SQL-COMMIT statement.
 8. Fire the Post-Database-Commit trigger.
- Transactional triggers include:
 - Pre-Commit: Fires once if form changes are made or uncommitted changes are posted
 - [Pre | Post] – [Update | Insert | Delete]
 - On- [Update | Insert | Delete]: Fires per record, replacing default DML on row; should perform default functions with built-ins:
[UPDATE | INSERT | DELETE]_RECORD
 - DML statements issued during commit processing are based on base-table items; UPDATE and DELETE statements use ROWID by default.
 - Common uses for commit triggers: Check authorization, set up special locking requirements, generate sequence numbers, check complex constraints, and replace default DML statements issued by Forms.
 - Getting and setting the commit status: System variables and built-ins
 - Use array DML to reduce server roundtrips.

Practice 21: Overview

This practice covers the following topics:

- Automatically populating order IDs by using a sequence
- Automatically populating item IDs by adding the current highest order ID
- Customizing the commit messages in the Customers form
- Customizing the login screen in the Customers form



Practice 21: Overview

In this practice, you add transactional triggers to the Orders form to automatically provide sequence numbers to records at save time. You also customize commit messages and the login screen in the Customers form.

Writing Flexible Code

22

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe flexible code
- State the advantages of using system variables
- Identify the built-in subprograms that assist flexible coding
- Write code to reference objects:
 - By internal ID
 - Indirectly



Lesson Aim

In this lesson, you learn about the Forms Builder features that enable you to write code in a flexible, reusable way.

What Is Flexible Code?

Flexible code:

- Is reusable
- Is generic
- Avoids hard-coded object names
- Makes maintenance easier
- Increases productivity



What Is Flexible Code?

Flexible code is code that you can use again. It is often generic code that you can use in any form module in an application. It typically includes the use of system variables instead of hard-coded object names.

Why Write Flexible Code?

Writing flexible code gives you the following advantages:

- It is easier to maintain.
- It increases productivity.

Using System Variables for Current Context

- Input focus:
 - :SYSTEM.cursor_block
 - :SYSTEM.cursor_record
 - :SYSTEM.cursor_item
 - :SYSTEM.cursor_value

```
IF :SYSTEM.cursor_block = 'orders' THEN
    GO_BLOCK('order_items');

ELSIF :SYSTEM.cursor_block = 'order_items' THEN
    GO_BLOCK('inventories');

ELSIF :SYSTEM.cursor_block = 'inventories' THEN
    GO_BLOCK('orders');

END IF;
```

ORACLE®

22 - 4

Copyright © 2009, Oracle. All rights reserved.

Using System Variables for Current Context

In this lesson, you use the system variables that provide the current status of the record, the block, and the form, as well as system variables that return the current input focus location.

System Variables for Locating Current Input Focus

System Variable	Function
:cursor_block	The block that has the input focus
:cursor_record	The record that has the input focus
:cursor_item	The item and block that has the input focus
:cursor_value	The value of the item with the input focus

Example

The example in the slide shows code that can be put in a When-Button-Pressed trigger to enable users to navigate to another block in the form. It tests the current block name, then navigates depending on the result.

Note: Be sure to set the button's Mouse Navigate property to No; otherwise, :SYSTEM.cursor_block will always be the block on which the button is located.

Using System Variables for Current Context

- Trigger focus:
 - SYSTEM.trigger_block
 - SYSTEM.trigger_record
 - SYSTEM.trigger_item

ORACLE®

22 - 5

Copyright © 2009, Oracle. All rights reserved.

Using System Variables for Current Context (continued)

System Variables for Locating Trigger Focus

System Variable	Function
trigger_block	The block that the input focus was in when the trigger initially fired
trigger_record	The number of the record that Forms is processing
trigger_item	The block and item that the input focus was in when the trigger initially fired

Uses for Trigger Focus Variables

The variables for locating trigger focus are useful for navigating back to the initial block, record, and item after the trigger code completes. For example, the trigger code may navigate to other blocks, records, or items to perform actions upon them, but after the trigger fires, you may want the cursor to be in the same item instance that it was in originally. Because the navigation in the trigger occurs behind the scenes, the user will not even be aware of it.

Note: The best way to learn about system variables is to look at their values when a form is running. You can examine the system variables by using the Debugger.

Using System Variables to Determine the Current Status of the Form

System status variables:

- SYSTEM.record_status
- SYSTEM.block_status
- SYSTEM.form_status

When-Button-Pressed:

```
ENTER;  
IF :SYSTEM.block_status = 'CHANGED' THEN  
    COMMIT_FORM;  
END IF;  
CLEAR_BLOCK;
```

ORACLE®

22 - 6

Copyright © 2009, Oracle. All rights reserved.

Using System Variables to Determine the Current Status of the Form

You can use these system status variables presented in the previous lesson to write the code that performs one action for one particular status and a different action for another:

- SYSTEM.record_status
- SYSTEM.block_status
- SYSTEM.form_status

The example in the slide performs a commit before clearing a block if there are changes to commit within that block.

Using `GET_<object>_PROPERTY` Built-Ins

- `GET_APPLICATION_PROPERTY`
- `GET_FORM_PROPERTY`
- `GET_BLOCK_PROPERTY`
- `GET_RELATION_PROPERTY`
- `GET_RECORD_PROPERTY`
- `GET_ITEM_PROPERTY`
- `GET_ITEM_INSTANCE_PROPERTY`
- `GET_LOV_PROPERTY`
- `GET_RADIO_BUTTON_PROPERTY`
- `GET_MENU_ITEM_PROPERTY`
- `GET_CANVAS_PROPERTY`
- `GET_TAB_PAGE_PROPERTY`
- `GET_VIEW_PROPERTY`
- `GET_WINDOW_PROPERTY`

ORACLE®

22 - 7

Copyright © 2009, Oracle. All rights reserved.

Using `GET_<object>_PROPERTY` Built-Ins

Some of Forms Builder built-in subprograms provide the same type of run-time status information that built-in system variables provide.

`GET_APPLICATION_PROPERTY`

The `GET_APPLICATION_PROPERTY` built-in returns information about the current Forms application.

Example

The following example captures the username and the operating system information:

```
:GLOBAL.username := GET_APPLICATION_PROPERTY(USERNAME);  
:GLOBAL.o_sys := GET_APPLICATION_PROPERTY(OPERATING_SYSTEM);
```

Note: The `GET_APPLICATION_PROPERTY` built-in returns information about the Forms application running on the middle tier. If you require information about the client machine, you can use a JavaBean.

Using GET_<object>_PROPERTY Built-Ins (continued)

GET_BLOCK_PROPERTY

The GET_BLOCK_PROPERTY built-in returns information about a specified block.

Example

To determine the current record that is visible at the first (top) line of a block, use:

```
...GET_BLOCK_PROPERTY('blockname',top_record)...
```

GET_ITEM_PROPERTY

The GET_ITEM_PROPERTY built-in returns information about a specified item.

Example

To determine the canvas that the item with the input focus displays on, use:

```
DECLARE
    cv_name varchar2(30);
BEGIN
    cv_name := GET_ITEM_PROPERTY(:SYSTEM.cursor_item,item_canvas);
    ...

```

Using SET_<object>_PROPERTY Built-Ins

- SET_APPLICATION_PROPERTY
- SET_FORM_PROPERTY
- SET_BLOCK_PROPERTY
- SET_RELATION_PROPERTY
- SET_RECORD_PROPERTY
- SET_ITEM_PROPERTY
- SET_ITEM_INSTANCE_PROPERTY
- SET_LOV_PROPERTY
- SET_RADIO_BUTTON_PROPERTY
- SET_MENU_ITEM_PROPERTY
- SET_CANVAS_PROPERTY
- SET_TAB_PAGE_PROPERTY
- SET_VIEW_PROPERTY
- SET_WINDOW_PROPERTY

ORACLE®

22 - 9

Copyright © 2009, Oracle. All rights reserved.

Using SET_<object>_PROPERTY Built-Ins

SET_ITEM_INSTANCE_PROPERTY

The SET_ITEM_INSTANCE_PROPERTY built-in modifies the specified instance of an item in a block by changing the specified item property.

Example

The following example sets the visual attribute to VA_CURR for the current record of the current item:

```
SET_ITEM_INSTANCE_PROPERTY(:SYSTEM.cursor_item,  
    VISUAL_ATTRIBUTE, CURRENT_RECORD, 'va_curr');
```

SET_MENU_ITEM_PROPERTY

The SET_MENU_ITEM_PROPERTY built-in modifies the given properties of a menu item.

Example

To enable the save menu item in a file menu, use:

```
SET_MENU_ITEM_PROPERTY('file.save', ENABLED, PROPERTY_TRUE);
```

SET_<object>_PROPERTY Built-Ins (continued)

SET_TAB_PAGE_PROPERTY

The SET_TAB_PAGE_PROPERTY built-in sets the tab page properties of the specified tab canvas page.

Example

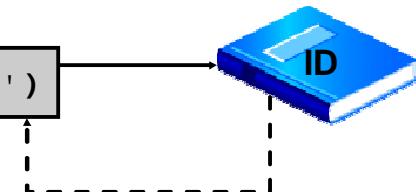
To enable tab_page_1, if it is already disabled, use:

```
DECLARE
    tbpg_id TAB_PAGE;
BEGIN
    tbpg_id := FIND_TAB_PAGE('tab_page_1');
    IF GET_TAB_PAGE_PROPERTY(tbpg_id, enabled) = 'FALSE'
    THEN
        SET_TAB_PAGE_PROPERTY(tbpg_id, enabled, property_true);
    END IF;
END;
```

Referencing Objects by Internal ID

Finding the object ID:

```
lov_id := FIND_LOV('my_lov')
```



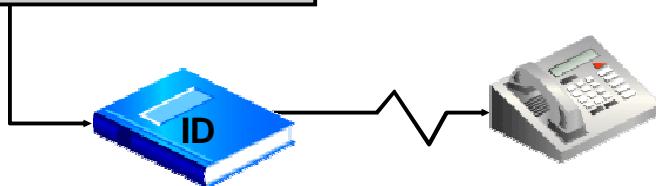
Referencing an object by ID:

```
...SHOW_LOV(lov_id)
```



Referencing an object by name:

```
...SHOW_LOV('my_lov')
```



ORACLE®

Referencing Objects by Internal ID

Forms Builder assigns an ID to each object that you create. An object ID is an internal value that is never displayed. You can get the ID of an object by calling the built-in FIND_ subprogram appropriate for the object. The FIND_ subprograms require a fully qualified object name as a parameter. For instance, when referring to an item, use `<block_name>. <item_name>`. This is similar to looking up a number in a telephone book, as illustrated by the top graphic in the slide.

The return values of the FIND_ subprograms (the object IDs) are of a specific type. The types for object IDs are predefined in Forms Builder. There is a different type for each object.

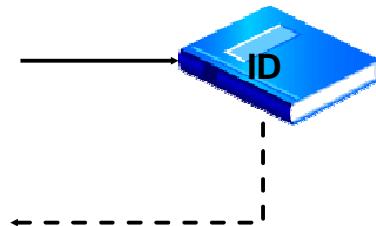
Reasons for Using Object IDs

- Improving performance (Forms looks up the object only once when you initially call the FIND_ subprogram to get the ID. When you refer to an object by name in a trigger, Forms must look up the object ID each time.)
- Writing more generic code
- Testing whether an object exists (using the ID_NULL function and FIND_object)

The middle graphic in the slide illustrates using the ID to reference an object, which is similar to dialing a phone. The bottom graphic shows that referencing by name is similar to looking up the phone number every time before you dial it.

Using FIND_ Built-Ins

- FIND_ALERT
- FIND_BLOCK
- FIND_CANVAS
- FIND_EDITOR
- FIND_FORM
- FIND_ITEM
- FIND_LOV
- FIND_RELATION
- FIND_VIEW
- FIND_WINDOW



ORACLE®

22 - 12

Copyright © 2009, Oracle. All rights reserved.

Using FIND_ Built-Ins

The following table lists some of the FIND_ subprograms, along with the object classes that use them and the return types they produce:

Object Class	Subprogram	Return Type
Alert	FIND_ALERT	ALERT
Block	FIND_BLOCK	BLOCK
Canvas	FIND_CANVAS	CANVAS
Editor	FIND_EDITOR	EDITOR
Form	FIND_FORM	FORMMODULE
Item	FIND_ITEM	ITEM
LOV	FIND_LOV	LOV
Relation	FIND_RELATION	RELATION
View	FIND_VIEW	VIEWPORT
Window	FIND_WINDOW	WINDOW

The graphic shows that each of these built-ins retrieves the ID of the object (in the same way as looking up a telephone number).

Using Object IDs

- Declare a PL/SQL variable of the same data type.
- Use the variable for any later reference to the object.
- Use the variable within the current PL/SQL block only.

ORACLE®

22 - 13

Copyright © 2009, Oracle. All rights reserved.

Using Object IDs

To use an object ID, you must first assign it to a variable. You must declare a variable of the same type as the object ID.

The following example uses the FIND_ITEM built-in to assign the ID of the item that currently has input focus to the id_var variable.

After you assign an object ID to a variable in a trigger or PL/SQL program unit, you can use that variable to reference the object, rather than referring to the object by name:

```
DECLARE
    id_var ITEM;
BEGIN
    id_var := FIND_ITEM(:SYSTEM.cursor_item);
    .
    .
    .
END;
```

Example: Using Object IDs

```
DECLARE
    item_var ITEM;
BEGIN
    item_var := FIND_ITEM(:SYSTEM.cursor_item);
    SET_ITEM_PROPERTY(item_var,position,30,55);
    SET_ITEM_PROPERTY(item_var,prompt_text,
        'Current');
END;
```



Example: Using Object IDs

The following lines of code show that you can pass either an item name or an item ID to the SET_ITEM_PROPERTY built-in subprogram. The following calls are logically equivalent:

```
SET_ITEM_PROPERTY('orders.order_id',position,50,35);
SET_ITEM_PROPERTY(id_var,position,50,35);
```

You can use either object IDs or object names in the same argument list, provided that each individual argument refers to a distinct object.

You cannot, however, use an object ID and an object name to form a fully qualified object_name (blockname.itemname). The following call is illegal:

```
GO_ITEM(block_id.'item_name');
```

Note: Use the FIND_ built-in subprograms only when referring to an object more than once in the same trigger or PL/SQL program unit.

Increasing the Scope of Object IDs

- A PL/SQL variable has limited scope.
- A `.id` extension:
 - Broadens the scope
 - Converts to a numeric format
 - Enables assignment to a global variable
 - Converts back to the object data type

ORACLE®

22 - 15

Copyright © 2009, Oracle. All rights reserved.

Increasing the Scope of Object IDs

You have seen how object IDs are referenced within the trigger or program unit by means of PL/SQL variables. You can reference these PL/SQL variables only in the current PL/SQL block; however, you can increase the scope of an object ID.

To reference an object ID outside the initial PL/SQL block, you need to convert the ID to a numeric format using an `.id` extension for your declared PL/SQL variable, then assign it to a global variable.

Example

The following example of trigger code assigns the object ID to a local PL/SQL variable (`item_var`) initially, then to a global variable (`:GLOBAL.item`):

```
DECLARE
    item_var ITEM;
BEGIN
    item_var := FIND_ITEM(:SYSTEM.cursor_item);
    :GLOBAL.item := item_var.id;
END;
```

Increasing the Scope of Object IDs (continued)

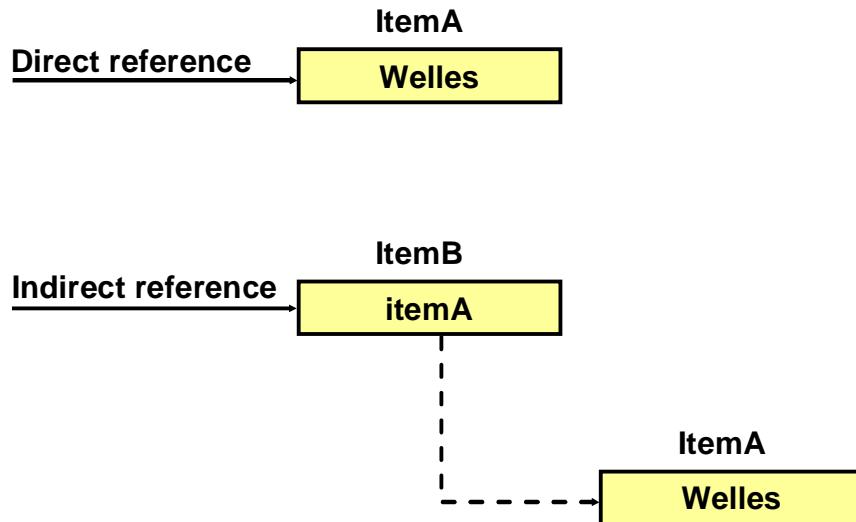
You can pass the global variable around within the application. To be able to reuse the object ID, you need to convert it back to its original data type.

Example

The following example shows the conversion of the global variable back to its original PL/SQL variable data type:

```
DECLARE
    item_var ITEM;
BEGIN
    item_var.id := TO_NUMBER(:GLOBAL.item);
    GO_ITEM(item_var);
END;
```

Referencing Items Indirectly



ORACLE®

22 - 17

Copyright © 2009, Oracle. All rights reserved.

Referencing Items Indirectly

By referencing items indirectly, you can write more generic, reusable code. Using variables instead of actual item names, you can write a PL/SQL program unit to use any item whose name is assigned to the indicated variable.

You can reference items indirectly with the NAME_IN and COPY built-in subprograms.

Note: Use indirect referencing when you create procedures and functions in a library module because direct references cannot be resolved.

Using the NAME_IN Function for Indirect References

- The NAME_IN function returns:
 - The contents of a variable
 - The character string
- Use conversion functions for NUMBER and DATE.

ORACLE®

22 - 18

Copyright © 2009, Oracle. All rights reserved.

Using the NAME_IN Function for Indirect References

The NAME_IN function returns the contents of an indicated variable. The following statements are equivalent. The first one uses a direct reference to customer.name, whereas the second uses an indirect reference:

```
IF :customers.cust_last_name = 'Welles'...
```

In a library, you can avoid this direct reference by using:

```
IF NAME_IN('customers.cust_last_name') = 'Welles'...
```

The return value of NAME_IN is always a character string. To use NAME_IN for a date or number item, convert the string to the desired data type with the appropriate conversion function. For instance:

```
date_var := TO_DATE(NAME_IN('orders.order_date'));
```

Using the COPY Procedure for Indirect References

The COPY procedure enables:

- Direct copy:

```
COPY( 'Welles' , 'customers.cust_last_name' );
```

- Indirect copy:

```
COPY( 'Welles' ,NAME_IN( 'GLOBAL.customer_name_item' ));
```

ORACLE®

22 - 19

Copyright © 2009, Oracle. All rights reserved.

Using the COPY Procedure for Indirect References

The COPY built-in assigns an indicated value to an indicated variable or item. Unlike the standard PL/SQL assignment statement, using the COPY built-in enables you to indirectly reference the item whose value is being set. The first example in the slide shows copying using a direct reference to the form item.

Using COPY with NAME_IN

Use the COPY built-in subprogram with the NAME_IN built-in to indirectly assign a value to an item whose name is stored in a global variable, as in the second example in the slide.

Summary

In this lesson, you should have learned that:

- Flexible code is reusable, generic code that you can use in any form module in an application
- With system variables, you can:
 - Perform actions conditionally based on the current location:
`(SYSTEM.CURSOR_[RECORD | ITEM | BLOCK])`
 - Use the value of an item without knowing its name:
`(SYSTEM.CURSOR_VALUE)`
 - Navigate to the initial location after a trigger completes:
`(SYSTEM.TRIGGER_[RECORD | ITEM | BLOCK])`
 - Perform actions conditionally based on commit status:
`SYSTEM.[RECORD | BLOCK | FORM]_STATUS`



Summary

Use the following to write flexible code:

- System variables:
 - To avoid hard-coding object names
 - To return information about the current state of the form

Summary

- The [GET | SET]_<object>_PROPERTY built-ins are useful in flexible coding
- Code that references objects is more efficient and generic:
 - By internal ID: Use FIND_<object> built-ins.
 - Indirectly: Use COPY and NAME_IN built-ins.

ORACLE®

22 - 21

Copyright © 2009, Oracle. All rights reserved.

Summary (continued)

- GET_<object>_PROPERTY built-ins, to return current property values for Forms Builder objects
- Object IDs, to improve performance
- Indirect referencing, to allow form module variables to be referenced in library and menu modules

Practice 22: Overview

This practice covers the following topics:

- Populating product images only when the image item is displayed
- Modifying the When-Button-Pressed trigger of Image_Button in order to use object IDs instead of object names
- Writing generic code to print out the names of the blocks in a form



Practice 22: Overview

In this practice, you use properties and variables in the Orders form to provide flexible use of its code. You also make use of object IDs.

Sharing Objects and Code

23

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe the various methods for reusing objects and code
- Inherit properties from property classes
- Group related objects for reuse
- Explain the inheritance symbols in the Property Palette
- Reuse objects from an object library
- Reuse PL/SQL code



Lesson Aim

Forms Builder includes some features specifically for object and code reuse. In this lesson, you learn how to define property classes that enable you to easily apply sets of properties to different objects. You learn how to share objects between form modules using the object library and how to share code using a PL/SQL library.

Benefits of Reusable Objects and Code

- Increases productivity
- Decreases maintenance
- Increases modularity
- Maintains standards
- Improves application performance

ORACLE®

23 - 3

Copyright © 2009, Oracle. All rights reserved.

Benefits of Reusable Objects and Code

When you are developing applications, you should share and reuse objects and code wherever possible in order to:

- **Increase productivity:** You can develop applications much more effectively and efficiently if you are not trying to “start over” each time you write a piece of code. By sharing and reusing frequently used objects and code, you can cut down development time and increase productivity.
- **Decrease maintenance:** By creating applications that use or call the same object or piece of code several times, you can decrease maintenance time.
- **Increase modularity:** Sharing and reusing code increases the modularity of your applications.
- **Maintain standards:** You can maintain standards by reusing objects and code. If you create an object once and copy it again and again, you do not run the risk of introducing minor changes. In fact, you can create a set of standard objects and some pieces of standard code and use them as a starting point for all of your new form modules.

Benefits of Reusable Objects and Code (continued)

- **Improve application performance:** When Forms Services communicates the user interface to the Forms Client, it sends metadata about the items on the form. This metadata includes values of properties that differ from the default. After an item is defined, the metadata about the next item includes only those properties that differ from the previous item. This is referred to as message diffing. Promoting similarities among items by using the methods of object reuse presented in this lesson improves the efficiency of message diffing and thus decreases network traffic and increases performance.

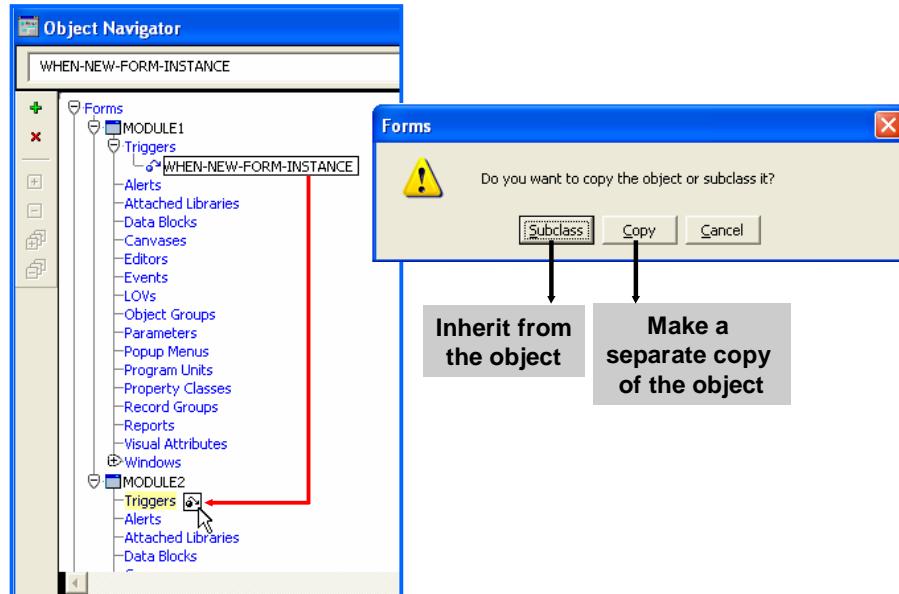
Promoting Similarities Among Objects

One of the easiest ways a developer can increase the efficiency of network performance through message diffing is by using consistent standards for all objects within an application. Items of different types should at least have common values for common or shared properties.

To maximize reuse, the developer should apply the following guidelines in the order shown:

- **Accept default properties as much as possible:** If the properties are not overwritten for each object, the value for common properties will be the same regardless of the object type, except for position and size.
- **Use SmartClasses to describe an object:** If, because of design standards, the use of default properties is not a viable option, the subclassing of objects from a set of SmartClasses ensures that the development standards are being met. It also forces a high degree of property sharing across widgets. Items of the same type will then have (unless overridden) the same properties and, therefore, will be able to share properties more effectively. You will learn about SmartClasses in this lesson.
- **Use sets of visual attributes:** If SmartClasses are not being used to enforce standards and common properties, use sets of partial visual attributes to enforce a common set of properties across objects of different types (for example, font, font size, foreground color, background color, and so on). These sets of visual attributes can be defined as Property Classes, as explained in the following slides.

Copying and Subclassing Objects and Code



ORACLE®

23 - 5

Copyright © 2009, Oracle. All rights reserved.

Copying and Subclassing Objects and Code

You can copy or subclass objects:

- Between modules, by dragging objects between the modules in the Object Navigator
- Within a single module by selecting the object in the Object Navigator, pressing Ctrl, and dragging it to create the new object

When you drag objects, a dialog box appears that asks whether you want to copy or subclass the object, as shown in the slide.

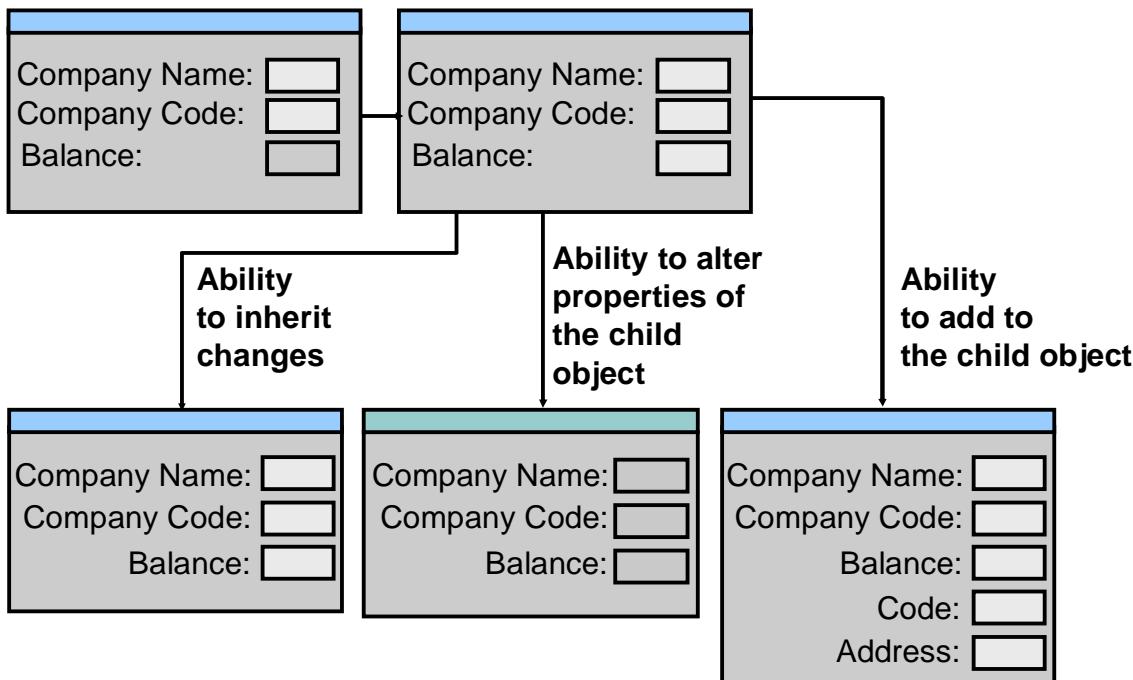
Copying an Object

Copying an object creates a separate, unique version of that object in the target module. Any objects owned by the copied object are also copied.

Points to Remember

- Use copying to export the definition of an object to another module.
- Changes made to a copied object in the source module do not affect the copied object in the target module.

Using Subclassing



Using Subclassing

Subclassing is an object-oriented term that refers to the following capabilities:

- Inheriting the characteristics of a base class (Inheritance)
- Overriding the properties of the base class (Specialization)

With subclassing, you can make an exact copy, and then alter some of the properties of the object if desired. If you change the parent class, the changes also apply to those properties of the subclassed object that you have not altered. However, any properties that you override remain overridden. This provides a powerful object inheritance model.

When you subclass a data block, you can:

- Change the structure of the parent, automatically propagating the changes to the child
- Add or change properties to the child to override the inheritance

When you subclass a data block, you cannot:

- Delete items from the child
- Change the order of items in the child
- Add items to the child unless you add them to the end

Using Subclassing (continued)

Ability to Add to an Object

You can create an exact copy of an object, and you can add to the subclassed object. For example, you can add additional items to the end of a subclassed block.

Ability to Alter Properties

With subclassing, you can make an exact copy and then alter the properties of some objects. If you change the parent class, the changes also apply to the properties of the subclassed object that you have not altered. However, any properties that you override remain overridden.

Ability to Inherit Changes

When you change the properties of a parent object, all child objects inherit those properties if they are not already overridden.

The child inherits changes:

- Immediately, if the parent and child objects are in the same form
- When you reload the form containing a child object

Ability to Reinherit

If you make changes to the child object to override properties of the parent object, you can click the Inherit button to reinherit the property from the parent object.

Property Palette icons: Enable you to identify inherited or overridden properties

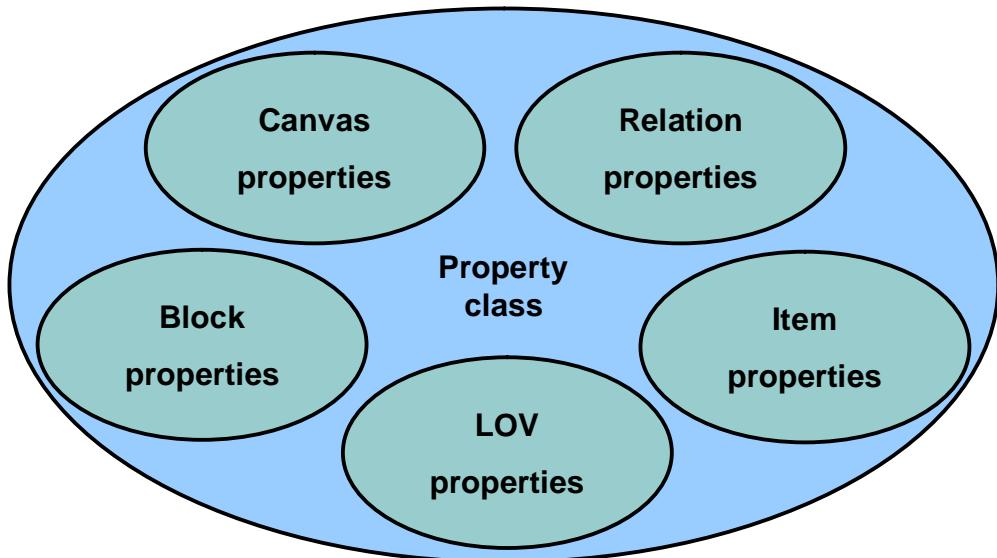
Property Palette Icon	Meaning
Circle	The value for the property is the default.
Square	The value for the property was changed from the default.
Arrow	The value for the property was inherited.
Arrow with red X	The value for the property was inherited but overridden (variant property).

Example

The graphics in the slide show a parent object at the top (both in its original form and after a change has been made,) with three subclassed objects at the bottom. The properties of the parent have been changed since the subclassing was first done; the background color of the Balance item has been changed from gray to white. Subclassing principles are illustrated as follows:

- The subclassed object at the bottom left shows that if you change the parent, the child inherits the changes; its Balance object has inherited the changed background color.
- The subclassed object in the middle shows that you can change the property values in the child to override those of the parent; the background color of all the items is changed to gray in the child object.
- The subclassed object at the bottom-right illustrates that you can add items to the end of the child object.

What Are Property Classes?



ORACLE®

23 - 8

Copyright © 2009, Oracle. All rights reserved.

What Are Property Classes?

A *property class* is a named object that contains a list of properties and their settings.

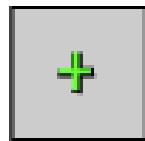
Use property classes to:

- Increase productivity by setting standard or frequently used values for common properties and associating them with several Forms Builder objects. You can use property classes to define standard properties not just for one particular object, but for several at a time. This results in increased productivity, because it eliminates the time spent on setting identical properties for several objects. The slide graphic illustrates that you can apply a single property class to a variety of objects.
- Improve network performance by increasing the efficiency of message diffing

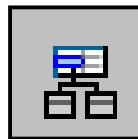
Example

You can define a property class with the following properties: Access Key, Automatic Refresh, and Background Color. If you apply that property class to a button, a canvas, and an LOV, all three objects use the Background Color from the property class, because all three objects have that property. However, only the button inherits Access Key, and only the LOV inherits Automatic Refresh.

Creating a Property Class



Add Property



Property Class



Delete Property

ORACLE®

Creating a Property Class

When you create a property class, you have all the properties from every Forms Builder object available. You choose the properties and their values to include in the property class. You can create a property class in two ways:

- Using the Create button in the Object Navigator
- Using the Create Property Class button

How to Create a Property Class from the Object Navigator

1. Select the Property Class node.
2. Click Create. A new property class entry is displayed.
3. In the Property Palette, use the Add Property button (shown in the slide) to add the required properties, and then set their values.

How to Create a Property Class from the Property Palette

1. In the Object Navigator, double-click the icon of the object whose properties you want to copy into a property class.
2. In the Property Palette, select the properties you want to copy into a property class.
3. Click the Property Class button (shown in the slide). An information alert is displayed.
4. Use the Object Navigator to locate the property class and change its name.

Creating a Property Class (continued)

Adding a Property

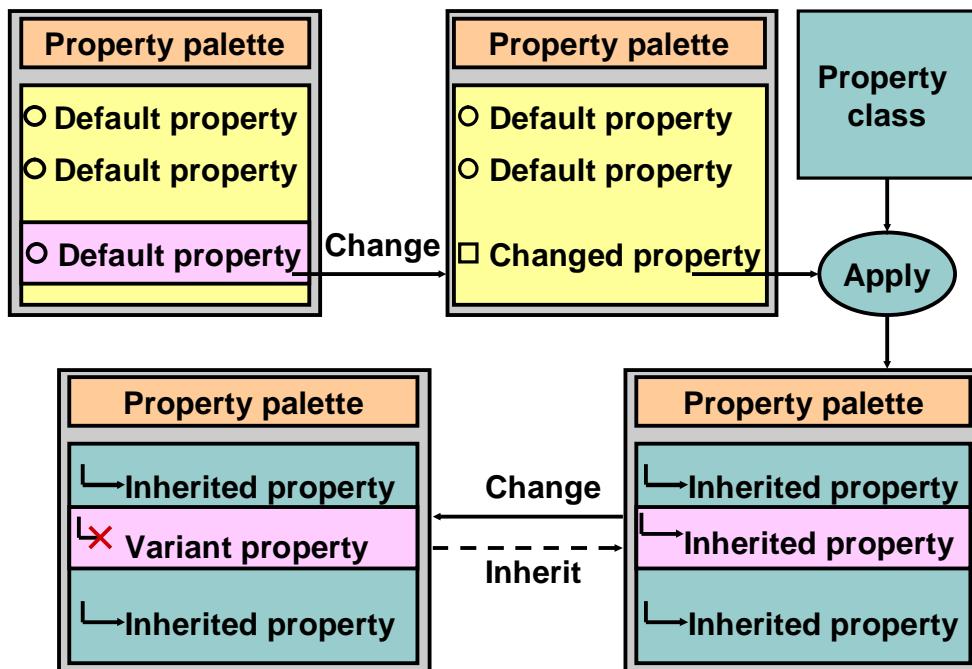
After you create a property class, you can add a property by clicking the Add Property button (shown in the slide) and selecting a property from the list. Set the value for that property using the Property Palette.

You can also use the Copy Properties button and the Paste Properties button to add several properties at a time to a property class.

Deleting a Property

You can remove properties from a property class using the Delete Property button (shown in the slide).

Types of Properties



Types of Properties

After you create a property class and add properties, you can apply its properties to an object by using the Subclass Information property in the Property Palette.

What Is an Inherited Property?

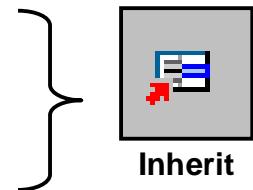
An *inherited property* is one that takes its value from the property class that you associated with the object. An inherited property is displayed with a bent arrow to the left of the property name.

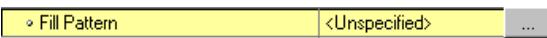
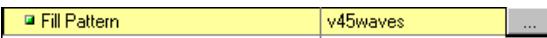
What Is a Variant Property?

A *variant property* is one whose value has been modified from the inherited value that comes from the property class associated with the object. You can override the setting of any inherited property to make that property variant. Variant properties are displayed with a red cross over an arrow.

Inheriting from a Property Class

- Set the Subclass Information property.
- Convert an inherited property to a variant property.
- Convert a variant property to an inherited property.
- Convert a changed property to a default property.



Inherited property	 Background Color r100g0b50 ...
Variant property	 Background Color r100g100b50 ...
Default property	 Fill Pattern <Unspecified> ...
Changed property	 Fill Pattern v45waves ...

ORACLE®

Inheriting from a Property Class

How to Inherit Property Values from a Property Class

1. In the Object Navigator, double-click the object to which you want to apply the properties from the property class.
2. Click the Subclass Information property in the Property Palette.
3. Select the property class whose properties you want to use. The object takes on the values of that property class. Inherited properties are displayed with an arrow.

Converting an Inherited Property to a Variant Property

To convert an inherited property to a variant property, simply enter a new value over the inherited one.

Converting a Variant Property to an Inherited Property

To convert a variant property to an inherited property, click the Property Palette's Inherit button (shown in the slide).

Converting a Changed Property to a Default Property

You can also use the Inherit button to revert a changed property to its default.

The screenshots in the slide show how the different types of properties appear in the Property Palette: inherited, variant, default, and changed.

What Are Object Groups?

Object groups:

- Are logical containers
- Enable you to:
 - Group related objects
 - Copy multiple objects in one operation



What Are Object Groups?

An *object group* is a logical container for a set of Forms Builder objects.

You define an object group when you want to:

- Package related objects for copying or subclassing in another module
- Bundle numerous objects into higher-level building blocks that you can use again in another application

Example

Your application can include an appointment scheduler that you want to make available to other applications. You can package the various objects in an object group and copy the entire bundle in one operation.

Creating and Using Object Groups

- Blocks include:
 - Items
 - Item-level triggers
 - Block-level triggers
 - Relations
- Object groups cannot include other object groups.
- Deleting an object group does not affect the objects.
- Deleting an object affects the object group.

ORACLE®

23 - 14

Copyright © 2009, Oracle. All rights reserved.

Creating and Using Object Groups

How to Create an Object Group

1. Click the Object Group node in the Object Navigator.
2. Click the Create button. A new object group entry is displayed.
3. Rename the new object group.
4. Select the form module and, from the menu, select View > Expand All.
5. Control-click all the objects of one type that you want to include in the object group.
6. Drag the selected objects into the new object group entry. The objects are displayed as object group children.
7. Repeat steps 5 and 6 for different object types.

The objects in the object group are still displayed in their usual position in the Object Navigator, as well as within the object group. The objects in the object group are not duplicates, but pointers to the source objects.

Creating and Using Object Groups (continued)

Things to Consider when Using Object Groups

- Including a block in an object group also includes its items, the item-level triggers, the block-level triggers, and the relations. You cannot use any of these objects in an object group without the block.
- It is not possible to include another object group.
- Deleting an object from a module automatically deletes the object from the object group.
- Deleting an object group from a module does not delete the objects it contains from the module.

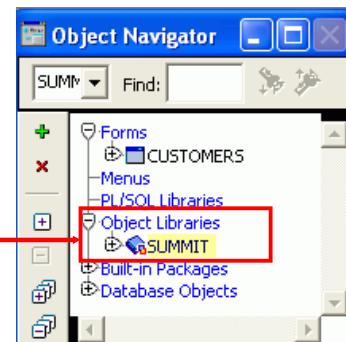
Subclass Information Dialog Box

The Subclass Information property of a form object shows a dialog box that provides information about the origins of the object. You can see whether an object is local to the form document or foreign to it. If the object is foreign to the current form, the Module field shows the module from which the object originates. The original object name is shown in the Object Name field.

What Are Object Libraries?

An Object library:

- Is a convenient container of objects for reuse
- Simplifies reuse in complex environments
- Supports corporate, project, and personal standards
- Simplifies the sharing of reusable components
- Is separate from the form module



What Are Object Libraries?

Object libraries are convenient containers of objects for reuse. They simplify reuse in complex environments, and support corporate, project, and personal standards.

An object library can contain simple objects, property classes, object groups, and program units, but they are protected against change in the library. Objects can be used as standards (classes) for other objects.

Object libraries simplify the sharing of reusable components. Reusing components enables you to:

- Apply standards to simple objects, such as buttons and items, for consistent look and feel. This also improves network performance by promoting similarities among objects, thus increasing the efficiency of message diffing.
- Reuse complex objects such as a Navigator

In combination with SmartClasses, which are discussed later, object libraries support both of these requirements.

The screenshot in the slide of the Object Navigator shows that object libraries are not part of forms modules, but are separate objects.

What Are Object Libraries? (continued)

Why Object Libraries Instead of Object Groups?

- Object libraries are external to the form, so are easily shared among form modules.
- Object libraries can contain individual items (for example, buttons). The smallest unit accepted in an object group is a block.
- Object libraries accept PL/SQL program units.
- If you change an object in an object library, all forms that contain the subclassed object reflect the change.

Benefits of the Object Library

- Simplifies the sharing and reuse of objects
- Provides control and enforcement of standards
- Promotes increased network performance
- Eliminates the need to maintain multiple referenced forms



ORACLE®

23 - 18

Copyright © 2009, Oracle. All rights reserved.

Benefits of the Object Library

There are several advantages to using object libraries to develop applications:

- Simplifies the sharing and reuse of objects
- Provides control and enforcement of standards
- Increases the efficiency of message diffing by promoting similarity of objects, thus increasing the performance of the application
- Eliminates the need to maintain multiple referenced forms

The slide graphic represents a library.

Working with Object Libraries



Object libraries:

- Appear in the Navigator if they are open
- Are used with a simple tabbed interface
- Are populated by dragging Forms objects to tab page
- Are saved to .olb files

ORACLE®

Working with Object Libraries

Object libraries appear in the Navigator if they are open. You can create, open, and close object libraries like other modules. Forms Builder automatically opens all object libraries that were open when you last closed Forms Builder.

It is easy to use object libraries with a simple tabbed interface. Using the Edit menu, you can add or remove tab pages that help you to create your own groups. You can save object libraries to a file system as .olb files.

Note: You cannot modify objects inside the object library itself. To make changes, drag the object into a form, change it, and drag it back to the object library.

How to Populate an Object Library

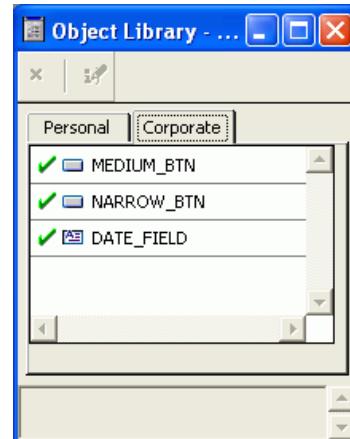
1. Select Tools > Object Library to display the object library, or right-click it in the Object Navigator and select Object Library.
2. Drag objects from the Object Navigator into the object library.
3. You can edit the descriptive comment by clicking the Edit button on the object library interface.

The screenshot shows an object library with two tabs: Personal and Corporate. The selected Personal tab contains three objects: TOOLBAR, QUESTION_ALERT, and CONTROL, which is a block.

What Are SmartClasses?

- A SmartClass:
 - Is an object in an object library that is frequently used as a class
 - Can be applied easily and rapidly to existing objects
 - Is the preferred method to promote similarity among objects for performance
- You can have many SmartClasses of a given object type.
- You can define SmartClasses in multiple object libraries.

To create:
Edit > SmartClass



The check mark indicates a SmartClass.

ORACLE®

What Are SmartClasses?

A SmartClass is a special member of an object library. It can be used to easily subclass existing objects in a form using the SmartClass option from the shortcut menu. To use object library members, which are not SmartClasses for existing objects, you have to use the Subclass Information dialog box available in the Property Palette of the form object that you are modifying.

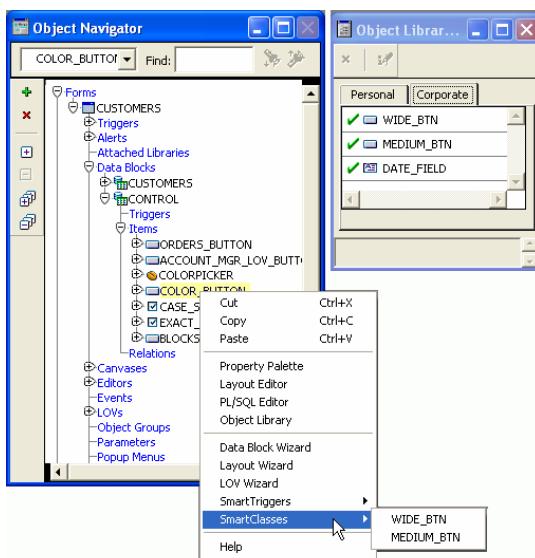
If you frequently use certain objects as standards, such as standard buttons, date items, and alerts, you can mark them as SmartClasses by selecting each object in the object library and selecting Edit > SmartClass.

You can mark many different objects, spread across multiple object libraries, as SmartClasses. Other than accepting default values for all object properties, using Smart Classes is the preferred method to promote similarities between objects for efficiency in message diffing, resulting in better performance of applications.

You can also have many SmartClasses of a given object type (for example, Wide_button, Narrow_button, and Small_iconic_button).

The screenshot shows the Corporate tab of the object library with three objects: MEDIUM_BTN, NARROW_BTN, and DATE_FIELD. All have green check marks indicating that they are SmartClasses.

Working with SmartClasses



1. Right-click an object in the Layout Editor or the Object Navigator.
2. From the pop-up menu, select SmartClasses.
3. Select a class from the list to apply it to the object.

Working with SmartClasses

To apply a SmartClass to a Forms object, perform the following steps:

1. Right-click an object in the Layout Editor or Object Navigator.
2. From the pop-up menu, select SmartClasses. The SmartClasses pop-up menu lists all the SmartClasses from all open object libraries that have the same type as the object, and, for items, also have the same item type (for example, push button, text item).
3. Select a class for the object; it then becomes the parent class of the object. You can see its details in the Subclass Information dialog box in the object's Property Palette, just as in any other subclassed object.

This mechanism makes it very easy to apply classes to existing objects. The screenshot shows right-clicking a button in the Object Navigator and selecting SmartClasses from the pop-up menu. The choices are MEDIUM_BTN and WIDE_BTN, which are the only SmartClasses applicable to buttons.

Reusing PL/SQL

- Triggers:
 - Copy and paste text.
 - Copy and paste within a module.
 - Copy to or subclass from another module.
 - Move to an object library.
- PL/SQL program units:
 - Copy and paste text.
 - Copy and paste within a module.
 - Copy to or subclass in another module.
 - Create a library module.
 - Move to an object library.

ORACLE®

23 - 22

Copyright © 2009, Oracle. All rights reserved.

Reusing PL/SQL

PL/SQL in Triggers

You can reuse PL/SQL in your triggers by:

- Copying and pasting using the Edit menu
- Copying to another area of the current form module using Copy and Paste on the shortcut menu
- Copying to or subclassing from another form module using the drag-and-drop feature in the Object Navigator
- Dragging the trigger to an object library

PL/SQL Program Units

Although triggers are the primary way to add programmatic control to a Forms Builder application, using PL/SQL program units to supplement triggers, you can reuse code without having to retype it.

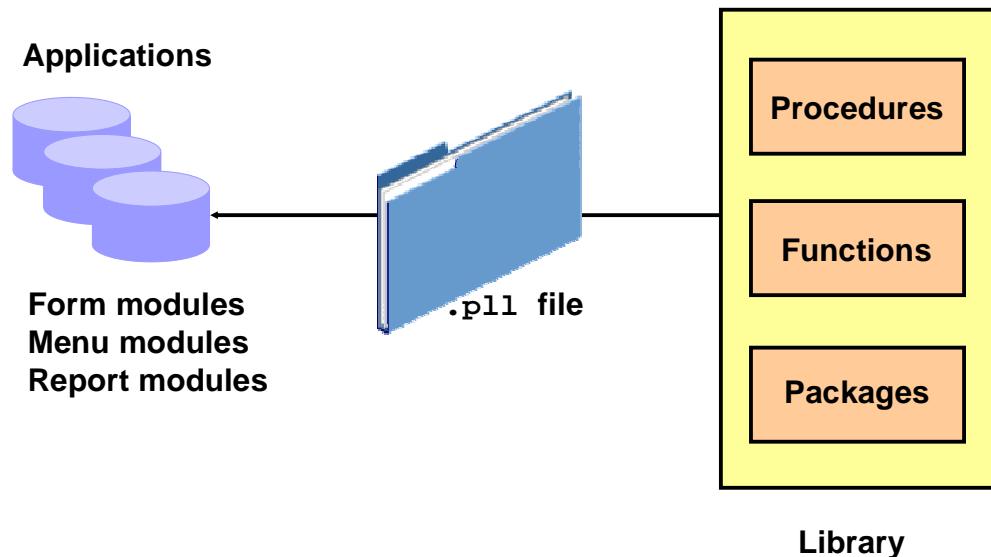
With Forms Builder, you can create PL/SQL program units to hold commonly used code. These PL/SQL program units can use parameters, which decrease the need to hard-code object names within the procedure body.

Reusing PL/SQL (continued)

You can reuse PL/SQL program units by:

- Copying and pasting using the Edit menu
- Copying or subclassing to another form module using the drag-and-drop feature in the Object Navigator
- Creating a library module
- Dragging the program unit to an object library

What Are PL/SQL Libraries?



ORACLE®

23 - 24

Copyright © 2009, Oracle. All rights reserved.

What Are PL/SQL Libraries?

A *library* is a collection of PL/SQL program units that can include procedures, functions, and packages. A single library can contain many program units that can be shared among the Oracle Forms Developer modules and applications that need to use them.

A library:

- Is produced as a separate module and stored in a .p11 file
- Provides a convenient means of storing client-side code and sharing it among applications
- Provides a way for many form, menu, or report modules to use a single copy of program units
- Supports dynamic loading of program units

The slide graphic depicts a library on the right that contains procedures, functions, and packages. Because it is a separate .p11 file, shown in the middle of the slide, it can be used by different forms, menus, and reports.

Writing Code for PL/SQL Libraries

- A library is a separate module, holding procedures, functions, and packages.
- Bind variables in forms, menus, and reports are outside the scope of the library, so direct references to bind variables are not allowed.
- Use subprogram parameters for passing bind variables.
- Use functions, where appropriate, to return values.

ORACLE®

23 - 25

Copyright © 2009, Oracle. All rights reserved.

Writing Code for PL/SQL Libraries

Because libraries are compiled independently of the Forms modules that use them, bind variables in forms, menus, and reports are outside the scope of the library. This means that you cannot directly refer to variables that are local to another module because the compiler does not know about them when you compile the library program units.

There are two ways to avoid direct references to bind variables:

- You can refer to global variables and system variables in forms indirectly as quoted strings by using certain built-in subprograms.
- Write program units with IN and IN OUT parameters that are designed to accept references to bind variables. You can then pass the names of bind variables as parameters when calling the library program units from your Forms applications.

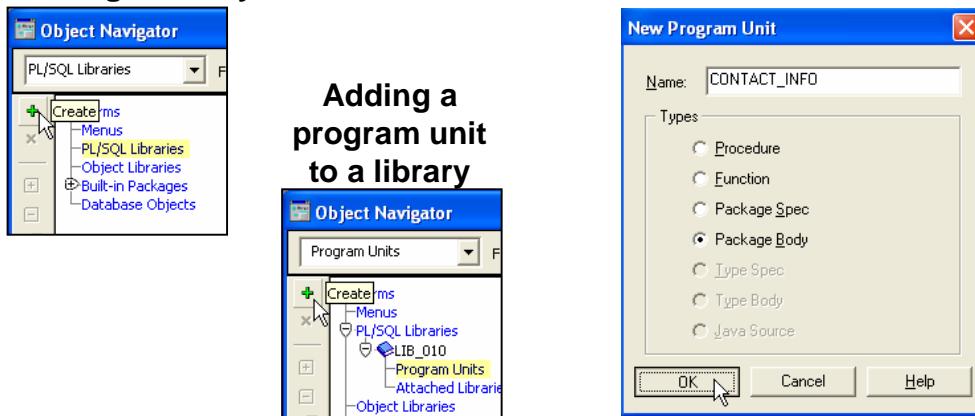
Example

Consider the second method listed above in the following library subprogram:

```
FUNCTION locate_emp(bind_value IN NUMBER) RETURN VARCHAR2 IS
    v_ename VARCHAR2(15);
BEGIN
    SELECT last_name INTO v_ename FROM employees
    WHERE employee_id = bind_value;
    RETURN(v_ename);
END;
```

Creating Library Program Units

Creating a library



Adding a program unit to a library

23 - 26

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Creating Library Program Units

Creating a Library

You must first create libraries in the builder before you add program units. To do this, you can do one of the following:

- Select File > New > PL/SQL Library from the menu; an entry for the new library then appears in the Object Navigator.
- Select the PL/SQL Libraries node in the Object Navigator, and then click Create on the tool bar (shown in the screenshot at the left of the slide).

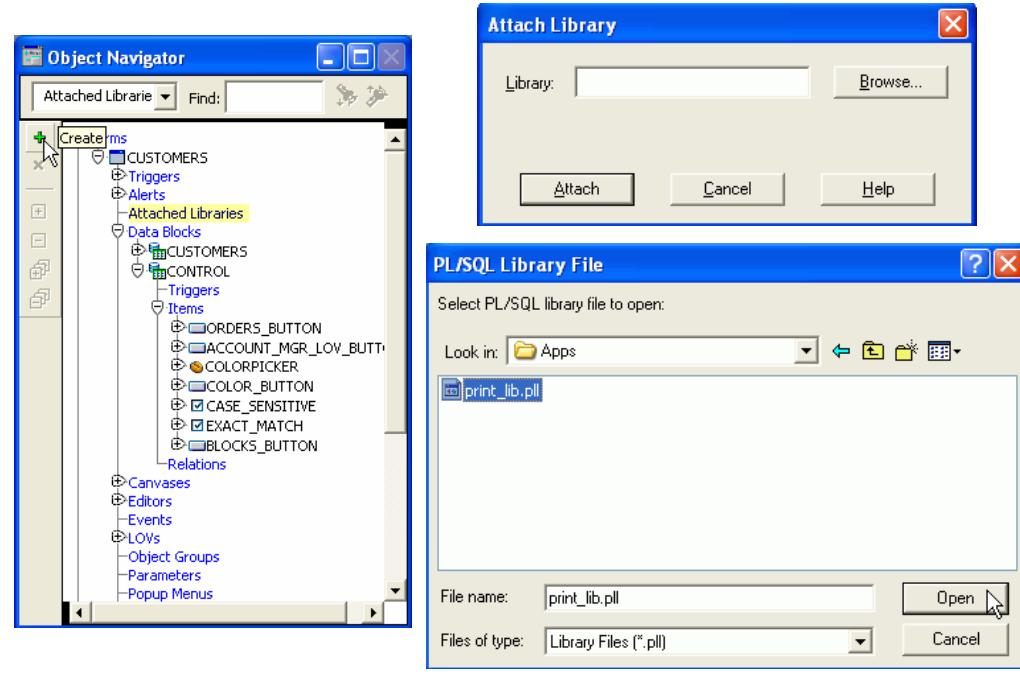
Adding Program Units

There is a Program Units node within the library's hierarchy in the Object Navigator. From this node, you can create procedures, functions, package bodies, and specifications in the same way as in other modules. Select the Program Units node and click Create. This opens a New Program Unit dialog box, where you select the type of object to create.

How to Save the Library

1. With the context set on the library, click Save.
2. Enter the name by which the library is to be saved.

Attaching Libraries



ORACLE®

23 - 27

Copyright © 2009, Oracle. All rights reserved.

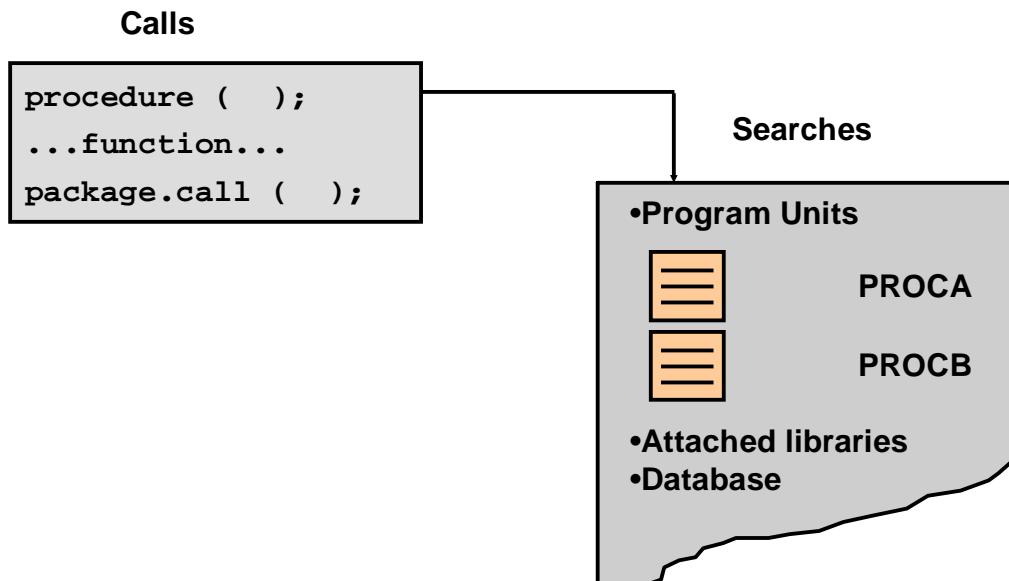
Attaching Libraries

Before you can refer to a library program unit from a form, menu, or report, you must attach the library to the modules. To attach a library to a module, perform the following steps:

1. Open the module that needs to have the library attached. This may be a form, menu, or another library module.
2. Expand the module and select the Attached Libraries node in the Object Navigator.
3. Click Create to display the Attach Library dialog box.
4. In the Attach Library dialog box, specify the name of the library. You can click Browse to open a file dialog box, where you can select a library file to open.
5. Click Attach. You are asked if you want to remove the path to the library. You should click Yes and include the path to the library in the FORMS_PATH design-time and run-time environment variables to avoid a hard-coded path, thus facilitating deployment.
6. Save the module to which you have attached the library. This permanently records the library attachment in the definition of this module.

To detach a library later, in the Object Navigator, delete the library entry from the list of Attached Libraries for that module. That module will then no longer be able to reference the library program units, either in the Builder or at run time.

Referencing Attached Library Program Units



Referencing Attached Library Program Units

You refer to library program units in the same way as those that are defined locally, or stored in the database. Remember that objects declared in a package must be referenced with the package name as a prefix, whether or not they are part of a library.

Program units are searched for first in the calling module, and then in the libraries that are attached to the calling module, and finally in stored database code.

Example

Assume that the program units `report_totals`, `how_many_people`, and `pack5.del_emps` are defined in an attached library. You call them as follows:

```
report_totals(:sub1);      --library procedure
v_sum := how_many_people; --library function
pack5.del_emps;           --library package procedure
```

Referencing Attached Library Program Units (continued)

When Several Libraries Are Attached

You can attach several libraries to the same Oracle Forms Developer module. References are resolved by searching through libraries in the order in which they occur in the attachment list.

If two program units of the same name and type occur in different libraries in the attachment list, the one in the “higher” library will be executed because it is located first.

Creating .plx Files

The library .plx file is a platform-specific executable that contains no source.

When you are ready to deploy your application, you should probably generate a version of your library that contains only the compiled p-code, without any source. You can generate a .plx file from Forms Builder. These compiled files are optional; if no .plx file exists, the .pll file is used.

Summary

In this lesson, you should have learned that:

- You can reuse objects or code in the following ways:
 - Property classes
 - Object groups
 - Copying and subclassing
 - Object libraries and SmartClasses
- To inherit properties from a property class, you set an item's Subclass Information property
- You can create an object group in one module to make it easy to reuse related objects in other modules

ORACLE®

23 - 30

Copyright © 2009, Oracle. All rights reserved.

Summary

Forms provides a variety of methods for reusing objects and code. This lesson described how to use these methods.

Reasons to share objects and code:

- Increased productivity
- Increased modularity
- Decreased maintenance
- Maintaining standards
- Increased performance

Methods of sharing objects and code:

- Property classes
- Object groups
- Copying
- Subclassing
- Creating a library module
- Using object libraries and SmartClasses

Summary

- Inheritance symbols in the Property Palette show whether the value is changed, inherited, overridden, or the default
- You can drag objects from an object library or mark them as SmartClasses for even easier reuse
- You can reuse PL/SQL code by:
 - Copying and pasting in the PL/SQL Editor
 - Copying or subclassing
 - Defining program units to call the same code at multiple places within a module
 - Creating PL/SQL library to call the same code from multiple forms

ORACLE®

Practice 23: Overview

This practice covers the following topics:

- Creating an object group and using this object group in a new form module
- Using property classes
- Creating an object library and using this object library in a new form module
- Modifying an object in the object library and observing the effect on subclassed objects
- Setting and using SmartClasses
- Creating a PL/SQL program unit to be called from multiple triggers

ORACLE®

23 - 32

Copyright © 2009, Oracle. All rights reserved.

Practice 23: Overview

In this practice, you use an object group and an object library to copy Forms Builder objects from one form to another. You will also create a property class and use it to set multiple properties for several objects. You set SmartClasses in the object library and use these classes in the form module.

Using WebUtil to Interact with the Client

24

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe the benefits of the WebUtil utility
- Integrate WebUtil into a form
- Use WebUtil to interact with a client machine



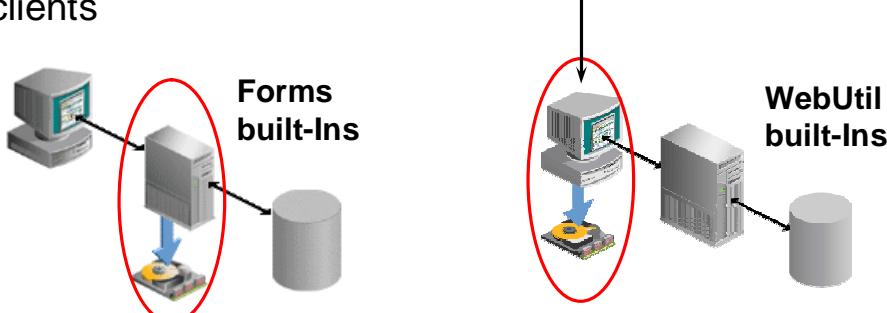
Lesson Aim

Forms built-in subprograms are called on the middle-tier application server. However, there are times when you want to be able to interact with the client machine. You can do so with JavaBeans and Pluggable Java Components (PJC's), but there is a utility called WebUtil that includes much prewritten functionality for client interaction. This lesson shows you how to use WebUtil to interface with the client machine.

WebUtil: Overview

WebUtil is a utility that:

- Enables you to provide client-side functionality on Win32 clients



- Consists of:
 - Java classes
 - Forms objects
 - PL/SQL library

ORACLE®

24 - 3

Copyright © 2009, Oracle. All rights reserved.

WebUtil: Overview

Forms built-ins typically are executed on the application server machine. Some Forms built-ins interact with the machine to create or read a file, read an image file, or execute operating system commands. Although, in some cases, it is desirable to execute such built-ins on the application server machine, there is often a need to perform such functionality on the client. To do this, you can use a JavaBean or PJC, but that requires that you either write or locate prewritten components and integrate each into Forms applications.

WebUtil is a utility that enables you to easily provide client-side functionality. It consists of a set of Java classes, Forms objects, and a PL/SQL API that enables you to execute the many Java functions of WebUtil without knowing Java.

For further information about WebUtil, see the WebUtil page on OTN at:

<http://otn.oracle.com/products/forms/htdocs/webutil/webutil.htm>

The graphics in the slide illustrate the fact that, while Forms built-ins interact with the middle tier, WebUtil code executes functionality on the client.

Note: The client machine must be a Windows 32-bit platform, although the middle-tier server on which WebUtil is installed can be any platform on which Forms Services is supported.

Benefits of the WebUtil Utility

Why use WebUtil?

- The developer has to code only in PL/SQL (no Java knowledge required).
- WebUtil is a part of Forms 11g.
- It is easy to integrate into a Forms application.
- It is extensible.



Benefits of the WebUtil Utility

Any Forms developer can use WebUtil to carry out complex tasks on client browser machines by simply coding PL/SQL.

It is very easy to integrate WebUtil into your Forms applications using its object group and PL/SQL library, and you can easily extend it by adding your own custom functionality while leveraging its basic structure.

What Functionality Is Available?

WebUtil provides:

- Client/server parity APIs
- Client/server added value functions
- Public functions
- Utility functions
- Internal functions



What Functionality Is Available?

You can use WebUtil to perform a multitude of tasks, enabling you to:

- Read and write text files on the client machine
- Transfer files between the client, application server, and database
- Read client-side variables
- Manipulate client-side files
- Integrate with C code on the client
- Obtain information about the client
- Use a file selection dialog box on the client
- Run operating system commands on the client machine and call back into Forms
- Integrate with the browser, such as displaying messages to the browser window
- Perform OLE (Object Linking and Embedding) automation, such as using Word and Excel, on the client
- Read and write client-side images

What Functionality Is Available? (continued)

There is a wealth of functionality available in the utility, including the following:

- Client/server parity APIs that enable you to retrieve a file name from the client, read or write an image to or from the client, get information about the client machine, or perform HOST and TEXT_IO commands and OLE automation on the client; without the WebUtil functions, these built-ins execute on the application server machine.
- Client/server added value functions (ported from d2kwutil, a client/server package):

```
CREATE_REGISTRY_KEY and DELETE_REGISTRY_KEY  
GET_COMPUTER_NAME  
GET_NET_CONNECTION  
GET_TEMP_DIRECTORY, GET_WINDOWS_DIRECTORY, and  
GET_WORKING_DIRECTORY  
GET_WINDOWS_USERNAME  
READ_INI_FILE and WRITE_INI_FILE  
READ_REGISTRY, WRITE_REGISTRY, and WRITE_REGISTRYEX
```

Note: Some of these may duplicate other WebUtil functions, but are provided as an easy way to migrate code that uses d2kwutil.

- **Public functions:** The core of WebUtil is a set of packages, each of which provides an API to implement certain functionality.
 - WebUtil_ClientInfo package: Contains the following functions to obtain information about the client machine:

Function	Returns:
GET_DATE_TIME	Date and time on the client machine
GET_FILE_SEPARATOR	Character used on the client as file separator (“\” on Windows)
GET_HOST_NAME	Name of the client machine
GET_IP_ADDRESS	IP address of the client (string)
GET_JAVA_VERSION	Java Virtual Machine (JVM) version that is running the Forms applet
GET_LANGUAGE	Language code of the client machine, such as de for German
GET_OPERATING_SYSTEM	Name of the operating system (OS) running the browser
GET_PATH_SEPARATOR	Character used on the client to separate directory locations on paths (“;” on Windows)
GET_SYSTEM_PROPERTY	Any Java system property
GET_TIME_ZONE	Time zone of the client machine
GET_USER_NAME	Name of the user logged on to the client

What Functionality Is Available? (continued)

- WebUtil_C_API package: Contains the following functions to call into C libraries on the client:

Function	Purpose:
IsSupported	Returns True if the client is a valid platform
RegisterFunction	Returns a handle to a specified C library function
DeregisterFunction	Deregisters function and frees client resources
Create_Parameter_List	Creates a parameter list to pass to C function
Destroy_Parameter_List	Deletes a parameter list and frees its resources
Add_Parameter	Adds a parameter to the parameter list
Get_Parameter_Number Get_Parameter_Ptr Get_Parameter_String	Typed functions to return parameter values
Rebind_Parameter	Changes parameter values of the existing parameter list to reuse it
Invoke_*	Typed functions to execute a registered C function
Parameter_List_Count	Returns the number of parameter lists that have been created
Function_Count	Returns the number of functions that have been registered
Id_Null	Checks to see whether the various types used in the package are null

What Functionality Is Available? (continued)

- WebUtil_File package: Contains a new type called FILE_LIST, which is a PL/SQL table used to pass back multiple file names; also contains the following APIs to manipulate files and directories on the client and to display file selection dialog boxes:

Function	Purpose
COPY_FILE RENAME_FILE DELETE_FILE	Copy, rename, or delete a file and return a Boolean value to indicate success
CREATE_DIRECTORY	Creates the named directory if it does not exist; returns a Boolean value to indicate success
DIRECTORY_ROOT_LIST	Returns a FILE_LIST containing the directory roots on the client system (the drives on Windows)
DIRECTORY_FILTERED_LIST	Returns a list of files in a directory that you filter using wildcard characters (*) and (?)
FILE_EXISTS	Returns a Boolean value indicating the existence of the specified file on the client
FILE_IS_DIRECTORY	Returns a Boolean value indicating whether the specified file is a directory
FILE_IS_HIDDEN	Returns a Boolean value indicating whether the specified file has its hidden attribute set
FILE_IS_READABLE FILE_IS_WRITABLE	Returns a Boolean value indicating whether the file can be read from or written to
DIRECTORY_SELECTION_DIALOG	Displays a directory selection dialog box and returns the selected directory
FILE_SELECTION_DIALOG	Enables definition of the File Save or File Open dialog box with configurable file filter and returns the selected file
FILE_MULTI_SELECTION_DIALOG	Enables definition of the File Save or File Open dialog box with configurable file filter and returns the selected files in a FILE_LIST
GET_FILE_SEPARATOR	Returns the character used on the client machine as a file separator ("\ on Windows)
GET_PATH_SEPARATOR	Returns character used on client machine to separate directory locations on paths (";" on Windows)

What Functionality Is Available? (continued)

- `WebUtil_File_Transfer` package: The `WebUtil_File_Transfer` package contains APIs to transfer files to and from the client browser machine and to display a progress bar as the transfer occurs. The following APIs are included in the `WebUtil_File_Transfer` package:

Function	Purpose:
<code>URL_TO_CLIENT</code> <code>URL_TO_CLIENT_WITH_PROGRESS</code>	Transfers a file from a URL to the client machine (and displays a progress bar)
<code>CLIENT_TO_DB</code> <code>CLIENT_TO_DB_WITH_PROGRESS</code>	Uploads a file from the client to a database BLOB column (and displays a progress bar)
<code>DB_TO_CLIENT</code> <code>DB_TO_CLIENT_WITH_PROGRESS</code>	Downloads file from a BLOB column in the database to the client machine (and displays a progress bar)
<code>CLIENT_TO_AS</code> <code>CLIENT_TO_AS_WITH_PROGRESS</code>	Uploads a file from the client to the application server (with a progress bar)
<code>AS_TO_CLIENT</code> <code>AS_TO_CLIENT_WITH_PROGRESS</code>	Transfers a file from the application server to the client (with a progress bar)
<code>IN_PROGRESS</code>	Returns True if an asynchronous update is in progress
<code>ASYNCHRONOUS_UPLOAD_SUCCESS</code>	Returns a Boolean value indicating whether an asynchronous upload succeeded
<code>GET_WORK_AREA</code>	Returns a work area directory on the application server that is private to the user
<code>IS_AS_READABLE</code> <code>IS_AS_WRITABLE</code>	Returns True if the rules defined in the WebUtil configuration allow the specified file to be read from or written to

- `WebUtil_Session` package: Provides a way to react to an interruption of the Forms session by defining a URL to which the user is redirected if the session ends or crashes. It contains the following:

Function	Purpose
<code>ENABLE_REDIRECT_ON_TIMEOUT</code>	Enables the time-out monitor and the specification of a redirection URL
<code>DISABLE_REDIRECT_ON_TIMEOUT</code>	Switches off the monitor; if you do not call this function before <code>EXIT_FORM</code> , the redirection occurs even if the exit is normal

What Functionality Is Available? (continued)

- WebUtil_Host package: Contains routines to execute commands on the client machine. Includes two types: PROCESS_ID to hold a reference to a client-side process, and OUTPUT_ARRAY, a PL/SQL table which holds the VARCHAR2 output from a client-side command. The WebUtil_Host package also contains the following functions:

Function	Purpose
HOST	Runs the specified command in BLOCKING mode on the client and optionally returns the return code
BLOCKING	Runs the specified command in BLOCKING mode on the client and optionally returns the process ID
NONBLOCKING	Runs the specified command in NON-BLOCKING mode on the client and optionally returns the process ID
NONBLOCKING_WITH_CALLBACK	Runs the specified command in NON-BLOCKING mode on the client and executes a supplied trigger when the process is complete
TERMINATE_PROCESS	Kills the specified process on the client
GET_RETURN_CODE	Returns the return code of a specified process as an integer
GET_STANDARD_OUTPUT	Returns OUTPUT_ARRAY containing output that was sent to standard output by the specified client process
GET_STANDARD_ERROR	Returns OUTPUT_ARRAY containing output that was sent to standard error by the specified client process
RELEASE_PROCESS	Frees resources of the specified client process
GET_CALLBACK_PROCESS	Returns the process ID of the finished process when a callback trigger executes following a command called from NonBlocking_With_Callback
ID_NULL	Tests if a process ID is null
EQUALS	Tests whether two process IDs represent the same process

What Functionality Is Available? (continued)

- WebUtil_Core package: Contains mostly private functions, but you can call the following functions:

Function	Purpose
IsError	Checks whether the last WebUtil call succeeded
ErrorCode	Returns the last WebUtil error code
ErrorText	Returns the text of the last WebUtil error

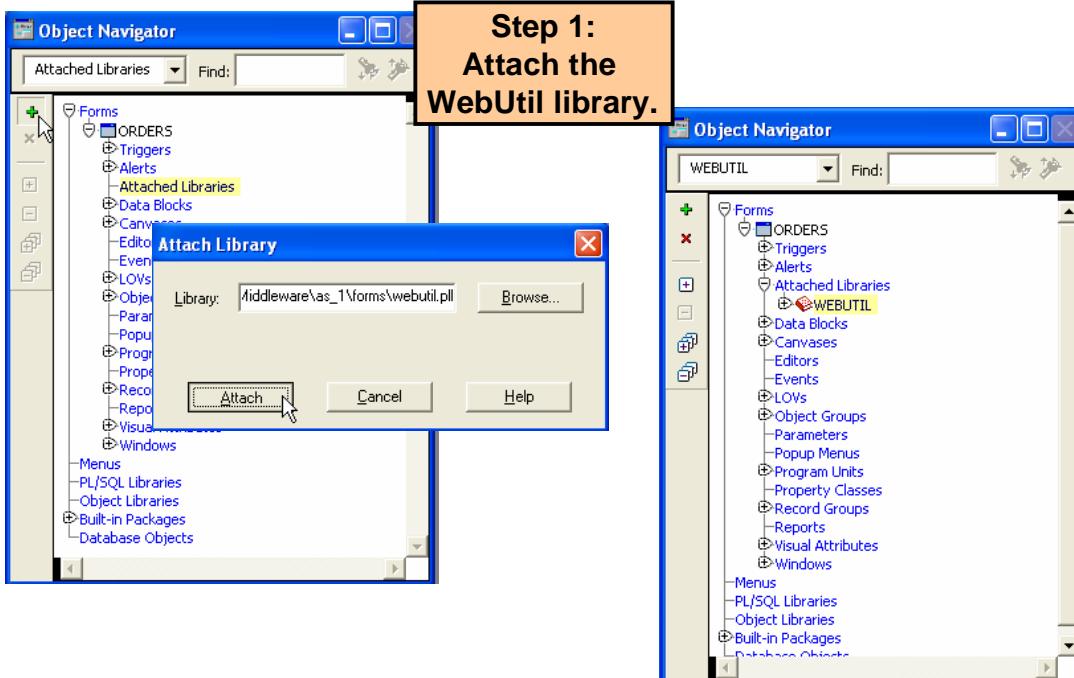
- **Utility functions:** The following functions are not related to client integration, but they can be useful:

Function	Purpose
DelimStr	Provides interaction with delimited strings
Show_Web_Util_Information	Calls the hidden WebUtil window to show the version of all WebUtil components
WebUtil_Util	Provides the BoolToStr() function for converting Boolean to text

- Internal APIs that should not be called directly

For more information about WebUtil, including a sample form that showcases its functionality, see OTN at <http://otn.oracle.com/products/forms/htdocs/webutil/webutil.htm>.

Integrating WebUtil into a Form



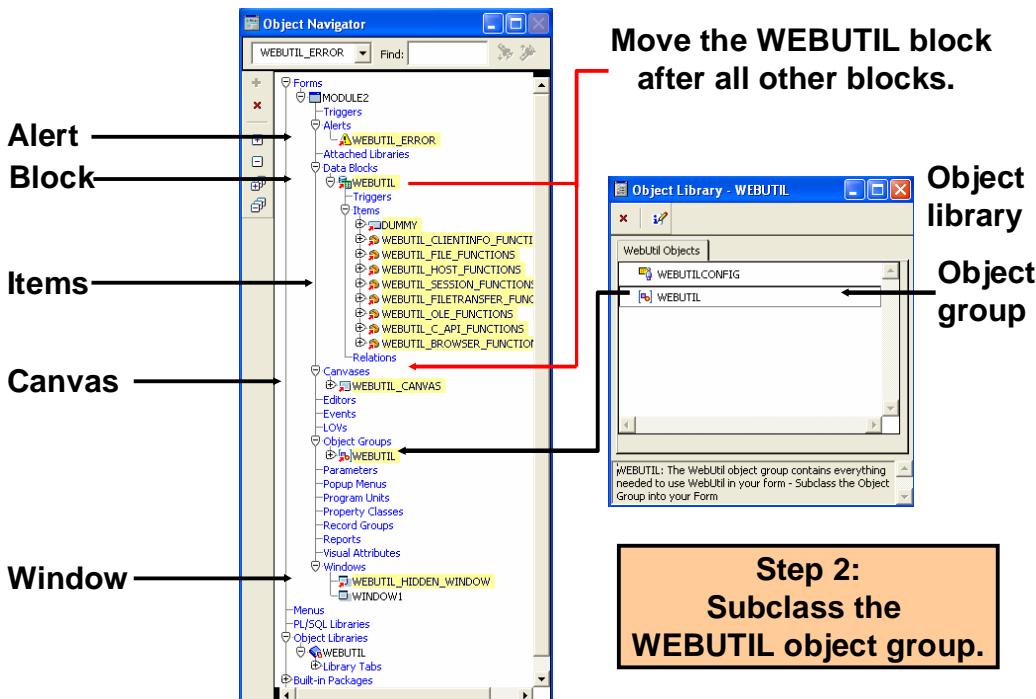
Integrating WebUtil into a Form

Step 1: Attaching the WebUtil Library

To use the functions of WebUtil in a Forms application, you must first attach the `webutil.p11` library to any module that will use the WebUtil PL/SQL API. Select the Attached Libraries node in the Orders form and click Create. This invokes the Attach Library dialog box, in which you can browse to the location of `webutil.p11`. The screenshot shows loading it from `<ORACLE_HOME>\forms`. The slide also shows the Object Navigator after attaching, with a WEBUTIL node under Attached Libraries.

Note: To avoid Bug 6861382, you need to first load `webutil.p11` into Forms Builder and resave it, or compile it into a `.plx` file.

Integrating WebUtil into a Form



Integrating WebUtil into a Form (continued)

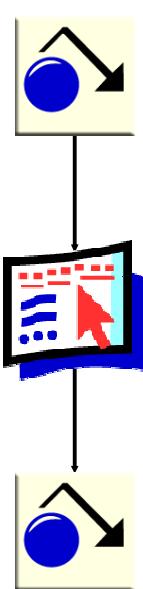
Step 2: Subclassing WebUtil Forms Objects

Part of the WebUtil utility is a set of Forms objects contained in `webutil.olb`. This object library contains an object group called WebUtil, which you can subclass into your form.

The screenshot at the left of the slide shows all the objects that are created by subclassing the WEBUTIL object group:

- A generic alert named `Webutil_Error` to display WebUtil error messages
- A data block named `WEBUTIL`; ensure that this is the last block in the Navigator
- Items:
 - A button named `Dummy`
 - Several bean area items to implement the JavaBeans (the bean area items are hidden because there is no visual component) : `Webutil_Clientinfo_Functions`, `Webutil_File_Functions`, `Webutil_Host_Functions`, `Webutil_Session_Functions`, `Webutil_Filetransfer_Functions`, `Webutil_Ole_Functions`, `Webutil_C_Api_Functions`, and `Webutil_Browser_Functions`
- A canvas named `WEBUTIL_CANVAS` to display the items
- A window named `WEBUTIL_HIDDEN_WINDOW` to display the canvas

When to Use the WebUtil Functionality



Pre-Form
When-New-Form-Instance
When-New-Block-Instance
(first block)

Form starts
JavaBeans are instantiated

**Any trigger after form starts
and while form is running**



ORACLE®

24 - 14

Copyright © 2009, Oracle. All rights reserved.

When to Use the WebUtil Functionality

After the WebUtil library has been attached to your form, you can start to add calls to the various PL/SQL APIs defined by the utility. However, there is an important restriction in the use of WebUtil functions—WebUtil can communicate with the client only after the form has instantiated the WebUtil JavaBeans.

This means that you cannot call WebUtil functions before the user interface is rendered, so you should not use the WebUtil functionality in triggers such as Pre-Form, When-New-Form-Instance, and When-New-Block-Instance for the first block in the form. In the case of the When-New-Form-Instance trigger, it is possible, however, to call WebUtil functions after a call to the SYNCHRONIZE built-in has been issued because this ensures that the user interface is rendered.

Further, you cannot call WebUtil functions after the user interface has been destroyed. For example, you should not use a WebUtil call in a Post-Form trigger.

The graphics in the slide illustrate that triggers that fire before the form starts and JavaBeans are initiated should not call WebUtil code. However, once the JavaBeans are instantiated at form startup, you can call WebUtil functionality from any trigger.

Interacting with the Client

Forms Built-Ins/Packages	WebUtil Equivalents
HOST	CLIENT_HOST
GET_FILE_NAME	CLIENT_GET_FILE_NAME
READ_IMAGE_FILE	CLIENT_IMAGE.READ
WRITE_IMAGE_FILE	(WRITE)_IMAGE_FILE
OLE2	CLIENT_OLE2
TEXT_IO	CLIENT_TEXT_IO
TOOL_ENV	CLIENT_TOOL_ENV

ORACLE®

Interacting with the Client

As previously mentioned, Forms built-ins work on the application server. For the most common Forms built-ins that you would want to use on the client, rather than on the application server, you can add a prefix to use the WebUtil equivalent.

These client/server parity APIs make it easy to provide similar functionality in applications that were written for client/server deployment by prefixing those built-ins with “CLIENT_” or “CLIENT_IMAGE”. Although this makes it easy to upgrade such applications, other WebUtil commands may provide similar, but better, functionality. The client/server parity APIs include the following:

- CLIENT_HOST
- CLIENT_GET_FILE_NAME

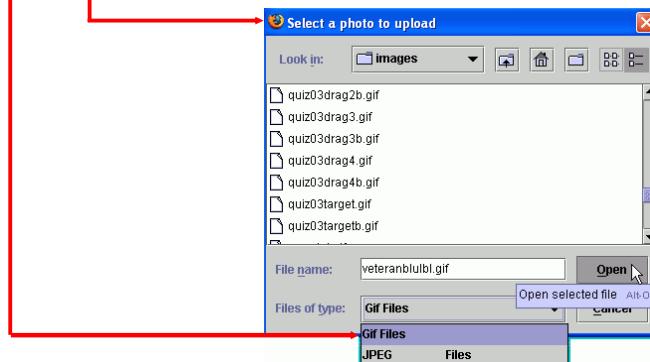
You can use READ_IMAGE_FILE on the client by calling the WebUtil equivalent contained in a package: CLIENT_IMAGE.READ_IMAGE_FILE.

In addition, there are certain Forms packages that you can use on the client with WebUtil:

- CLIENT_OLE2
- CLIENT_TEXT_IO
- CLIENT_TOOL_ENV

Example: Opening a File Dialog Box on the Client

```
DECLARE
    v_file VARCHAR2(250):= CLIENT_GET_FILE_NAME('','','',
'Gif Files|*.gif|JPEG Files|*.jpg|',
'Select a photo to upload',open_file,TRUE);
```



ORACLE®

24 - 16

Copyright © 2009, Oracle. All rights reserved.

Example: Opening a File Dialog Box on the Client

To open a file dialog box on the client for selecting a file, you can use:

```
CLIENT_GET_FILE_NAME ( DIRECTORY_NAME IN VARCHAR2,
FILE_NAME IN VARCHAR2, FILE_FILTER IN VARCHAR2,
MESSAGE IN VARCHAR2, DIALOG_TYPE IN NUMBER,
SELECT_FILE IN BOOLEAN) RETURN VARCHAR2;
```

The arguments for this WebUtil function are:

- **DIRECTORY_NAME:** Specifies the name of the directory containing the file you want to open. If DIRECTORY_NAME is NULL, subsequent invocations of the dialog box may open the last directory visited.
- **FILE_NAME:** Specifies the name of the file you want to open
- **FILE_FILTER:** Specifies that only particular files be shown. On Windows, the default is “All Files (*.*) | *.* |” if NULL.
- **MESSAGE:** Specifies the title of the file upload dialog box
- **DIALOG_TYPE:** Specifies the intended dialog box to OPEN_FILE or SAVE_FILE
- **SELECT_FILE:** Specifies whether the user is selecting files or directories. The default value is TRUE; if set to FALSE, the user must select a directory. If DIALOG_TYPE is set to SAVE_FILE, SELECT_FILE is internally set to TRUE.

Example: Reading an Image File into Forms from the Client

The screenshot shows a Windows file selection dialog box titled "Select a photo to upload". The "Look in:" dropdown is set to "images". A list of files is shown, with "veteranblu.jpg" selected. Below the list are fields for "File name:" (set to "veteranblu.jpg") and "Files of type:" (set to "JPEG Files (*.jpg)"). A red arrow points from the "Open" button in the dialog box to a button labeled "Upload Picture" in the Oracle Forms application window. The application window has a title bar "PHOTO_WIN" and contains a preview image of a person's face.

```
DECLARE
    v_file VARCHAR2(250)
    'Gif Files|*.gif|JPEG Files|*.jpg|',
    'Select a photo to upload',open_file,TRUE);
    it_image_id ITEM := FIND_ITEM
    ('employee_photos.photo');
BEGIN
    CLIENT_IMAGE.READ_IMAGE_FILE(v_file,'',it_image_id);
END;
```

24 - 17

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Example: Reading an Image File into Forms from the Client

You can use the CLIENT_IMAGE package to read or write image files. For example, the CLIENT_IMAGE.READ_IMAGE_FILE procedure reads an image from the client file system and displays it in the Forms image item:

```
CLIENT_IMAGE.READ_IMAGE_FILE (FILE_NAME VARCHAR2,
FILE_TYPE VARCHAR2, ITEM_ID ITEM or ITEM_NAME VARCHAR2);
```

The arguments for this WebUtil procedure are:

- **FILE_NAME:** The valid file name, which can include a full path statement appropriate to your operating system.
- **FILE_TYPE:** The valid image file type, which can be BMP, CALS, GIF, JFIF, JPG, PICT, RAS, TIFF, or TPIC. (**Note:** File type is optional because Oracle Forms will attempt to deduce it from the source image file. To optimize performance, however, you should specify the file type.)
- **ITEM_ID:** The unique ID that Oracle Forms assigns to the image item when it creates it
- **ITEM_NAME:** The name you gave the image item when you created it

The screenshots in the slide show the dialog box that is opened by the example code in the previous slide and the image item into which the selected image is loaded by the code in this slide.

Example: Writing Text Files on the Client

```
DECLARE
    v_dir VARCHAR2(250) := 'd:\temp';
    ft_tempfile CLIENT_TEXT_IO.FILE_TYPE;
begin
    ft_tempfile := CLIENT_TEXT_IO.FOPEN(v_dir || 
        '\tempdir.bat','w');
    CLIENT_TEXT_IO.PUT_LINE(ft_tempfile,'dir ' || 
        v_dir || '> '|| v_dir || '\mydir.txt');
    CLIENT_TEXT_IO.PUT_LINE(ft_tempfile,
        'notepad ' || v_dir || '\mydir.txt');
    CLIENT_TEXT_IO.PUT_LINE(ft_tempfile,'del ' || 
        v_dir || '\mydir.*');
    CLIENT_TEXT_IO.FCLOSE(ft_tempfile);
    CLIENT_HOST('cmd /c ' || v_dir || '\tempdir');
END;
```

- 1
- 2
- 3
- 4

ORACLE

24 - 18

Copyright © 2009, Oracle. All rights reserved.

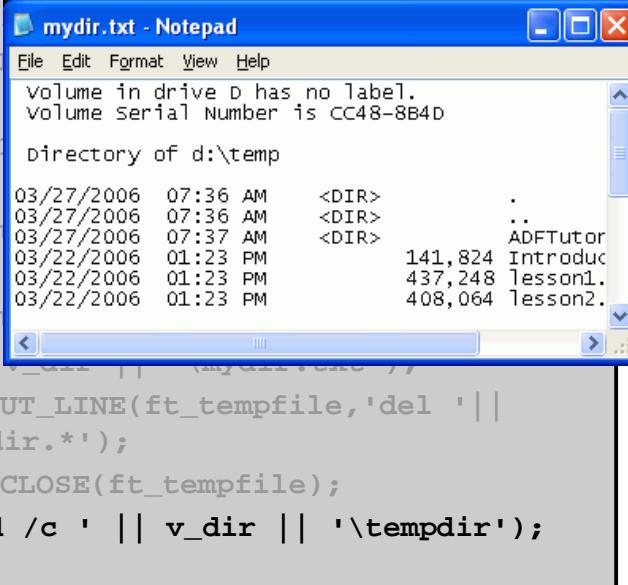
Example: Writing Text Files on the Client

With CLIENT_TEXT_IO commands, you can create or read text files on the client containing any ASCII text. This example creates a batch file on the client to display a directory listing of the c :\temp directory. The code does the following:

1. Declares a variable to hold a handle to the file
2. Opens a file named tempdir.bat on the client for writing
3. Writes the following lines of text to the file:
 dir d:\temp> d:\temp\mydir.txt
 notepad d:\temp\mydir.txt
 del d:\temp\mydir.*
4. Closes the file

Example: Executing Operating System Commands on the Client

```
DECLARE
    v_dir VARCHAR2(255);
    ft_tempfile CLIENT_TEXT_IO.FILE_TYPE;
begin
    ft_tempfile := CLIENT_TEXT_IO.POPEN('notepad > mydir.txt');
    v_dir || '>' || ft_tempfile;
    CLIENT_TEXT_IO.PUT_LINE(ft_tempfile,'del '|| v_dir || '\mydir.*');
    CLIENT_TEXT_IO.FCLOSE(ft_tempfile);
    CLIENT_HOST('cmd /c ' || v_dir || '\tempdir');
END;
```



ORACLE®

24 - 19

Copyright © 2009, Oracle. All rights reserved.

Example: Executing Operating System Commands on the Client

You can execute simple HOST commands on the client using the CLIENT_HOST command of WebUtil. The example shows running the batch file that was created in the previous example with CLIENT_TEXT_IO; cmd /c opens a command window and closes it after running the command. You must use cmd /c rather than running the command directly or it will not work. You can run any command that you would be able to execute from the Windows Start > Run menu. The screenshot shows the file that was created by running the batch file, displaying a directory listing of d:\temp on the client.

Rather than creating the batch file with CLIENT_TEXT_IO, you can execute the commands it contains as follows:

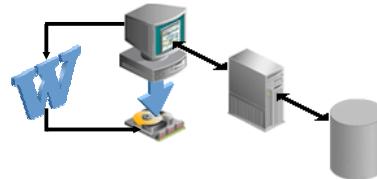
```
CLIENT_HOST('cmd /c dir c:\temp > c:\temp\mydir.txt');
CLIENT_HOST('cmd /c notepad c:\temp\mydir.txt');
CLIENT_HOST('cmd /c del c:\temp\mydir.*');
```

Note: You can obtain greater versatility in executing OS commands by using the WebUtil_Host package. This enables you to execute commands synchronously or asynchronously and to call back into Forms from asynchronous commands when execution is complete.

Example: Performing OLE Automation on the Client

You can use the following for OLE automation:

CLIENT_OLE2.OBJ_TYPE	CLIENT_OLE2.CREATE
CLIENT_OLE2.LIST_TYPE	_ARGLIST
CLIENT_OLE2.CREATE_OBJ	CLIENT_OLE2.ADD_ARG
CLIENT_OLE2.SET	CLIENT_OLE2.INVOKE
_PROPERTY	CLIENT_OLE2.DESTROY
CLIENT_OLE2.GET_OBJ	_ARGLIST
_PROPERTY	CLIENT_OLE2.RELEASE_OBJ
CLIENT_OLE2.INVOKE_OBJ	



ORACLE®

24 - 20

Copyright © 2009, Oracle. All rights reserved.

Example: Performing OLE Automation on the Client

You can use any OLE2 package on the client by prefixing it with CLIENT_. You can see the list of the OLE2 package procedures and functions in the Forms Builder Object Navigator under the Built-in Packages node.

You can see examples of client OLE on OTN. The slide graphic represents using Word on the client machine. The following example takes data from a form to construct a Word document and save it to the client machine:

```
DECLARE
    app CLIENT_OLE2.OBJ_TYPE;
    docs CLIENT_OLE2.OBJ_TYPE;
    doc CLIENT_OLE2.OBJ_TYPE;
    selection CLIENT_OLE2.OBJ_TYPE;
    args CLIENT_OLE2.LIST_TYPE;
BEGIN
    -- create a new document
    app := CLIENT_OLE2.CREATE_OBJ('Word.Application');
    CLIENT_OLE2.SET_PROPERTY(app,'Visible',1);
```

Example: Performing OLE Automation on the Client (continued)

```
docs := CLIENT_OLE2.GET_OBJ_PROPERTY(app, 'Documents');
doc := CLIENT_OLE2.INVOKE_OBJ(docs, 'add');

selection := CLIENT_OLE2.GET_OBJ_PROPERTY(app,
'Selection');
-- Skip 10 lines
args := CLIENT_OLE2.CREATE_ARGLIST;
CLIENT_OLE2.ADD_ARG(args,6);
FOR i IN 1..10 LOOP
    CLIENT_OLE2.INVOKE(selection, 'InsertBreak', args);
END LOOP;

-- insert data into new document

CLIENT_OLE2.SET_PROPERTY(selection, 'Text',
'RE: Order# ' || :orders.order_id);
FOR i in 1..2 LOOP
    CLIENT_OLE2.INVOKE(selection, 'EndKey');
    CLIENT_OLE2.INVOKE(selection, 'InsertBreak', args);
END LOOP;

CLIENT_OLE2.SET_PROPERTY(selection, 'Text',
'Dear ' || :customer_name || ':');
FOR i in 1..2 LOOP
    CLIENT_OLE2.INVOKE(selection, 'EndKey');
    CLIENT_OLE2.INVOKE(selection, 'InsertBreak', args);
END LOOP;

CLIENT_OLE2.SET_PROPERTY(selection, 'Text', 'Thank you for
your ' || 'order dated' ||
to_char(:orders.order_date, 'fmMonth DD, YYYY') ||
', in the amount of ' ||
to_char(:control.total, '$99,999.99') ||
'. We will process your order immediately and want you to '
|| 'know that we appreciate your business');

FOR i in 1..2 LOOP
    CLIENT_OLE2.INVOKE(selection, 'EndKey');
    CLIENT_OLE2.INVOKE(selection, 'InsertBreak', args);
END LOOP;
```

Example: Performing OLE Automation on the Client (continued)

```
CLIENT_OLE2.SET_PROPERTY(selection,'Text','Sincerely,' );
FOR i in 1..5 LOOP
    CLIENT_OLE2.INVOKE(selection,'EndKey');
    CLIENT_OLE2.INVOKE(selection,'InsertBreak',args);
END LOOP;

IF :orders.sales_rep_id IS NOT NULL THEN
    CLIENT_OLE2.SET_PROPERTY(selection,'Text',
        :orders.sales_rep_name || ' , Sales Representative');
    CLIENT_OLE2.INVOKE(selection,'EndKey');
    CLIENT_OLE2.INVOKE(selection,'InsertBreak',args);
END IF;

CLIENT_OLE2.SET_PROPERTY(selection,'Text','Summit Office
Supply');

-- save document in temporary directory
CLIENT_OLE2.DESTROY_ARGLIST(args);
args := CLIENT_OLE2.CREATE_ARGLIST;
CLIENT_OLE2.ADD_ARG(args, 'letter_' ||
    :orders.order_id || '.doc');
CLIENT_OLE2.INVOKE(doc, 'SaveAs', args);
CLIENT_OLE2.DESTROY_ARGLIST(args);

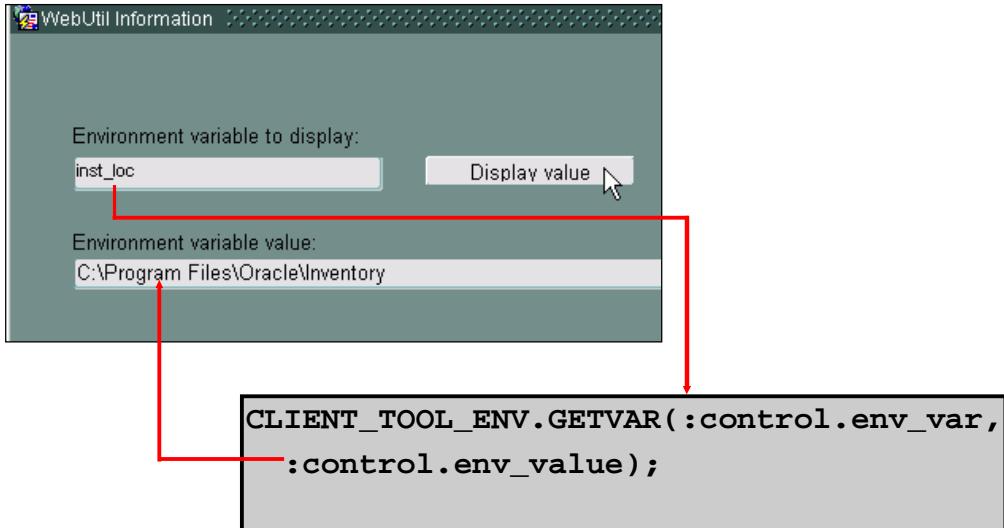
-- close example.tmp
args := CLIENT_OLE2.CREATE_ARGLIST;
CLIENT_OLE2.ADD_ARG(args, 0);
CLIENT_OLE2.INVOKE(doc, 'Close', args);
CLIENT_OLE2.DESTROY_ARGLIST(args);

CLIENT_OLE2.RELEASE_OBJ(selection);
CLIENT_OLE2.RELEASE_OBJ(doc);
CLIENT_OLE2.RELEASE_OBJ(docs);

-- exit MSWord
CLIENT_OLE2.INVOKE(app,'Quit');
message('Letter created: letter_' ||
    :orders.order_id || '.doc');
END;
```

Note: To use client OLE, you must download jacob.jar, and for the developer environment you must include jacob.jar in FORMS_BUILDER_CLASSPATH.

Example: Obtaining Environment Information About the Client



Example: Obtaining Environment Information About the Client

You can use the `CLIENT_TOOL_ENV.GETVAR` procedure from WebUtil to obtain information about registry variables on the client machine. You can obtain the values of any registry variables in the key `HKEY_LOCAL_MACHINE > SOFTWARE > ORACLE`.

The example in the slide shows a form with a text item where the user can input the environment variable whose value they want to see, which is `inst_loc` in this example. You click a button to display the value in another text item, and its When-Button-Pressed trigger contains the code shown in the slide. The value that is displayed is “`C:\Program Files\Oracle\Inventory`”.

Note: You can obtain more information about the client with the `WebUtil_ClientInfo` package.

Summary

In this lesson, you should have learned that:

- WebUtil is a free extensible utility that enables you to interact with the client machine
- Although WebUtil uses Java classes, you code in PL/SQL
- You integrate WebUtil into a form by attaching its PL/SQL library and using an object group from its object library; then you can use its functions after the form has started and while it is running
- With WebUtil, you can do the following on the client machine: open a file dialog box, read and write image or text files, execute operating system commands, perform OLE automation, and obtain information about the client machine

ORACLE®

24 - 24

Copyright © 2009, Oracle. All rights reserved.

Summary

WebUtil, included as part of Developer Suite 10g Patchset 1 and a free download from OTN before that, consists of a set of Java classes and a PL/SQL API. You can extend WebUtil by adding Java classes. The PL/SQL API enables you to do all coding within the form in PL/SQL.

After the middle tier has been configured for WebUtil, in order to use it in a form, you need only add an object group from WebUtil's object library and attach WebUtil's PL/SQL library. You should not use WebUtil functions in triggers that fire as the form is starting up or after its user interface has been destroyed.

WebUtil includes various functionalities. Some of the most common commands enable you to:

- Open a file dialog box on the client (CLIENT_GET_FILE_NAME)
- Read or write an image file on the client (CLIENT_IMAGE package)
- Read or write a text file on the client (CLIENT_TEXT_IO)
- Execute OS commands (CLIENT_HOST or the WebUtil.HOST package)
- Perform OLE automation on the client (CLIENT_OLE2)
- Obtain information about the client machine (CLIENT_TOOL_ENV)

Practice 24: Overview

This practice covers the following topics:

- Integrating WebUtil with a form
- Using WebUtil functions to:
 - Open a file dialog box on the client
 - Read an image file from the client into the form
 - Obtain the value of a client environment variable
 - Create a file on the client
 - Open the file on the client with Notepad



Practice 24: Overview

In this practice, you integrate WebUtil with a form, and then use WebUtil to perform various functions on the client machine.

Introducing Multiple Form Applications

25

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Call one form from another form module
- Define multiple form functionality
- Share data among open forms



Lesson Aim

Oracle Forms applications rarely consist of a single form document. This lesson introduces you to the ways in which you can link two or more forms.

Multiple Form Applications: Overview

- Behavior:
 - Flexible navigation between windows
 - Single or multiple database connections
 - Transactions may span forms, if required
 - Commits in order of opening forms, starting with the current form
- Links:
 - Data is exchanged by global variables, parameter lists, global record groups, or PL/SQL variables in shared libraries.
 - Code is shared as required, through libraries and the database.

ORACLE®

25 - 3

Copyright © 2009, Oracle. All rights reserved.

Multiple Form Applications: Overview

As you know, you can design Forms applications where blocks are distributed over more than one form, producing a modular structure. A modular structure indicates the following:

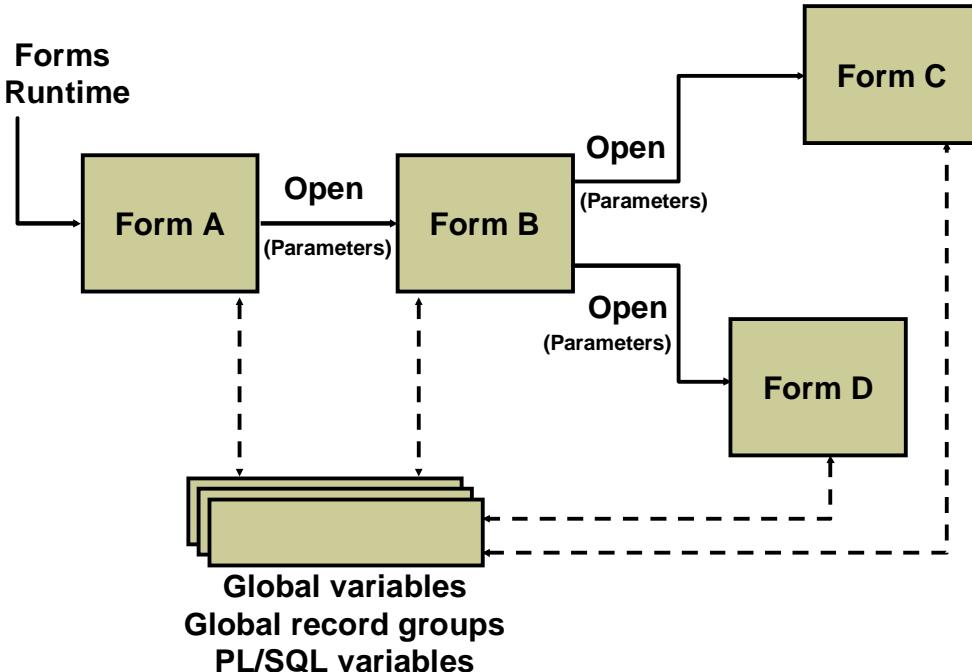
- Component forms are only loaded in memory if they are needed.
- One form can be called from another, providing flexible combinations, as required.

How Does the Application Behave?

The first form module to run is specified before the Forms session begins in the HTML file that starts the application. Other form modules can be opened in the session by calling built-ins from triggers.

You can design forms to appear in separate windows, so the user can work with several forms concurrently in a session (when forms are invoked by the OPEN_FORM built-in). Users can then navigate between visible blocks of different forms, much as they can in a single form.

Multiple Form Applications: Overview



Multiple Form Applications: Overview (continued)

You can design forms for a Forms Runtime session according to the following conditions:

- Forms share the same database session, or open their own separate sessions.
- Database transactions are continued across forms, or ended before control is passed to another form. The commit sequence starts from the current form and follows the opening order of forms.
- Forms Builder provides the same menus across the application, or each form provides its own separate menus when it becomes the active form.

What Links the Forms Together?

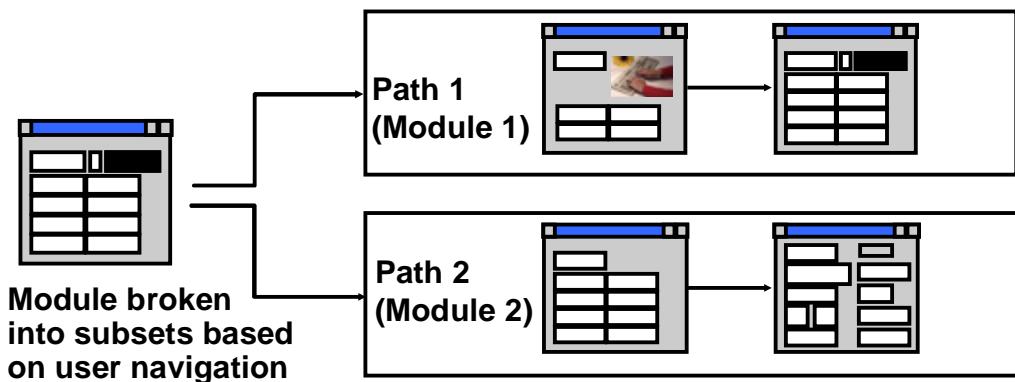
Each form runs within the same Forms Runtime session, and Forms remembers the form that invoked each additional form. This chain of control is used when you exit a form or commit transactions.

There are several methods to exchange data between forms, and code can also be shared. The slide graphics depict several forms: Form A opens Form B, which opens Form C and Form D. Parameters are passed from each form to the one it opens. The following objects are available to provide information to all open forms: Global variables, global record groups, and PL/SQL variables. These mechanisms for sharing data are discussed later in this lesson.

Benefits of Multiple Form Applications

Breaking your application into multiple forms offers the following advantages:

- Easier debugging
- Modularity
- Performance and scalability



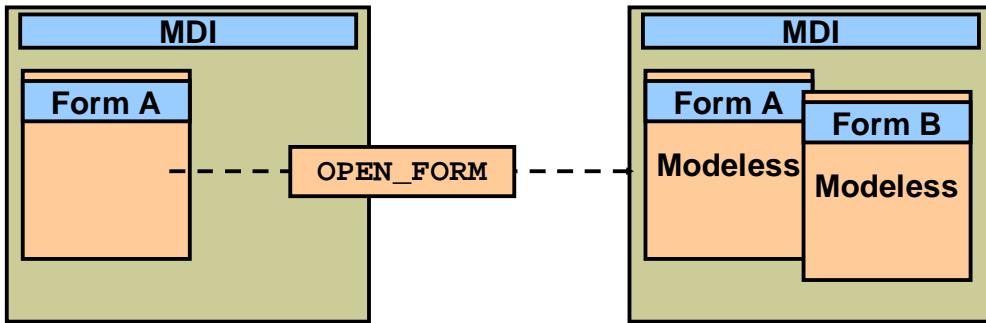
Benefits of Multiple Form Applications

Because you can use different windows and canvases, it may seem easier to put the entire application into one large form. However, separating your application into multiple forms offers the following benefits:

- **Debugging:** It is easier to debug a small form; when a single form is working perfectly, you have only to integrate it into your application.
- **Logic modularity:** You break the application into pieces based on the logical functions the module is designed to perform. For example, you would not want the functions of human resources management and accounts receivable combined in a single form, and within each of these major functions are other logical divisions.
- **Network performance and scalability:** Sufficient information must be downloaded to describe the entire form before the form appears on the user's screen. Large forms take longer to download the relevant information, and fewer users can be supported on the given hardware. Break large applications into smaller components based on the likelihood of user navigation, enabling logical areas to be loaded on demand rather than all at once. This approach enables the module to start faster and uses less memory on the application server.

The graphics in the slide depict breaking a form module into multiple forms based on how the user navigates through the application.

Starting Another Form Module



ORACLE®

Starting Another Form Module

When the first form in a Forms Runtime session has started, it can provide the user with facilities for starting additional forms. This can be performed by one of two methods:

- Calling a built-in procedure from a trigger in the form
- Calling a built-in procedure from a menu item in an attached menu

Built-in Procedures for Starting Another Form

You can use the `OPEN_FORM` built-in to start another form module from the one that is already active. This is a restricted procedure and cannot be called in Enter Query mode. `OPEN_FORM` enables you to start another form in a modeless window so the user can work in other running forms at the same time.

The graphics show the MDI window with Form A displayed in it. After Form A issues an `OPEN_FORM` command to open Form B, both forms are open in the MDI window, and both are modeless.

You can start another form by using `OPEN_FORM` without passing control to it immediately, if required. This built-in also gives you the option to begin a separate database session for the new form.

Starting Another Form Module (continued)

Syntax:

```
OPEN_FORM('form_name', activate_mode, session_mode,  
          data_mode, paramlist);
```

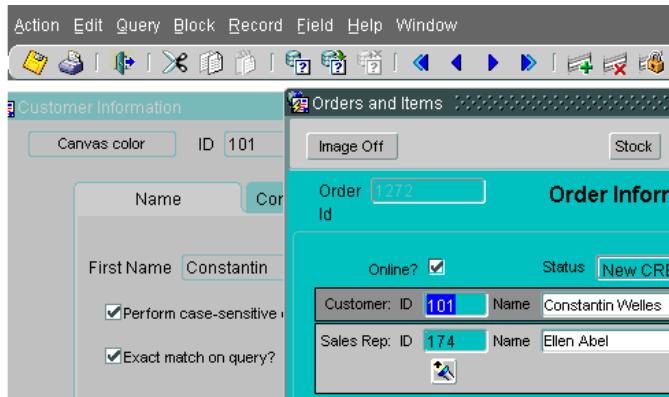
Parameter	Description
Form_Name	File name of the executable module (without the .FMX suffix)
Activate_Mode	Either ACTIVATE (the default) or NO_ACTIVATE
Session_Mode	Either NO_SESSION (the default) or SESSION
Data_Mode	Either NO_SHARE_LIBRARY_DATA (the default) or SHARE_LIBRARY_DATA (Use this parameter to enable Forms to share data among forms that have identical libraries attached.)
Paramlist	Either the name (within quotation marks) or internal ID of a parameter list

There are two other built-ins that can call another form. This course concentrates on OPEN_FORM, which is considered the primary method to invoke multiple forms, rather than the CALL_FORM or NEW_FORM built-ins. For a discussion about these additional built-ins, see the *Oracle9i Forms Developer: Enhance Usability* online course. You can access the online library from the Oracle Education Web page at: <http://www.oracle.com/education>.

Using OPEN_FORM to Provide Forms in Multiple Windows

Summit application scenario:

- Run the Customers and Orders forms in the same session, navigating freely between them.
- You can make changes in the same transaction across forms.
- All forms are visible together.



ORACLE®

25 - 8

Copyright © 2009, Oracle. All rights reserved.

Using OPEN_FORM to Provide Forms in Multiple Windows

You can use OPEN_FORM to link form modules in an application and enable the user to work in them concurrently. Consider these requirements for the Summit application:

- The Customers form must provide an option to start the Orders form in the same transaction, and orders for the current customer can be viewed, inserted, updated, and deleted.
- The user can see all open forms at the same time and freely navigate between them to apply changes.
- Changes in all forms can be saved together.

Using OPEN_FORM to open both forms in the same session satisfies the requirements.

However, having both forms open in the same session may have an undesired effect: If changes are made in the opened form, but not in the calling form, when saving the changes, users may receive an error message indicating that no changes have been made. This error is produced by the calling form; changes in the opened form are saved successfully, but the error message may be confusing to users. To avoid this, you may decide to open the second form in a separate session, but then changes to each form would need to be saved separately.

The screenshot shows Customer 101 in the Customers form, and shows the Orders form immediately after OPEN_FORM is issued. Orders for Customer 101 are queried into the Orders form. The method for doing this is discussed in this lesson.

Control and Transactions When Opening Another Form

When you use OPEN_FORM:

- By default, control passes immediately to the Orders form and no statements after OPEN_FORM are processed
- If the Activate_Mode argument is set to NO_ACTIVATE, you retain control in the current form
- The transaction continues unless it was explicitly committed before

ORACLE®

25 - 9

Copyright © 2009, Oracle. All rights reserved.

Control and Transactions When Opening Another Form

You should be aware of how form control and transactions operate when using OPEN_FORM:

- When you default the Activate_Mode argument in OPEN_FORM, control is passed immediately to the specified form, and any remaining statements after OPEN_FORM are not executed.
- If you set Activate_Mode to NO_ACTIVATE, control remains in the calling form, although the specified form starts up and the rest of the trigger is processed. Users can then navigate to the other form when they choose.
- If you want to end the current transaction before opening the next form, call the COMMIT_FORM built-in before OPEN_FORM. You can check to see if the value of :SYSTEM.form_status='CHANGED' to decide whether a commit is needed. Alternatively, you can just post changes to the database with POST, then open the next form in the same transaction.

Defining Multiple Form Functionality

Actions:

1. Define windows and positions for each form.
2. Plan shared data, such as global variables and their names.
3. Implement triggers to:
 - Open other forms
 - Initialize shared data from calling forms
 - Use shared data in opened forms

ORACLE®

25 - 10

Copyright © 2009, Oracle. All rights reserved.

Defining Multiple Form Functionality

To provide this kind of functionality, perform the following steps:

1. Create each of the form modules. Plan where the windows of each module will appear in relation to those of other modules.
2. Plan how to share data among forms, such as identifying names for global variables. You need one for each item of data that is to be accessible across all the forms in the application. Note that each form must reference a global variable by the same name.
Note: You can also share data among forms using parameter lists, global record groups, or PL/SQL variables in shared libraries.
3. Plan and implement triggers to:
 - Open another form (You can do this from item interaction triggers, such as When-Button-Pressed, or from When-New-*object*-Instance triggers, or from a Key trigger that fires on a keystroke or equivalent menu selection.)
 - Initialize shared data in calling forms so that values such as unique keys are accessible to other forms when they open. This might need to be done in more than one trigger, if the reference value changes in the calling form.
 - Make use of shared data in opened forms. For example, a Pre-Query trigger can use the contents of the global variable as query criteria.

Conditional Opening

Example:

```
IF  ID_NULL(FIND_FORM('orders')) THEN
    OPEN_FORM('orders');
ELSE
    GO_FORM('orders');
END IF;
```

ORACLE®

25 - 11

Copyright © 2009, Oracle. All rights reserved.

Conditional Opening

The OPEN_FORM built-in enables you to start up several instances of the same form. To prevent this, the application can perform appropriate tests, such as testing a flag (global variable) set by an opened form at startup, which the opened form could reset on exit. This method may be unreliable, however, because if the form exits with an error, the flag may not be properly reset. A better practice is to use the FIND_FORM built-in.

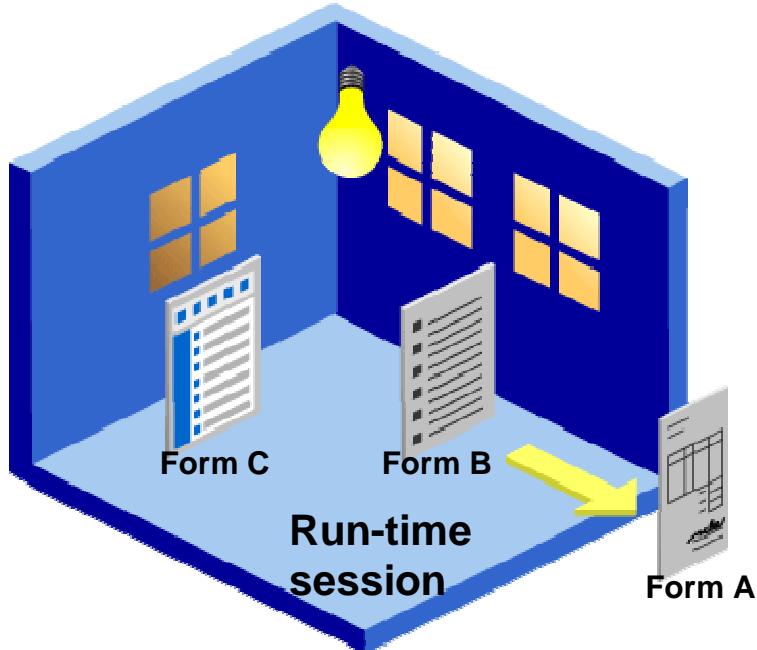
Here is a variation of the When-Button-Pressed trigger on Orders_Button in the Customers form. If the Orders form is already running, it simply uses GO_FORM to pass control to it.

```
IF  ID_NULL(FIND_FORM('orders')) THEN
    OPEN_FORM('orders');
ELSE
    GO_FORM('orders');
END IF;
```

Note: If the name of the form module and its file name are different:

- Use the file name for OPEN_FORM: OPEN_FORM('orderswk23');
- Use the form module name for GO_FORM: GO_FORM('orders');

Closing the Session



“Will the last one out please turn off the light?”

ORACLE®

25 - 12

Copyright © 2009, Oracle. All rights reserved.

Closing the Session

A form may close down and pass control back to its calling form under the following conditions:

- The user clicks Exit or selects Exit from the Action menu.
- The EXIT_FORM built-in is executed from a trigger.

If the closing form is the only form still running in the Forms run-time session, the session will end as a result. When a multiple form session involves the OPEN_FORM built-in, it is possible that the last form to close is not the one that began the session.

The slide graphic depicts the run-time session as a room with a light on in it. Form A is exiting the room, while Form B and Form C are still in the room. The analogy is that the last one to leave should turn out the light, in other words, exit the run-time session.

Closing a Form with EXIT_FORM

- The default functionality is the same as for the Exit key.
- The commit_mode argument defines the action on uncommitted changes.

```
ENTER;

IF  :SYSTEM.form_status = 'CHANGED' THEN
    EXIT_FORM( DO_COMMIT );
ELSE
    EXIT_FORM( NO_COMMIT );
END IF;
```

ORACLE®

25 - 13

Copyright © 2009, Oracle. All rights reserved.

Closing a Form with EXIT_FORM

When a form is closed, Forms checks to see whether there are any uncommitted changes. If there are, the user is prompted with the standard alert:

Do you want to save the changes you have made?

If you are closing a form with EXIT_FORM, the default functionality is the same as described above. You can, however, make the decision to commit (save) or roll back through the EXIT_FORM built-in, so the user is not asked. Typically, you might use this built-in from a Key-Exit or When-Button-Pressed trigger.

```
EXIT_FORM(commit_mode);
```

Parameter	Description
Commit_Mode	Defines what to do with uncommitted changes to the current form: <ul style="list-style-type: none">• ASK_COMMIT (the default) gives the decision to the user.• DO_COMMIT posts and commits changes across all forms for the current transaction.• NO_COMMIT validates and rolls back uncommitted changes in the current form.• NO_VALIDATE is the same as NO_COMMIT, but without validation.

Other Useful Triggers

Triggers to maintain referential links and synchronize data between forms:

- In the parent form:
 - When-Validate-Item
 - When-New-Record-Instance
- In opened forms: When-Create-Record
- In any form: When-Form-Navigate



Other Useful Triggers

One drawback of designing applications with multiple forms is that you do not have the functionality that is available within a single form to automatically synchronize data such as master and detail records. You must provide your own coding to ensure that related forms remain synchronized.

Because OPEN_FORM enables the user to navigate among open forms, potentially changing and inserting records, you can use the triggers shown in the slide to help keep referential key values synchronized across forms.

Example

In the parent form (Customers), the following assignment to GLOBAL.customerid can be performed in a When-Validate-Item trigger on CUSTOMERS.Customer_Id, so that the global variable is kept up-to-date with user changes.

```
:GLOBAL.customerid := :customers.customer_id;
```

The statement can also be issued from a When-New-Record-Instance trigger on the CUSTOMERS block, in case the user navigates to a different customer record.

Other Useful Triggers (continued)

Example

In the opened form (Orders), the following code in a When-Create-Record trigger on the ORDERS block ensures that new records use the value of GLOBAL.customerid as their default.

```
:orders.customer_id := :GLOBAL.customerid;
```

When items are assigned from this trigger, the record status remains NEW, so that the user can leave the record without completing it.

Example

You may have a query-only form that displays employee IDs and names, with a button to open another form that has all the columns from the EMPLOYEE table so that users can insert new records.

You can use a When-Form-Navigate trigger in the query-only form to reexecute the query, so that when the user navigates back to that form, the newly created records are included in the display.

Sharing Data Among Modules

You can pass data between modules using:

- Global variables
- Parameter lists
- Global record groups
- PL/SQL package variables in shared libraries

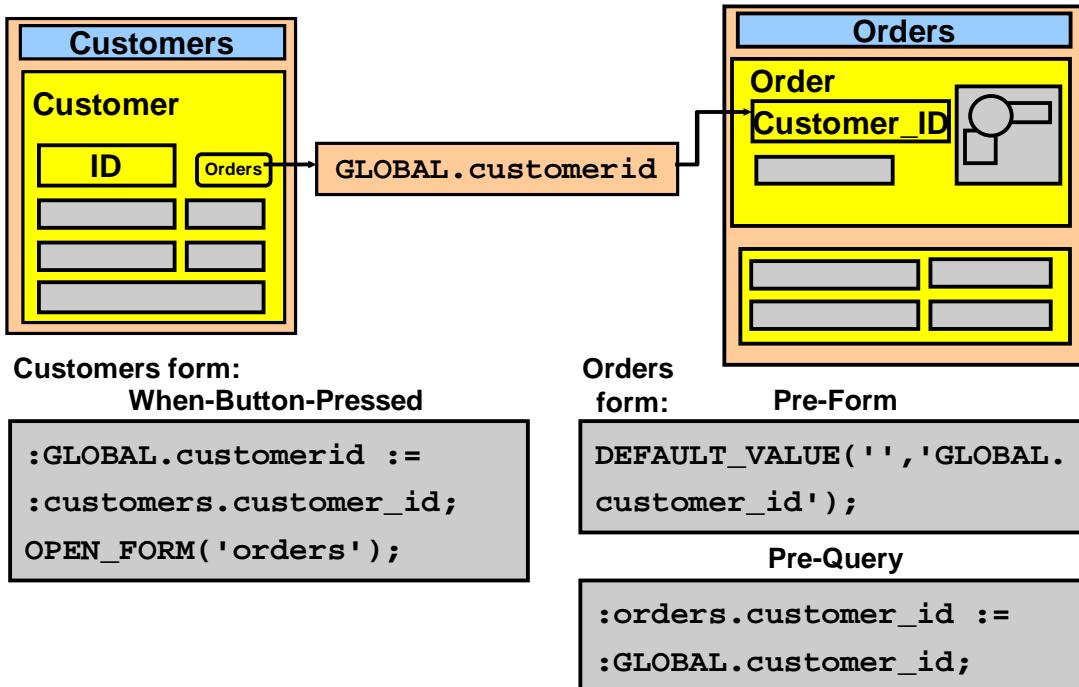


Sharing Data Among Modules

Data can be exchanged between forms as follows:

- Through global variables, which span sessions
- Through parameter lists, for passing values between specific forms
- Through record groups created in one form with global scope
- Through PL/SQL variables in shared libraries

Linking by Global Variables



ORACLE®

Linking by Global Variables

Planning Global Variables and Their Names

You need a global variable for each item of data that is used across the application.

Reminders:

- Global variables contain character data values, with a maximum of 255 characters.
- Each global variable is known by the same name to each form in the session.
- Global variables can be created by a PL/SQL assignment, or by the DEFAULT_VALUE built-in, which has no effect if the variable already exists.
- Attempting to read from a nonexistent global variable causes an error.

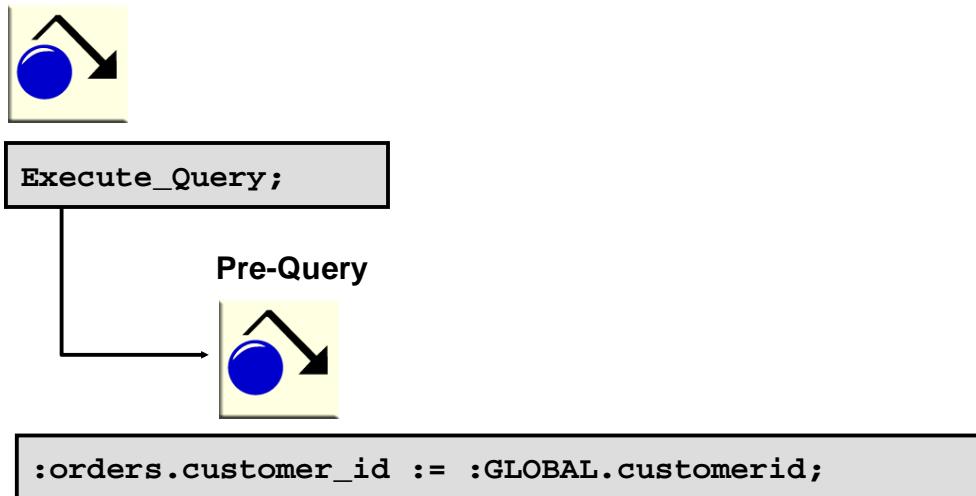
Opening the Orders Form from the Customers Form

The scenario in the slide shows one global variable: GLOBAL.customerid. In the Customers form, the When-Button-Pressed trigger on CONTROL.Orders_Button sets a global variable to the value of Customer_Id, opens the Orders form, and passes control immediately to it. The Orders form uses the same database session and transaction.

The code in the Pre-Query trigger for the Orders form ensures that orders queried at the startup of the Orders form apply to the current customer. The Pre-Form trigger is necessary in case the Orders form is used on its own, without a global variable being passed from another form. Subsequent slides elaborate on these triggers.

Global Variables: Restricted Query at Startup

When-New-Form-Instance



ORACLE®

25 - 18

Copyright © 2009, Oracle. All rights reserved.

Performing a Restricted Query on Startup

To display a query automatically in the opened form, with data in context to the calling form, you produce two triggers:

- **When-New-Form-Instance:** This form-level trigger fires when the form is opened (regardless of whether control is passed to this form immediately or not). You can use this trigger to initiate a query by using the EXECUTE_QUERY built-in procedure. Executing a query fires a Pre-Query trigger if one is defined. The Orders form contains the following When-New-Form-Instance trigger:
EXECUTE_QUERY;
- **Pre-Query:** This is usually on the master block of the opened form. Because this trigger fires in Enter Query mode, it can populate items with values from global variables, which are then used as query criteria. This restriction applies for every other query performed on the block thereafter.

This Pre-Query trigger is on the ORDERS block of the Orders form:

```
:orders.customer_id := :GLOBAL.customerid;
```

Initializing Global Variables in the Opened Form

- DEFAULT_VALUE ensures the existence of globals.
- You can use globals to communicate that the form is running.



Pre-Form example:

```
DEFAULT_VALUE(' ', 'GLOBAL.customerid');
```

ORACLE®

25 - 19

Copyright © 2009, Oracle. All rights reserved.

Initializing Global Variables in the Opened Form

If, for some reason, a global variable has not been initialized before it is referenced in a called form, an error is reported:

FRM-40815: Variable GLOBAL.customerid does not exist.

You can provide independence, and ensure that global variables exist by using the DEFAULT_VALUE built-in when the form is opening.

Example

This Pre-Form trigger in the Orders form assigns a NULL value to GLOBAL.CUSTOMERID if it does not exist when the form starts. Because the Pre-Form trigger fires before record creation and before all of the When-New-object-Instance triggers, it ensures the existence of global variables at the earliest point.

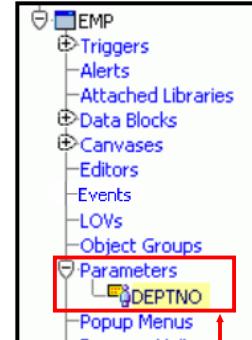
```
DEFAULT_VALUE(' ', 'GLOBAL.customerid');
```

The Orders form can now potentially be called without the Customers form.

Linking by Parameters

Parameters:

- Are form module objects
- Properties:
 - Name
 - Parameter Data Type
 - Maximum Length
 - Parameter Initial Value
- Can optionally receive a new value at run time:



```
http://myhost:8889/forms/frmservlet  
?form=emp.fmx&otherparams=deptno=140
```

ORACLE®

25 - 20

Copyright © 2009, Oracle. All rights reserved.

Linking by Parameters

You can create any number of parameters in a form to hold data. Unlike global variables, parameters can be of any data type. However, their use in multiform applications is limited by the fact that they are visible only to the form in which they are defined.

When you run a form, you can pass a value to the parameter as a name/value pair, such as &otherparams=deptno=140. After the form receives the parameter value, a trigger can use that value to perform such functionality as restricting a query to records containing that value. You can use the parameter by preceding its name with :parameter (for example, :parameter.deptno).

The screenshot in the slide shows an Emp form with a parameter named deptno defined. You can construct a Pre-Query trigger to assign the value of :parameter.deptno to the :employees.department_id form item. When a query is executed on the EMPLOYEES block, only the employees from department 140 are retrieved.

Linking by Parameter Lists: The Calling Form

Example:

```
DECLARE
    pl_id      ParamList;
    pl_name    VARCHAR2(10) := 'tempdata';
BEGIN
    pl_id := GET_PARAMETER_LIST(pl_name);
    IF ID_NULL(pl_id) THEN
        1 pl_id := CREATE_PARAMETER_LIST(pl_name);
    ELSE
        DELETE_PARAMETER(pl_id,'deptno');
    END IF;
    2 ADD_PARAMETER(pl_id,'deptno',TEXT_PARAMETER,
                   to_char(:departments.department_id));
    3 OPEN_FORM('called_param',ACTIVATE,NO_SESSION,pl_id);
END;
```

ORACLE®

25 - 21

Copyright © 2009, Oracle. All rights reserved.

Creating and Passing Parameter Lists: The Calling Form

You can also pass parameters to called forms programmatically by means of a parameter list.

A parameter list is a named programmatic construct that is simply a list of parameter names and character values. The built-in OPEN_FORM optionally takes as an argument the name or ID of a parameter list. To receive this information, the called form must contain a parameter with the same name as each of those in the parameter list.

To use a parameter list, in the calling form, perform the following steps:

1. Create the parameter list (after checking that it does not already exist).
2. Add a parameter as a name/value pair text parameter.
3. Open the called form and pass the parameter list.

There are several built-ins that enable you to work with parameter lists, including:

```
GET_PARAMETER_LIST
CREATE_PARAMETER_LIST
DESTROY_PARAMETER_LIST
ADD_PARAMETER
DELETE_PARAMETER
```

Linking by Parameter Lists: The Called Form

Example:
Called form



When-New-Form-Instance trigger

```
IF :parameter.deptno IS NOT NULL THEN
    SET_BLOCK_PROPERTY('employees',
        DEFAULT WHERE, 'department_id =
        ' || :parameter.deptno);
    SET_WINDOW_PROPERTY('window1',
        TITLE, 'Employees in Department
        ' || :parameter.deptno);
END IF;
GO_BLOCK('employees');
EXECUTE_QUERY;
```

**Create parameter
in the form.**

**Use parameter name
preceded by :parameter.**

Creating and Passing Parameter Lists: The Called Form

To use a parameter in the called form, you must first create the parameter in the form, as shown in the screenshot of the Object Navigator with the parameter deptno defined. Select the Parameters node in the Object Navigator and click Create. Then change the name of the parameter to be the name that you are passing in the parameter list from the calling form.

After you have defined the parameter, you can use it in any of the called form's code by preceding the parameter name with :parameter. You can use the form independently of the calling form if you check to see whether the parameter is NOT NULL before using it or if you set the Parameter Initial Value property of the parameter.

Linking by Global Record Groups

1. Create a record group with global scope:

```
DECLARE
    rg_name      VARCHAR2(40) := 'LIST';
    rg_id        RecordGroup;
    Error_Flag   NUMBER;
BEGIN
    rg_id := FIND_GROUP(rg_name);
    IF ID_NULL(rg_id) THEN
        rg_id := CREATE_GROUP_FROM_QUERY(rg_name,
            'Select last_name, to_char(employee_id)
            from employees' GLOBAL_SCOPE);
    END IF;
```

2. Populate the record group:

```
Error_Flag := POPULATE_GROUP(rg_id);
```

3. Use the record group in any form.

ORACLE®

25 - 23

Copyright © 2009, Oracle. All rights reserved.

Sharing Global Record Groups Among Forms

The CREATE_GROUP_FROM_QUERY built-in has a scope argument that defaults to FORM_SCOPE. However, if you use GLOBAL_SCOPE, the resulting record group is global, and can be used within all forms in the application. After it is created, a global record group persists for the remainder of the run-time session. For a description of using a record group as a basis for a list of values (LOV), see the lesson titled “Creating LOVs and Editors.”

There are many other ways to use record groups that are not covered in this course.

To use a global record group, perform the following steps:

1. Use CREATE_GROUP_FROM_QUERY to create the record group with GLOBAL_SCOPE.
2. Populate the record group with the POPULATE_GROUP built-in.
3. Use the record group in any form in the same session.

Linking by Package Variables in Shared PL/SQL Library

Advantages:

- Uses less memory than global variables
- Can be of any data type

To use, perform the following steps:

1. Create a PL/SQL library.
2. Create a package specification with variables.
3. Attach the library to multiple forms.
4. Set variable values in the calling form.
5. OPEN_FORM with the SHARE_LIBRARY_DATA option.
6. Use variables in the opened form.

ORACLE®

25 - 24

Copyright © 2009, Oracle. All rights reserved.

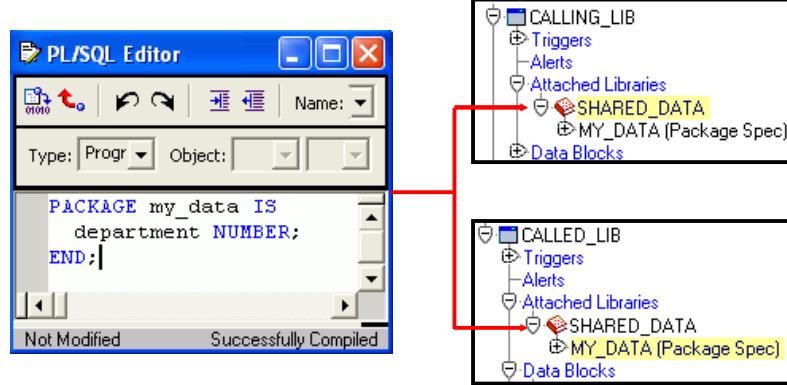
Linking by Package Variables in Shared PL/SQL Library

Perhaps the simplest and most efficient way to share data among forms is by using packaged variables in PL/SQL libraries. This enables you to use any data type, even user-defined types, to pass information between forms.

You create a library with at least a package specification that contains one or more variable declarations. You then attach that library to the calling and called forms.

When you open the called form with the SHARE_LIBRARY_DATA option, the variable value can be set and used by both open forms, making it very easy to share information among multiple forms.

Example of Linking by Shared PL/SQL Variables



```
OPEN_FORM('called_lib',ACTIVATE,
          NO_SESSION,SHARE_LIBRARY_DATA);
```

ORACLE®

Example of Linking by Shared PL/SQL Variables

For example, the slide shows the following package definition:

```
PACKAGE my_data IS
    department NUMBER;
END;
```

In the example, the package is contained in the SHARED_DATA library that is attached to two forms.

The calling form has the following When-Button-Pressed trigger:

```
my_data.department := :departments.department_id;
OPEN_FORM('called_lib',ACTIVATE,NO_SESSION,SHARE_LIBRARY_DATA);
```

The called form has the following When-New-Form-Instance trigger:

```
IF my_data.department IS NOT NULL THEN
    SET_BLOCK_PROPERTY('employees',DEFAULT_WHERE,
                      'department_id='||TO_CHAR(my_data.department));
END IF;
GO_BLOCK('employees');
EXECUTE_QUERY;
```

Summary

In this lesson, you should have learned that:

- OPEN_FORM is the primary method to call one form from another form module
- You define multiple form functionality such as:
 - Whether all forms run in the same session
 - Where the windows appear
 - Whether multiple forms should be open at once
 - Whether users should be able to navigate among open forms
 - How data will be shared among forms

ORACLE®

Summary

- You can share data among open forms with:
 - Global variables, which span sessions
 - Parameter lists, for passing values between specific forms
 - Record groups created in one form with global scope
 - PL/SQL variables in shared libraries

ORACLE®

Practice 25: Overview

This practice covers the following topics:

- Using a global variable to link the Orders and Customers forms
- Using built-ins to check whether the Orders form is running
- Using global variables to restrict a query in the Orders form



Practice 25: Overview

In this practice, you produce a multiple form application by linking the Customers and the Orders form modules.

Appendix A

Practices and Solutions

Table of Contents

Practices for Lesson 1	4
Practices for Lesson 2	5
2-1: Running the Course Application	6
Practices for Lesson 3	10
Practice 3-1: Setting Forms Builder Preferences	11
Practice 3-2: Examining the Object Navigator	12
Practice 3-3: Modifying the Appearance of a Form	13
Practice 3-4: Modifying the Design-Time Environment	15
Practices for Lesson 4	17
Practice 4-1: Creating a Form Module.....	18
Practices for Lesson 5	21
Practice 5-1: Creating a Master-Detail Form.....	22
Practice 5-2: Creating a Relation Explicitly	24
Practice 5-3: Using the Layout Wizard in Reentrant Mode.....	26
Practices for Lesson 6	27
Practice 6-1: Setting Properties.....	28
Practice 6-2: Using Visual Attributes	30
Practice 6-3: Creating Control Blocks	31
Practices for Lesson 7	33
Practice 7-1: Modifying Text Item Functionality	34
Practice 7-2: Modifying Text Item Appearance	36
Practices for Lesson 8	39
Practice 8-1: Creating Lists of Values (LOVs).....	40
Practice 8-2: Creating Editors.....	44
Practices for Lesson 9	45
Practice 9-1: Creating a List Item	46
Practice 9-2: Creating a Check Box.....	47
Practice 9-3: Creating a Radio Group.....	48
Practices for Lesson 10	49
Practice 10-1: Creating Display Items	50
Practice 10-2: Creating an Image Item	52
Practice 10-3: Creating Push Buttons	53
Practice 10-4: Creating Calculated Items	54
Practice 10-5: Creating a Bean Area Item	56
Practices for Lesson 11	57
Practice 11-1: Setting Window Properties.....	58
Practice 11-2: Creating a New Window	59
Practices for Lesson 12	60
Practice 12-1: Creating a Stacked Canvas	61
Practice 12-2: Creating a Toolbar Canvas	63
Practice 12-3: Creating a Tab Canvas.....	65
Practices for Lesson 13	68
Practices for Lesson 14	69
Practice 14-1: Using Triggers to Invoke LOVs	70

Practice 14-2: Using Triggers to Control Windows	71
Practices for Lesson 15	73
Practice 15-1: Using the Forms Debugger.....	74
Practices for Lesson 16	76
Practice 16-1: Responding to a Value Change	77
Practice 16-2: Invoking a JavaBean.....	78
Practice 16-3: Setting Properties of an Image Item	80
Practices for Lesson 17	82
Practice 17-1: Using a Customized Alert.....	83
Practice 17-2: Using a Generic Alert	84
Practices for Lesson 18	85
Practice 18-1: Populating Nonbase Table Items	86
Practice 18-2: Using the ONETIME_WHERE Clause.....	87
Practice 18-3: Determining the Mode of the Running Form	88
Practice 18-4: Enabling User Control of Queries	89
Practices for Lesson 19	91
Practice 19-1: Using an LOV for Validation	92
Practice 19-2: Using Validation Triggers	93
Practice 19-3: Using a Pluggable Java Component	95
Practices for Lesson 20	96
Practice 20-1: Initializing a Form at Startup.....	97
Practice 20-2: Initializing a Record	98
Practices for Lesson 21	100
Practice 21-1: Using Transactional Triggers to Populate Primary Keys	101
Practice 21-2: Using Transactional Triggers to Log Transaction Information.....	102
Practice 21-3: Controlling Login Behavior.....	104
Practices for Lesson 22	105
Practice 22-1: Using Object IDs	106
Practice 22-2: Writing Generic Code by Using System Variables	108
Practices for Lesson 23	110
Practice 23-1: Using an Object Group	111
Practice 23-2: Using a Property Class	112
Practice 23-3: Using an Object Library	113
Practice 23-4: Reusing PL/SQL Code	116
Practices for Lesson 24	119
Practice 24-1: Integrating WebUtil into a Form	120
Practice 24-2: Reading a Client Image File	121
Practice 24-3: Interacting with the Client Environment	122
Practice 24-4: Creating a Client File.....	123
Practices for Lesson 25	124
Practice 25-1: Calling One Form from Another	125
Practice 25-2: Sharing Data between Forms.....	126
Practice 25-3: Coding for Subsequent Queries	127

Practices for Lesson 1

There is no practice for lesson 1.

Practices for Lesson 2

In this practice, you start the Forms WebLogic Managed Server. You then run the completed course application and perform the following tasks:

- Display the key help.
- Query records.
- Insert a record.
- Update a record.
- Delete a record.
- Display a database error.

2-1: Running the Course Application

In this practice, you start an instance of the managed WebLogic Server where Forms Services are deployed. You then run the course application.

- 1) Start an instance of the WLS_FORMS managed server with a user name/password combination of weblogic/weblogic1.**
 - a) Double-click the desktop shortcut labeled Start WLS_FORMS Managed Server.
 - b) Enter the username and password when prompted.
 - c) Minimize the window when it displays the message “Server started in RUNNING mode.”
- 2) Invoke Internet Explorer and enter the following URL:**

```
http://<machine>:<port>/forms/frm servlet?form=customers.fmx
```

Your instructor will tell you the machine name and port number to use, as well as the username, password, and database for connection. For example, if the database is installed on your local machine with a single user, the values are as follows, although this may differ depending on the setup:

Machine name	localhost
Port	9001
Username	summit
Password	ORACLE
Database	ORCL

If you get an error indicating that the database is not running, such as "ORA_12541: TNS: no listener", double-click the Start Oracle Services desktop shortcut to start it.

No formal solution

- 3) Invoke the window that displays help for the shortcut keys used in the application.**
Select Help > Keys from the menu. After examining the shortcut keys, click OK to close the Keys window.
- 4) Browse through the records that were returned by the unrestricted query that executed automatically when the form started.**

2-1: Running the Course Application (continued)

Press Up and Down (or click Previous Record and Next Record) to browse through the records returned.

- 5) Execute a restricted query to retrieve information about the customer with the ID of 212.**
 - a) Put the form module in Enter-Query mode: Press F11, or select Query > Enter from the menu, or click Enter Query on the toolbar. Notice that the status line displays mode Enter-Qu... (for Enter-Query mode.)
 - b) Navigate to the Customer_Id item and enter the search value 212.
 - c) Execute the query: Press Ctrl + F11, or select Query > Execute from the menu, or click Execute Query on the toolbar. Notice that only one record is retrieved.
- 6) Execute a restricted query to retrieve the customer whose first name is “Meenakshi.”**
 - a) Put the form in Enter-Query mode again.
 - b) Move to the First Name item and enter the search value Meenakshi.
 - c) Then execute the query.
- 7) Try each of these restricted queries:**
 - a) **Retrieve all cities starting with “San”.**
 - i) Select Query > Enter.
 - ii) Click the Contact Information tab.
 - iii) Enter San% in the City item.
 - iv) Select Query > Execute.
 - b) **Retrieve all customers based in the U.S. with a low credit limit.**
 - i) Select Query > Enter.
 - ii) Click the Contact Information tab.
 - iii) Enter US in the Country_Id item.
 - iv) Click the Account Information tab.
 - v) Select the Low credit limit.
 - vi) Select Query > Execute.
- 8) Display the customer details for Harrison Sutherland, and then click Orders to invoke the Orders form module.**
 - a) Execute an unrestricted query (select Query > Execute).
 - b) Press Next Record until you see Harrison Sutherland.
 - c) Click Orders.

2-1: Running the Course Application (continued)

- 9) Click Image Off and notice that the image item is no longer displayed. Click Image On and notice that the image item is displayed.**

No formal solution

- 10) Query only those orders that were submitted online.**

- a) Select Query > Enter.
- b) Select the Online check box.
- c) Select Query > Execute. There is only one online order for Harrison Sutherland.

- 11) Move to the record in the items section of that shows Product 2322, and then click Stock. The inventory for that product is displayed in a separate window.**

No formal solution

- 12) Close the Stock Levels window. For the customer Harrison Sutherland, insert a new order, as follows:**

Item	Value
Online	Deselected
Status	New Credit Order (poplist)
Sales_Rep_Id	151 (Enter the ID, or press Ctrl + L. Then select David Bernstein from the list of values.)

- a) Click X in upper-right of the Stock Levels window.
- b) Move to orders section by clicking in the Order Date field.
- c) Select Record > Insert, or click Insert Record on the toolbar.
- d) Note that some items are already populated with default values. Then enter the values shown.

- 13) Insert a new order item with the following values:**

Item	Value
Product_Id	2289 (Enter the ID, or click the List button and select KB 101/ES.)
Quantity	2

Move to the section of the form that shows the order items and enter the values shown.

2-1: Running the Course Application (continued)

14) Save the new records.

Select Action > Save or click Save on the toolbar.

15) Update the order that you have just placed and save the change.

(Note: You may receive a message indicating that there are no changes to save. The Customers form generates this message because both forms are saved at the same time. Changes to the Orders form should be saved successfully, so you can acknowledge the message and then ignore it.)

Change the order date to last Monday and click Save.

16) Attempt to delete the order that you have just placed. What happens?

Click into the Order Date field and select Record > Remove. You receive a message that you are not able to delete the order because there are detail (item) records.

17) Delete the line item for your order and save the change.

Move to the section showing the order item and select Record > Remove. Click Save.

18) Now attempt to delete your order and save the change.

Click into the Order Date field and select Record > Remove. Click Save.

19) Exit the run-time session and close the browser window.

- a) Select Action > Exit from the menu, or click Exit on the toolbar.
- b) Close the browser window.

Practices for Lesson 3

In the practices for this lesson, you begin to work with Oracle Forms Builder, performing the following tasks:

- Setting preferences for both design time and run time
- Examining the Object Navigator
- Modifying the appearance of a form with the Layout Editor
- Modifying the design-time environment

Practice 3-1: Setting Forms Builder Preferences

In this practice, you modify preferences pertaining to welcome dialog boxes and the one button run.

- 1) Set your preferences so that a Welcome page is displayed when you first open Forms Builder and when you use any of the wizards.**
 - a) Open Forms Builder by double-clicking the desktop shortcut.
 - b) Select Edit > Preferences from the default menu system.
 - c) Click the Wizards tab in the Preferences dialog box.
 - d) Select all check boxes and click OK.
- 2) Set the run-time preferences for Forms Builder to use the WLS_FORMS managed WebLogic Server to test your applications. Set the Application Server URL by clicking “Reset to Default,” which will enter the following settings:**

URL Component	Value
Machine name	127.0.0.1 (or your local machine name)
Port	9001 (for the WLS_FORMS managed WebLogic Server)
Pointer to Forms Servlet	forms/frm servlet

- a) From the Forms Builder menu, select Edit > Preferences.
- b) Click the Runtime tab.
- c) Click “Reset to Default”.
- d) Click OK.

Practice 3-2: Examining the Object Navigator

In this practice, you examine the structure of the Forms Builder Object Navigator.

- 1) Invoke Forms Builder. If the Welcome page is displayed, select Open an existing form. If the Welcome page is not displayed, select File > Open.**

No formal solution

- 2) Open the `Orders.fmb` form module from the Open Dialog window.**

No formal solution

- 3) Close the Orders form.**

With the Orders form selected, select File > Close from the menu.

- 4) Open the `Summit.fmb` form module.**

No formal solution

- 5) Expand the Data Blocks node.**

No formal solution

- 6) Expand the Database Objects node. If you cannot expand the node, connect to the database and try again. What do you see below this node?**

No formal solution (To connect to the database, click the connect icon in the menu bar and enter the database connection information. See Practice 2-1, step 2.)

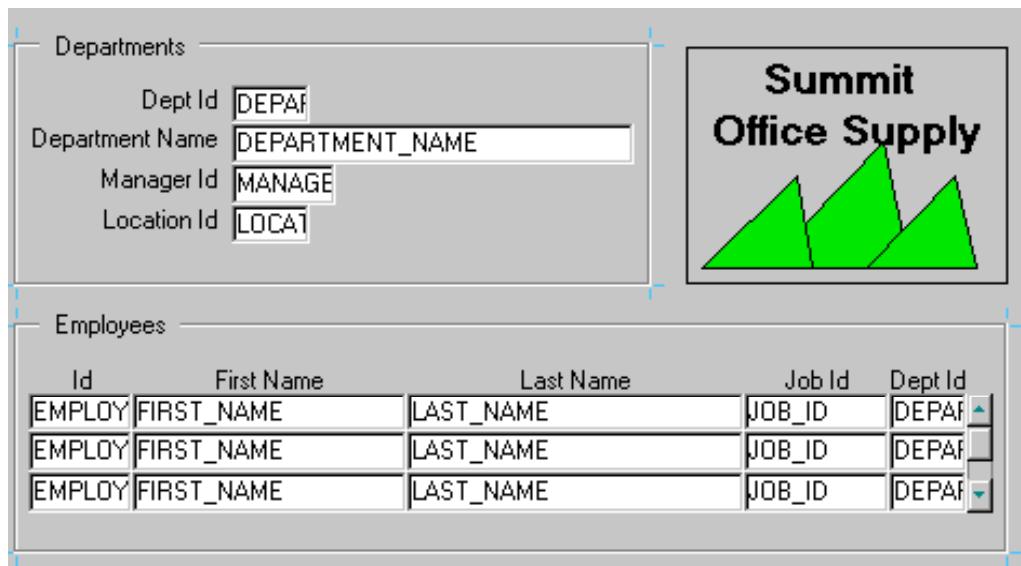
- 7) Collapse the Data Blocks node. Leave the form open in the Object Navigator.**

No formal solution

Practice 3-3: Modifying the Appearance of a Form

In this practice, you use the Layout Editor to change the arrangement of items on a canvas.

- 1) Change the layout of the `Summit.fmb` form module to match the following screenshot. At the end, save your work. Perform the following layout changes:**



- a) Invoke the Layout Editor.**

With the Summit form selected, select Tools > Layout Editor from the menu.

- b) Move the three summit shapes to the top-right corner of the layout. Align the objects along the bottom edge.**

- i) Shift-click each of the three shapes to select them together.
 - ii) Move them to the top-right corner of the layout.
 - iii) With all the three shapes still selected, select Layout > Align Components, and select the option to Align Bottom.

- iv) Click OK.

- c) Select the summit shape in the middle and place it behind the other two shapes.**

- i) Click outside the shapes to deselect them.
 - ii) Then select the middle summit shape and select Layout > "Send to Back".

- d) Draw a box with no fill around the summit shapes.**

- i) Select the Rectangle tool from the Tool Palette and draw a rectangle around the three summit shapes.
 - ii) With the rectangle still selected, click the Fill Color tool and select No Fill.

Practice 3-3: Modifying the Appearance of a Form (continued)

- e) Add the text **Summit Office Supply** in the box. If necessary, enlarge the box. Select the text tool from the Tool Palette, and then enter the text within the rectangle.
 - i) Choose a suitable font size and style.
 - ii) Click outside the text. From the menu, select Layout > Justify > Center.
- f) **Move the Manager_Id and Location_Id items to match the screenshot.**
 - i) Select and move the Manager_Id and Location_Id items below the Department_Name item.
 - ii) Press Shift and select the Dept_Id, Department_Name, Manager_Id, and Location_Id items to select them together.
 - iii) Click Align Left to align these items.
 - iv) To distribute them evenly, select Layout > Align Components, and then select the Distribute option in the Vertically column.
 - v) Click OK.
- g) **Move the First_Name item up to align it at the same level as the Last_Name item.**

Select the First_Name and Last_Name items together, and click Align Top on the toolbar.
- h) **Resize the scroll bar to make it the same height as the three records in the Employees block.**

Select the scroll bar and resize it with the mouse.
- i) **Save the form module.**

In the Object Navigator, select File > Save (or click Save).

Practice 3-4: Modifying the Design-Time Environment

In this practice, you modify the Forms Builder environment so that icons display on buttons at design time.

- 1) In Forms Builder, open and run the Customers form located in your local directory. (You must have the WLS_FORMS managed server running first as described in Practice 2-1, step 1).**

Note: Run-time fonts may look different from the fonts used in Forms Builder because Forms Builder uses operating system-specific fonts (but at run time only Java fonts are used). In addition, the appearance is different from the Layout Editor because the Oracle (rather than the Generic) look and feel is used by default.

- a) Click Open on the toolbar, or select File > Open from the menu.
Open customers.fmb.
 - b) Click Run Form, or select Program > Run Form from the menu.
Enter the connect information in the Logon dialog box and click Connect.

- 2) Click the Account Information tab. You should be able to see the image of a flashlight on the List button. Exit the run-time session and close the browser window.**

No formal solution

- 3) In Forms Builder, open the Layout Editor for the CV_CUSTOMER canvas by expanding the Canvases node in the Object Navigator and double-clicking the CV_CUSTOMER canvas icon. In the Layout Editor, click the Account Information tab. What do you observe about the List button?**

The List button displays without an image.

- 4) From the Windows Start menu, select Run, enter regedit, and click OK. Expand the registry nodes: HKEY_LOCAL_MACHINE > SOFTWARE > ORACLE. Click the ORACLE node or one of the HOME nodes beneath it; your instructor will tell you which node to open. Ensure that you have opened the correct node by verifying that the FORMS_PATH key exists in that node.**

No formal solution

Practice 3-4: Modifying the Design-Time Environment (continued)

- 5) Set the path for Forms Builder to locate icons, which are in the \icons subdirectory of your lab directory.**
 - a) Double-click the UI_ICON key to open it for editing.
 - b) For the value data, append the path to the .gif file that you will use for the button icon.
 - c) Separate this path from the remainder of the string with a semicolon (for example, ;e:\labs\lab\icons), and then click OK.
- 6) Similarly, set the value for Forms Builder to use for the icon extension, and then close the registry editor.**
 - a) If the UI_ICON_EXTENSION key exists, ensure that it is set to “gif.”
 - b) If it does not exist, from the registry menu, select Edit > New > String Value.
 - c) Enter the name UI_ICON_EXTENSION.
 - d) Double-click the string to open it for editing.
 - e) For the value data, enter gif, and then click OK.
- 7) Close and reopen Forms Builder. Open the Customers form and verify that the flashlight icon is now displayed in the Layout Editor.**

No formal solution

Practices for Lesson 4

In this practice, you learn how to use the Data Block Wizard and the Layout Wizard.

Practice 4-1: Creating a Form Module

In this practice, you use the Data Block and Layout Wizards to create a form module with a single-record data block.

- 1) Create a new form module and invoke the Data Block Wizard.**
 - a) If you are not already in Forms Builder, run Forms Builder and create a new form module by selecting “Use the Data Block Wizard” from the Welcome Wizard.
 - b) If you are already in Forms Builder, create a new form module by selecting File > New > Form or by highlighting the Forms node in the Object Navigator and clicking Create. To begin creating a block, select Tools > Data Block Wizard from the menu.
- 2) Create a new single block by using the Data Block Wizard.**

Base it on the CUSTOMERS table and include all columns.

Automatically invoke the Layout Wizard when you are through with creating the data block.

 - a) Select the block type as Table or View.
 - b) Set the Table or View field to CUSTOMERS.
 - c) Click Refresh. Click >> to include all columns, and then click Next.
 - d) Click Next, and select the “Create the data block, then call the Layout Wizard” option.
 - e) Click Finish.
- 3) Display the CUSTOMERS block on a new content canvas called CV_CUSTOMER and display one record at a time. Set the frame title to Customers. Set prompts and widths as shown in the following table:**

Name	Prompt	Width	Height
CUSTOMER_ID	ID	36	14
CUST_FIRST_NAME	First Name	95	14
CUST_LAST_NAME	Last Name	95	14
CUST_ADDRESS_STREET	Address	185	14
CUST_ADDRESS_POSTAL	Zip	50	14
CUST_ADDRESS_CITY	City	140	14
CUST_ADDRESS_STATE	State Province	50	14
CUST_ADDRESS_COUNTRY	Country Id	14	14
PHONE_NUMBERS	Phone	140	14
NLS_LANGUAGE	Nls Language	18	14
NLS_TERRITORY	Nls Territory	140	14
CREDIT_LIMIT	Credit Limit	54	14
CUST_EMAIL	Cust Email	140	14
ACCOUNT_MGR_ID	Account Mgr Id	36	14

- a) In the Layout Wizard, select [New Canvas] and make sure that the Type field is set to Content. Click Next.

Practice 4-1: Creating a Form Module (continued)

- b) Include all items and click Next.
 - c) Set values for prompts as shown, and then click Next.
 - d) Set Style to Form, and click Next.
 - e) Set Frame Title to Customers, and click Finish.
 - f) In Object Navigator, rename the canvas CV_CUSTOMER:
 - i) Select the canvas.
 - ii) Click the name.
 - iii) The cursor changes to an I-beam; edit the name and press Enter.
- 4) Change the form module name in the Object Navigator to CUSTOMERS.**
- a) Select the form module.
 - b) Click the name. The cursor changes to an I-beam.
 - c) Edit the name, and then press Enter.
- 5) Save the new module to a file called CUSTXX, where XX is the group number that your instructor has assigned to you.**
- Click Save and provide the file name in the file dialog box.
- 6) Run your form module and execute a query.**
Navigate through the fields. Exit the run-time session and return to Forms Builder.
- No formal solution
- 7) In the Layout Editor, reposition the items and edit item prompts so that the canvas resembles the following:**

The screenshot shows the Oracle Forms Layout Editor interface. A canvas titled "Customers" contains several input fields with their corresponding prompts:

- ID [CUSTOM] Last Name [CUST_LAST_NAME]
- Address [CUST_ADDRESS_STREET_ADDRESS]
- State Province [CUST_ADDF] Zip [CUST_ADDF] Country Id [CU]
- Phone [PHONE_NUMBERS]
- Credit Limit [CREDIT_LIMI] Account Mgr Id [ACCOUNT]
- Nls Language [NLS]
- First Name [CUST_FIRST_NAME]
- City [CUST_ADDRESS_CITY]
- Cust Email [CUST_EMAIL]
- Nls Territory [NLS_TERRITORY]

Hint: First resize the canvas. Do not attempt to resize the frame, or the items will revert to their original positions.

Practice 4-1: Creating a Form Module (continued)

- a) Reposition the items by dragging them.
 - b) Use the Align buttons to line the items up with one another.
 - c) Edit the following item prompts to include a carriage return as pictured (double-click in the item prompt to edit it—the cursor changes to an I-beam.):
 - i) Last Name
 - ii) First Name
 - iii) State Province
 - iv) Country Id
 - v) Credit Limit
 - vi) Account Mgr Id.
- 8) Test the form again. When finished, exit the run-time session and close the browser.**

Practices for Lesson 5

In the practices for this lesson, you create a master-detail form module and then create a third detail block with an explicitly created relationship between it and the other detail block. The result is a master-detail-detail form. You also modify the layout of a form by using the Layout Wizard in reentrant mode.

Practice 5-1: Creating a Master-Detail Form

In this practice, you create a master-detail form to display orders and order items.

1) Create a new form module.

Create a new block by using the Data Block Wizard.

Base it on the ORDERS table and include all the columns except ORDER_TOTAL and PROMOTION_ID.

Display the ORDERS block on a new content canvas called CV_ORDER and show just one record at a time. Use a form style layout. Set the frame title to Orders.

- a) Create a new form module by selecting File > New > Form or by clicking Create. Select Tools > Data Block Wizard to create a block.
- b) Select the block type as Table or View.
- c) Set the Table or View field to ORDERS.
- d) Click Refresh and include all the columns except ORDER_TOTAL and PROMOTION_ID.
- e) On the last page of the wizard, select the “Create the data block, then call the Layout Wizard” option.
- f) Click Finish.
- g) In the Layout Wizard, specify a new canvas and make sure that the Type field is set to Content.
- h) Include all the items.
- i) Leave prompts, widths, and heights at their default values.
- j) Set Style to Form.
- k) Set Frame Title to Orders, and click Finish.
- l) In the Object Navigator, rename the canvas CV_ORDER.

2) Create a new block by using the Data Block Wizard.

Base the block on the ORDER_ITEMS table and include all the columns.

Create a relationship and select the master block as ORDERS.

Display all the items except Order_Id on the CV_ORDER canvas.

Display six records in this detail block on the same canvas as the master block.

Use a tabular style layout and include a scroll bar.

Change the order of the blocks in the Object Navigator, moving the

ORDER_ITEMS block after the ORDERS block. Set the frame title to Items.

- a) In the same module, create a new block by selecting Tools > Data Block Wizard. (Ensure that you do not have the Orders block or frame selected in the Object Navigator when you invoke the Data Block Wizard, or it will be in reentrant mode to modify that block. If this happens, click Cancel, select a different object in the form, and invoke the Data Block Wizard again.)
- b) Select block type as Table or View.

Practice 5-1: Creating a Master-Detail Form (continued)

- c) Set Base Table to ORDER_ITEMS.
 - d) Include all the columns.
 - e) Click Create Relationship (ensure that Auto-join data blocks is selected.)
 - f) Select the ORDERS block as the master block, and then click OK.
 - g) Click Finish.
 - h) Use the Layout Wizard to create a layout.
 - i) Select Canvas as CV_ORDER.
 - j) Include all the items except Order_Id.
 - k) Do not change any prompts, widths, or heights.
 - l) Set Style to Tabular.
 - m) Set Frame Title to Items.
 - n) Set Records Displayed to 6.
 - o) Select the Display Scrollbar check box.
 - p) Click Finish.
 - q) In the Object Navigator, if ORDER_ITEMS is displayed first, drag the ORDER_ITEMS block to a position below the ORDERS block.
- 3) **Save the new module to a file called ORDXX, where XX is the group number that your instructor has assigned to you.**

No formal solution

Practice 5-2: Creating a Relation Explicitly

In this practice, you create an Inventories data block, and you then explicitly create a relation between the Inventories block as the detail and Order_Items as the master.

- 1) Create a new block named INVENTORIES (do not create any relationships with other blocks at this time) to display on a different canvas.**

Base it on the INVENTORIES table.

Display four records in this block and ensure that they are displayed on a new content canvas called CV_INVENTORY.

Use a tabular style layout and include a scroll bar.

In the Object Navigator, move the INVENTORIES block after the ORDER_ITEMS block. Set the frame title to Stock.

Do not create any relationships between blocks at this stage.

- a) In the same module, create a new block by selecting Tools > Data Block Wizard. (Ensure that you do not have a frame selected in the Object Navigator when you invoke the Data Block Wizard, or it will be in reentrant mode to modify that frame. If this happens, click Cancel, select a different object in the form, and invoke the Data Block Wizard again.)**
- b) Select block type as Table or View.**
- c) Set Base Table to INVENTORIES.**
- d) Include all the columns.**
- e) Click Finish.**
- f) Use the Layout Wizard to create a layout.**
- g) Select a New Canvas.**
- h) Include all the items.**
- i) Do not change any prompts, widths, or heights.**
- j) Set Style to Tabular, set Frame Title to Stock, and set Records Displayed to 4.**
- k) Select the Display Scrollbar check box.**
- l) Click Finish.**
- m) In the Object Navigator, rename the canvas CV_INVENTORY.**
- n) In the Object Navigator, if INVENTORIES is not displayed last in the block list, drag the INVENTORIES block after the ORDER_ITEMS block.**

- 2) Explicitly create a relation called Order_Items_Inventories between the ORDER_ITEMS and INVENTORIES blocks.**

Ensure that line item records can be deleted independently of any related inventory.

Set the coordination so that the INVENTORIES block is not queried until you explicitly execute a query.

- a) Create the relation: Select the Relations node in the ORDER_ITEMS block in the Object Navigator and click Create. The New Relation dialog box appears.**

Practice 5-2: Creating a Relation Explicitly (continued)

- b) Select INVENTORIES as the detail block.
- c) Select the Isolated option.
- d) Select Deferred and deselect Auto Query.
- e) Enter the join condition:
`order_items.product_id = inventories.product_id`
- f) Click OK.

Practice 5-3: Using the Layout Wizard in Reentrant Mode

In this practice, you modify the layout of data blocks by invoking the Layout Wizard in reentrant mode.

- 1) On the ORDER_ITEMS block, change the prompt for the Line_Item_Id item to Item# by using the reentrant Layout Wizard. First select the relevant frame in the Layout Editor, and then use the Layout Wizard.**
 - a) Select the frame for the ORDER_ITEMS block under the CV_ORDER canvas in the Object Navigator or in the Layout Editor, and select Tools > Layout Wizard from the menu.
 - b) Click the Items tab page.
 - c) Change the prompt for the Line_Item_Id item to Item#, and then click Finish.
- 2) In the INVENTORIES data block, change the prompt for “Quantity on Hand” to In Stock by using the Layout Wizard.**
 - a) In the Object Navigator or in the Layout Editor, select the frame that is associated with the INVENTORIES data block.
 - b) Select Tools > Layout Wizard from the menu.
 - c) Click the Items tab page and change the prompt for “Quantity on Hand” to In Stock, and then click Finish.
- 3) Click Run Form to run your form module.**
Execute a query.
Navigate through the blocks so that you see the INVENTORIES block. Execute a query in the INVENTORIES block.
Exit the run-time session, close the browser, and return to Forms Builder.
To navigate through the blocks, select Block > Next from the menu or click Next Block on the toolbar.
- 4) Change the form module name in the Object Navigator to ORDERS, and then save.**
No formal solution

Practices for Lesson 6

In the practices for this lesson, you use the Property Palette to set properties. You also create and use a visual attribute and control blocks.

Practice 6-1: Setting Properties

In this practice, you set block and frame properties in the Property Palette.

- 1) In the Customers form, ensure that the records retrieved in the CUSTOMERS block are sorted by the customer's ID.**
 - a) Open the Customers form.
 - b) In the Object Navigator, double-click the CUSTOMERS block to invoke the Property Palette.
 - c) In the Property Palette for the CUSTOMERS data block, set the ORDER BY Clause property to Customer_Id.
- 2) Set the frame properties for the CUSTOMERS block as follows:**
Remove the frame title, and ensure that the frame does not update automatically. After you have done this, you may resize the frame if desired without having the items revert to their original positions.
 - a) In the Layout Editor for the CV_CUSTOMER canvas, select the frame associated with the CUSTOMERS block and open the Property Palette.
 - b) Remove the Frame Title property value and set the Update Layout property to Manually.
 - c) You may now resize the frame to encompass all the items if this did not occur automatically when you deleted the frame title.
- 3) In the Orders form, ensure that the records retrieved in the ORDERS block are sorted by the Order_Id.**
 - a) Open the Orders form.
 - b) In the Object Navigator, double-click the ORDERS block to invoke the Property Palette.
 - c) In the Property Palette for the ORDERS data block, set the ORDER BY Clause property to Order_Id.
- 4) For the ORDER_ITEMS block, change the number of records displayed to 4 and resize the scroll bar accordingly.**
 - a) In the Object Navigator, select the ORDER_ITEMS block and open the Property Palette.
 - b) Set the "Number of Records Displayed" property to 4.
 - c) In the Layout Editor for the CV_ORDER canvas, resize the scroll bar to match the number of records displayed.
- 5) Ensure that the records retrieved in the ORDER_ITEMS block are sorted by the Line_Item_Id.**
For the ORDER_ITEMS data block, set the ORDER BY Clause property to Line_Item_Id.

Practice 6-1: Setting Properties (continued)

- 6) Set the ORDER_ITEMS block to automatically navigate to the next record when the user presses Next Item while the cursor is in the last item of a record.**

For the ORDER_ITEMS block, set the Navigation Style to Change Record.

- 7) Set the frame properties for all blocks as follows:**

Remove the frame title and ensure that the frame does not update automatically.

- a) In the Object Navigator, expand all nodes under the Canvases node.
- b) Multiselect all frames under the Graphics nodes and open the Property Palette.
- c) Remove the Frame Title property value and set the Update Layout property to Manually.

Practice 6-2: Using Visual Attributes

In this practice, you create a visual attribute and use it to highlight the current record in multiple blocks.

- 1) In the Orders form, ensure that the current record is displayed differently from the others in both the ORDER_ITEMS and INVENTORIES blocks.**

Create a Visual Attribute called Current_Record.

Using the Color Picker, set the foreground color to white and the background color to gray. Using the Pattern Picker, set the pattern to a light and unobtrusive pattern. Using the Font Picker, set the font to MS Serif, Italic, 10 pt. (If that font is not available on your window manager, use any available font.)

Use the multiple selection feature on both data blocks to set the relevant block property to use this Visual Attribute.

- a) In the Object Navigator, select the Visual Attributes node and click Create to create a new Visual Attribute.
- b) In the Property Palette, set the Name property to CURRENT_RECORD. Select the Foreground Color property and click the More button, which is labeled “...”.
- c) The Foreground Color dialog box is displayed. Set Foreground Color to white.
- d) Repeat the process to set the Background Color to gray.
- e) In the Property Palette, select the Fill Pattern property and click More.
- f) Select the third pattern from the left in the top row, which sets the pattern to gray3.3. (You can enter this value if you do not want to use the Pattern Picker.)
- g) In the Property Palette, select the Font category heading. (Do not select any of the properties under the Font category heading.)
- h) Click More and the Font dialog box appears.
- i) Select MS Serif, Italic, 10 pt., and click OK.
- j) In the Object Navigator, to use the multiple selection feature, select both of the ORDER_ITEMS and the INVENTORIES blocks by pressing Ctrl-click, and then open the Property Palette.
- k) Set the Current Record Visual Attribute Group property to CURRENT_RECORD.

Practice 6-3: Creating Control Blocks

In this practice, you create nonbase table blocks in both the Customers form and in the Orders form. In a later practice, you add objects to these control blocks.

1) Create a control block in the Customers form.

Create a new block manually, and rename this block CONTROL.

Set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed database properties to No. Set the Query Data Source Type property to None. Set the Single Record property to Yes. Leave other properties as default.

Move the CONTROL block after the CUSTOMERS block.

- a) Select the Data Blocks node in the Object Navigator.
- b) Click the Create icon in the Object Navigator, or select the Edit > Create option from the menu to create a new data block.
- c) Select the Build a new data block manually option.
- d) Rename this new data block CONTROL.
- e) Right-click this block, and open the Property Palette.
- f) In the Database category, set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed database properties to No, and set the Query Data Source Type property to None.
- g) In the Records category, set the Single Record property to Yes.
- h) Leave other properties as default.
- i) In the Object Navigator, if the CONTROL block is not displayed last, drag the CONTROL block after the CUSTOMERS block.

2) Create a CONTROL block in the Orders form.

Create a new block manually, and rename this block CONTROL.

Set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed database properties to No. Set the Query Data Source Type property to None. Set the Single Record property to Yes. Leave other properties as default.

Position the CONTROL block after the INVENTORIES block in the Object Navigator.

- a) Select the Data Blocks node in the Object Navigator.
- b) Click the Create icon in the Object Navigator, or select the Edit > Create option from the menu to create a new data block.
- c) Select the Build a new data block manually option.
- d) Rename this new data block CONTROL.
- e) Right-click this block, and open the Property Palette.
- f) In the Database category, set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed properties to No and set the Query Data Source Type property to None.

Practice 6-3: Creating Control Blocks (continued)

- g) In Records category, set the Single Record property to Yes.
- h) Leave other properties as default.
- i) In the Object Navigator, drag the CONTROL block after the INVENTORIES block.

3) Save and run the Orders form.

Test the effects of the properties that you have set.

Note: The Compilation Errors window displays a warning that advises you that the CONTROL block has no items. This is expected until you add some items to the CONTROL block in a later practice. You can safely ignore this warning.

- a) Execute a query. Ensure the following:

- Queried orders are sorted by the ID.
- Order items are sorted by line item ID.
- Four order items are displayed, with a scroll bar of the appropriate size.
- When you tab out of the last item in the ORDER_ITEMS block, you navigate to the next record instead of the beginning of the current record.
- The current record is highlighted in both the ORDER_ITEMS and the Inventories blocks.
- No frame titles appear in any of the blocks.

- b) Exit the form and close the browser window.

4) Save and run the Customers form.

Test the effects of the properties that you have set.

Note: The Compilation Errors window displays a warning that advises you that the CONTROL block has no items. This is expected until you add some items to the CONTROL block in a later practice. You can safely ignore this warning.

- a) Execute a query.
- b) Ensure that queried records are sorted by the customer ID and that the frame title is no longer displayed.
- c) Exit the form and close the browser window.

Practices for Lesson 7

In the practices for this lesson, you modify the functionality and appearance of text items. You perform the following tasks:

- Define a multiline text item.
- Set the initial value of a primary key item to use a database sequence.
- Set the initial value of a date item to use the current date.
- Control navigation between text items.
- Set format masks, justification, and the number of occurrences of a text item in a multirecord block.

Practice 7-1: Modifying Text Item Functionality

In this practice, you add to the functionality of text items by defining a multiline text item, setting initial values, and controlling navigation.

- 1) In the Customers form, remove the unneeded NLS_Language and NLS_Territory items, and then change the Phone_Numbers item to accept multiline text to display. The database column is long enough to accept two phone numbers if the second one is entered without “+1” in front of the number.**
 - a) In the Layout Editor, select and delete the two NLS items.
 - b) For the Phone Numbers item, set Multiline to Yes. Set Height to 30 and Width to 100.
- 2) Automatically display a unique, new customer number for each new record and ensure that it cannot be changed. Use the CUSTOMERS_SEQ sequence.**
 - a) In the Property Palette for Customer_Id, set Initial Value to :SEQUENCE.customers_seq.nextval.
 - b) Set the properties Insert Allowed and Update Allowed to No.
- 3) In the ORDERS block of the Orders form, create a two new text items called Customer_Name and Sales_Rep_Name.**

Ensure that these items are not associated with the ORDERS table.
Do not allow insert, update, or query operations on these items, and make sure that navigation is possible only by means of the mouse. Set the Prompts to Customer Name and Sales Rep Name. Display these items on CV_ORDER canvas.

 - a) Create two text items in the ORDERS block and name them Customer_Name and Sales_Rep_Name.
 - b) Item Type should be set to Text Item.
 - c) Set the Database Item, Insert Allowed, Update Allowed, Query Allowed, and Keyboard Navigable properties to No.
 - d) Set Prompt of the first item to Customer Name and of the second item to Sales Rep Name.
 - e) Set Prompt Attachment Offset to 5.
 - f) Set the Canvas property to CV_ORDER.
- 4) Set the relevant property for Order_Date, so that it displays the current date whenever a new record is entered.**

For Order_Date, set Initial Value to: \$\$date\$\$

Practice 7-1: Modifying Text Item Functionality (continued)

- 5) In the ORDER_ITEMS block, create a new text item called Item_Total. Ensure that Item_Total is not associated with the ORDER_ITEMS table. Do not allow insert, update, or query operations on this item and make sure that navigation is possible only by means of the mouse. Allow numeric data only and display it by using a format of 999G990D99. Set the Prompt text to Item Total. Display this item on the CV_ORDER canvas.
- a) Create a text item in the ORDER_ITEMS block and name it Item_Total.
 - b) Set the Item Type to Text Item.
 - c) Set the Database Item, Insert Allowed, Update Allowed, Query Allowed, and Keyboard Navigable properties to No.
 - d) Set the Data Type to Number.
 - e) Set the Prompt to Item Total.
 - f) Set the Prompt Attachment Edge to Top.
 - g) Set the Prompt Alignment to Center.
 - h) Set the Height to 14.
 - i) Set the Canvas property to CV_ORDER.
- 6) Alter the Unit_Price item, so that navigation is possible only by means of the mouse and updates are not allowed.

For Unit_Price, set Keyboard Navigable and Update Allowed to No.

Practice 7-2: Modifying Text Item Appearance

In this practice, you alter the appearance of text items by setting format masks and justification. You also change the number of records displayed for a text item in a multirecord block. You then rearrange items on the canvas and run the forms to test the functionality and appearance.

- 1) In the Orders form, set the Unit Price and the Item Total text items in the ORDER_ITEMS block to use a thousands separator and display two decimal places, with a leading zero if there are no digits to the left of the decimal. If you need help with format elements, see http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/sql_elements004.htm#SQLRF00211.
 - a) Multiselect the two text items.
 - b) In the Property Palette, set the Format Mask to 999G990D99.
- 2) Justify the values of Unit_Price, Quantity, and Item_Total to the right.
Multiselect the three text items. In the Property Palette, set Justification to Right.
- 3) In the INVENTORIES block, alter the number of instances of the Product_Id, so that it is displayed just once. Make its prompt display to the left of the item.
 - a) In the Property Palette for Product_Id, set “Number of Items Displayed” to 1.
 - b) Set Prompt Attachment Edge to Start.
 - c) Set Prompt Attachment Offset to 5.
- 4) Arrange the items and boilerplate on CV_INVENTORY, so that it resembles the screenshot.

Hint: Set the Update Layout property for the frame to Manually.

No formal solution

The screenshot shows a form titled "Stock Information". At the top, there is a text input field labeled "Product Id" with the value "PRODUC". Below this is a multirecord block. The block has a header row with columns "Warehouse Id" and "In Stock". The data rows show four entries, each with "WAR" in the "Warehouse Id" column and "QUANTITY" in the "In Stock" column. The "In Stock" column contains four dropdown menus with arrows pointing up and down, indicating they are scrollable.

Practice 7-2: Modifying Text Item Appearance (continued)

- 5) In the CV_ORDER canvas of the Orders form, resize and reposition the items according to the screenshot and the following tables.**

The screenshot shows the 'Order Information' section of the CV_ORDER canvas. It includes fields for Order Id, Order Date, Order Mode, Order Status, Customer Id, Customer Name, Sales Rep Id, Sales Rep Name, and a grid of Order Items. The grid has columns for Item#, Product Id, Unit Price, Quantity, and Item Total.

ORDERS Block Items	Suggested Width	ORDER_ITEMS Block Items	Suggested Width
Order_Id	60	Line_Item_Id	25
Order_Date	65	Product_Id	35
Order_Mode	40	Unit_Price	40
Customer_Id	40	Quantity	40
Customer_Name	150	Item_Total	50
Order_Status	15		
Sales_Rep_Id	40		
Sales_Rep_Name	150		

- a) Resize items by setting the width in the corresponding Property Palette or in the reentrant Layout Wizard.
- b) Drag items in the Object Navigator to reposition in navigation order.
- 6) Save, compile, and run the form to test the changes.**
- a) Ensure that the Orders and Order Items records have the new appearance that you defined.
- b) Execute a query.

Practice 7-2: Modifying Text Item Appearance (continued)

- c) Ensure that the Customer_Name, Sales_Rep_Name, and Item_Total text items appear on the form with the appropriate prompts. These text items should be blank at this point.
 - d) Ensure that you can tab through the items in sequence, but cannot navigate to Customer_Name, Sales_Rep_Name, Item_Total, or Unit_Price by keyboard tabbing, although you should be able to click into them with the mouse.
 - e) Ensure that the values in Unit_Price and Quantity are right justified (there should not be a value in Item_Total at this point.)
 - f) Create a new record in the ORDERS block and ensure that the current date appears in the Order_Date item. Remove the new record without entering any data.
 - g) Navigate to the INVENTORIES block and ensure that it has the new appearance that you defined.
- 7) In the Customers form, resize and reposition the items. Add the boilerplate text **Customer Information**. Reorder the items in the Object Navigator. Use the screenshot as a guide.

The screenshot shows the Oracle Forms Designer interface with a form titled "Customer Information". The form contains the following items:

- Last Name: CUST_LAST_NAME
- First Name: CUST_FIRST_NAME
- Address: CUST_ADDRESS_STREET_ADDRESS
- City: CUST_ADDRESS_CITY
- State Province: CUST_ADDF
- Zip: CUST_ADDF
- Country Id: CU
- Cust Email: CUST_EMAIL
- Phone: PHONE_NUMBERS
- Account Mgr Id: ACCOUNT_M
- Credit Limit: CREDIT_LIMI

No formal solution.

- 8) Save and compile the Customers form.
Test the changes by clicking Run Form to run the form.
Note: The entire form may not be visible at this time. This will be addressed in a later lesson.
- a) Ensure that the form has the new appearance that you defined.
 - b) Execute a query; ensure that you can navigate smoothly through the items and that you can enter the phone numbers on multiple lines.
 - c) Create a new record to test that the database sequence number is retrieved into the Customer_Id item, and that you cannot insert or update this value.
 - d) Remove the record without entering any data.

Practices for Lesson 8

In the practices for this lesson, you create three LOVs and an editor. You set text items to use these created objects and run the forms to test the functionality.

Practice 8-1: Creating Lists of Values (LOVs)

In this practice, you use the LOV Wizard to create LOVs and attach them to text items.

- 1) In the Orders form, create an LOV to display product numbers and descriptions to be used with the Product_Id item in the ORDER_ITEMS block. Use the PRODUCTS table and select the PRODUCT_ID and PRODUCT_NAME columns. Assign a title of Products to the LOV. Sort the list by the product name. Assign a column width of 25 for Product_Id, and assign the LOV a width of 200 and a height of 300. Position the LOV 30 pixels below and to the right of the upper-left corner. For the Product_Id column, set the return item to ORDER_ITEMS.PRODUCT_ID. Attach the LOV to the Product_Id item in the ORDER_ITEMS block. Change the name of the LOV to PRODUCTS_LOV and the name of the record group to PRODUCTS_RG.

- a) Create a new LOV: In the Object Navigator, select the LOVs node and click Create. Click OK to use the LOV Wizard.
- b) Select the New Record Group based on a query option, and click Next.
- c) In the SQL Query Statement, enter or import the following SQL query from pr8_1.txt (if you choose to import the query, in the Open dialog box, select All Files(*.*)) from the Files of type list):

```
SELECT product_id, product_name
      FROM products
 ORDER by product_name
```

- d) Click Next.
- e) Click >>, and then Next to select both of the record group values.
- f) With the Product_Id column selected, click “Look up return item.”
- g) Select ORDER_ITEMS.PRODUCT_ID, and then click OK.
- h) Set the display width for Product_Id to 25, and click Next.
- i) Enter the title Products.
- j) Assign the LOV a width of 200 and a height of 250.
- k) Select the “No, I want to position it manually” option, and set both Left and Top to 30.
- l) Click Next.
- m) Click Next to accept the default advanced properties.
- n) Click >, and then Finish to create the LOV and attach it to the Product_Id item.
- o) In the Object Navigator, change the name of the new LOV to PRODUCTS_LOV and change the name of the new record group to PRODUCTS_RG.

Practice 8-1: Creating Lists of Values (LOVs) (continued)

- 2) In the Orders form, use the LOV Wizard to create an LOV to display sales representatives' numbers and their names. Use the EMPLOYEES table, Employee_Id, First_Name, and Last_Name columns. Concatenate the First_Name and the Last_Name columns and give the alias of Name. Select employees whose Job_Id is SA REP.
- Assign a title of Sales Representatives to the LOV. Assign a column width of 20 for ID, and assign the LOV a width of 200 and a height of 300. Position the LOV 30 pixels below and to the right of the upper-left corner. For the ID column, set the return item to ORDERS.SALES REP_ID. For the Name column, set the return item to ORDERS.SALES REP_NAME. Attach the LOV to the Sales_Rep_Id item in the ORDERS block.
- Change the name of the LOV to SALES REP LOV and the record group to SALES REP RG.
- a) Create a new LOV, selecting to use the LOV Wizard.
 - b) Select the New Record Group based on a query option, and click Next.
 - c) In the SQL Query Statement, enter or import the following SQL query from pr8_2.txt:

```
SELECT employee_id, first_name || ' ' || last_name name
      FROM employees
     WHERE job_id = 'SA REP'
 ORDER BY last_name
```
 - d) Click Next.
 - e) Click >>, and then Next to select both of the record group values.
 - f) With the Employee_Id column selected, click "Look up return item." Select ORDERS.SALES REP_ID and click OK.
 - g) With the Name column selected, click "Look up return item." Select ORDERS.SALES REP_NAME, and then click OK.
 - h) Set the display width for Employee_Id to 20 and click Next.
 - i) Enter the title Sales Representatives. Assign the LOV a width of 200 and a height of 300. Select the "No, I want to position it manually" option, and set both Left and Top to 30. Click Next.
 - j) Click Next to accept the default advanced properties.
 - k) Select ORDERS.SALES REP_ID and click > and then Finish to create the LOV and attach it to the Sales_Rep_Id item.
 - l) In the Object Navigator, change the name of the new LOV to SALES REP LOV and the new record group to SALES REP RG.

Practice 8-1: Creating Lists of Values (LOVs) (continued)

- 3) In the Customers form, use the LOV Wizard to create an LOV to display account managers' numbers and their names. Use the EMPLOYEES table, Employee_Id, First_Name, and Last_Name columns. Concatenate the First_Name and the Last_Name columns and give the alias of Name. Select employees whose Job_Id is SA_MAN.
- Assign a title of Account Managers to the LOV. Assign a column width of 20 for ID, and assign the LOV a width of 200 and a height of 300. Position the LOV 30 pixels below and to the right of the upper-left corner. For the ID column, set the return item to CUSTOMERS.ACCOUNT_MGR_ID. Attach the LOV to the Account_Mgr_Id item in the CUSTOMERS block.
- Change the name of the LOV to ACCOUNT_MGR_LOV and the record group to ACCOUNT_MGR_RG.
- a) Create a new LOV. Click OK to use the LOV Wizard.
 - b) Select the “New Record Group based on a query” option, and click Next.
 - c) In the SQL Query Statement, enter or import the following SQL query from pr8_3.txt:

```
SELECT employee_id, first_name || ' ' || last_name name
      FROM employees
     WHERE job_id = 'SA_MAN'
 ORDER BY last_name
```
 - d) Click Next.
 - e) Click >>, and then Next, to select both of the record group values.
 - f) With the EMPLOYEE_ID column selected, click “Look up return item.” Select CUSTOMERS.ACCOUNT_MGR_ID, and then click OK.
 - g) Set the display width for EMPLOYEE_ID to 20, and click Next.
 - h) Enter the title Account Managers. Assign the LOV a width of 200 and a height of 300.
 - i) Select the “No, I want to position it manually” option, and set both Left and Top to 30.
 - j) Click Next.
 - k) Click Next to accept the default advanced properties.
 - l) Click >, and then Finish, to create the LOV and attach it to the Account_Mgr_Id item.
 - m) In the Object Navigator, change the name of the new LOV to ACCOUNT_MGR_LOV and the new record group to ACCOUNT_MGR_RG.

Practice 8-1: Creating Lists of Values (LOVs) (continued)

4) Test the functionality of the LOVs.

- a) Run the Customers form and execute a query.
- b) Navigate to the Account_Mgr_Id field. You should see that the LOV lamp appears in the status bar.
- c) Press Ctrl + L to invoke the LOV. You can move or resize the LOV window as needed.
- d) Select one of the account managers and click OK. You should see the Account_Mgr_Id value change.
- e) When you finish, exit the form without saving, and close the browser window.
- f) Run the Orders form and execute a query.
- g) In a similar fashion, test the LOV functionality for Sales_Rep_Id and Product_Id.
- h) When you finish, exit the form without saving and close the browser window.

Practice 8-2: Creating Editors

In this practice, you create a custom editor for use by a text item. You then test the functionality.

- 1) In the Customers form, create an editor named PHONE_NUMBER_EDITOR and attach it to the Phone_Numbers item. Set the title to Phone Numbers, the bottom title to Max 30 Characters, the background color to gray, and the foreground color to yellow.**
 - a) Create a new editor by selecting the Editors node in the Object Navigator and clicking Create. Change the name to PHONE_NUMBER_EDITOR.
 - b) Set the X Position and Y Position properties to 175.
 - c) Set the Width property to 100 and the Height property to 100.
 - d) Set the title to Phone Numbers, the bottom title to Max 30 Characters, Background Color property to gray, and the Foreground Color property to yellow.
 - e) In the Property Palette of the Phone_Numbers item, set the Editor property to Phone_Number_Editor.
- 2) Test the functionality of the editor.**
 - a) Run the Customers form.
 - b) Execute a query and navigate to the Phone field.
 - c) Press Ctrl + E to invoke the editor. You should see the editor that you defined. Enter some additional text and click OK to enter the value in the Phone field.
 - d) When you finish, exit the form without saving and close the browser window.

Practices for Lesson 9

In the set of practices for this lesson, you convert existing text items into other input item types: a list item, a check box, and a radio group.

Practice 9-1: Creating a List Item

In this practice, you convert the Order_Status text item into a list item.

- 1) In the Orders form, convert the Order_Status item into a poplist item.**

Add list elements shown in the table below.

Display any other values as Unknown.

Ensure that the newly created records display the value “New CASH order.”

Resize the poplist item in the Layout Editor, so that the elements do not truncate at run time.

List Element	List Item Value
New CASH order	0
CASH order being processed	1
CASH Backorder	2
CASH order shipped	3
New CREDIT order	4
CREDIT order being processed	5
CREDIT Backorder	6
CREDIT order shipped	7
CREDIT order billed	8
CREDIT order past due	9
CREDIT order paid	10
Unknown	11

- a) For Order_Status, set Item Type to List Item.
- b) Set List Style to Poplist (this is the default).
- c) Set the Mapping of Other Values to 11.
- d) Set the Initial Value to 0.
- e) Select the Elements in List property, and then click More to invoke the List Elements dialog box.
- f) Enter the elements shown in the table. Enter the corresponding database values in the List Item Value box for each item in the list.
- g) Click OK to accept, and close the dialog box.
- h) Open the Layout Editor, and resize the item so that elements do not truncate.

Practice 9-2: Creating a Check Box

In this practice, you convert the Order_Mode item into a check box.

- 1) In the Orders form, convert the Order_Mode text item to a check box.**

Set the checked state of Order_Mode to represent the base-table value of **online** and the unchecked state to represent **direct**.

Ensure that new records are automatically assigned the value **online**.

Display any other values as unchecked.

Remove the existing prompt and set label to **Online?**

In the Layout Editor, resize the check box so that its label is fully displayed to the right. Resize it a little longer than needed in Forms Builder so that the label does not truncate at run time.

- a) For Order_Mode, set Item Type to Check Box.**
- b) Set Value when Checked to online.**
- c) Set Value when Unchecked to direct.**
- d) Set the Check Box Mapping of Other Values to Unchecked.**
- e) Set the Initial Value to online.**
- f) Delete the Prompt value; set the Label property to Online?.**
- g) Open the Layout Editor and resize the check box if needed so that its label is fully displayed.**

- 2) Run the Orders form to test the Order_Status list item and the Order_Mode check box.**

No formal solution.

Practice 9-3: Creating a Radio Group

In this practice, you convert the Credit_Limit Item into a radio group with three radio buttons.

- 1) In the Customers form, convert the Credit_Limit text item into a radio group. Add radio buttons for Low, Medium, and High to represent database values of 500, 2000, and 5000. Arrange as shown below:**

The screenshot shows a form interface. On the left, there is a label "Account" followed by a text input field containing "ACCOUNT_M". To the right of this, under the heading "Credit Limit:", there are three radio buttons labeled "Low", "Medium", and "High". The "Low" radio button is selected, indicated by a filled circle.

Define access keys of L for Low, M for Medium, and H for High.

Add text Credit Limit to describe the radio group's purpose.

Set Label to Low for the Low radio button, Medium for the Medium radio button, and High for the High option button.

Ensure that new records display the default of Low, and that existing records with other values display as Medium.

- a) For Credit_Limit, set Item Type to Radio Group.
 - b) Set Initial Value to 500 and the Mapping of Other Values to 2000.
 - c) In the Object Navigator, expand the Credit_Limit node.
 - d) The Radio Buttons node is displayed.
 - e) Create three buttons; name them Low_Button, Medium_Button, and High_Button.
 - f) For the first button, set Access Key to L, Label to Low, and Radio Button Value to 500.
 - g) For the second button, set Access Key to M, Label to Medium, and Radio Button Value to 2000.
 - h) For the third button, set Access Key to H, Label to High, and Radio Button Value to 5000.
 - i) Use the Layout Editor to position the radio buttons so that all can be seen.
 - j) In the Layout Editor, create boilerplate text to identify radio buttons as Credit Limit.
- 2) Run the Customers form to test the Credit_Limit radio group.**

No formal solution.

Practices for Lesson 10

In this practice for this lesson, you add several items in the Customers and Orders forms: display items, an image item, push buttons, calculated items, and a bean area item.

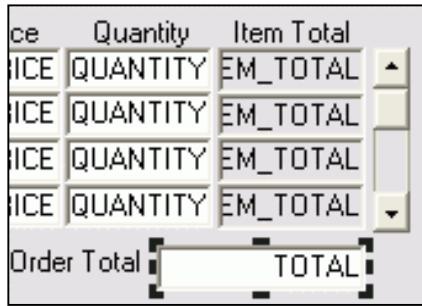
The calculated items function immediately because their functionality is defined declaratively. However, the other item types that you create in this practice do not function until programmatic logic is added in later lessons.

Practice 10-1: Creating Display Items

In this practice, you create two display items in the Order Items block of the Orders form. You also create another display item in the Control block.

- 1) In the ORDER_ITEMS block of the Orders form, create a display item called Description. Set the Prompt property to Description and display the prompt centered above the item. Rearrange the items in the layout so that all are visible.**
 - a) Open the Layout Editor, ensure that the block is set to ORDER_ITEMS, and select the Display Item tool.
 - b) Place Display Item to the right of the Product_Id. Move the other items to the right if it is needed to accommodate the display item.
 - c) Set Name property of the display item to DESCRIPTION.
 - d) Set Maximum Length to 50.
 - e) Set Width to 80. Set Height to 14.
 - f) Set Database Item to No.
 - g) Set the Prompt property to Description, the Prompt Attachment Edge property to Top, and the Prompt Alignment to Center.
- 2) Create another display item, Image_Description, in the ORDER_ITEMS block. This should synchronize with the Description item. Set the Maximum Length property to the same value as the Description item.**
 - a) Display the Layout Editor.
 - b) Ensure that the block is set to ORDER_ITEMS.
 - c) Select the Display Item tool.
 - d) Place the Display Item to the right and slightly above the frame containing the order items.
 - e) Set Name to IMAGE_DESCRIPTION.
 - f) Set “Synchronize with Item” to Description.
 - g) Set Maximum Length property to 0.
 - h) Set Database Item to No.
 - i) Set “Number of Items Displayed” to 1.
 - j) Set Width to 90. Set Height to 14.
- 3) To display the total of the item totals, create a new nondatabase display item in the CONTROL block.**
Set the Position, Size, and Prompt properties according to the screenshot.
Set the Format Mask property to 9G999G990D99.
Set the Justification property to Right.
Set the “Number of Items Displayed” property to 1.
Set the Keyboard Navigable property to No.

Practice 10-1: Creating Display Items (continued)



- a) In the Object Navigator, create a new text item in the CONTROL block.
- b) Change the Item Type to Display Item.
- c) Set the Name property to TOTAL, and the Prompt property to Order Total, with a Prompt Attachment Offset of 5.
- d) Set the Canvas property to CV_ORDER.
- e) Set the “Number of Items Displayed” property to 1.
- f) Set the Data Type property to Number.
- g) Set the Format Mask property to 9G999G990D99.
- h) Set the Justification property to Right.
- i) Set the Database Item property to No.
- j) Set Width to 70; set Height to 14.
- k) Reposition the item according to the screenshot.

Practice 10-2: Creating an Image Item

In this practice, you create an image item in the control block of the Orders form. The image will not display in the image item at this point; you will add code to populate the image item in the practices for Lesson 20.

- 1) Create an image item called Product_Image in the CONTROL block of the Orders form. (Use the CONTROL block because you do not want to pick up the Current Record Visual Attribute Group of the ORDER_ITEMS block, and this is a non-base-table item.) Position this and the other items you create in this practice so that your form resembles the screenshot.**

The screenshot shows the Oracle Forms Layout Editor with the following details:

- Header Section:** Contains fields for Order Id (ORDER_ID), Order Date (ORDER_DATE), Online? (checkbox checked), Order Status (dropdown), Customer Id (CUSTOMER_ID), Customer Name (CUSTOMER_NAME), Sales Rep Id (SALES_RI), and Sales Rep Name (SALES REP NAME).
- Image Item:** An image item labeled IMAGE: PRODUCT_IMAGE is positioned on the right side of the header. It displays a large 'X' symbol.
- Table Block:** A table block labeled IMAGE_DESCRIPTION is located below the header. It contains six rows of data, each representing a product line item. The columns are: Item#, Product Id, Description, Unit Price, Quantity, and Item Total.
- Buttons:** At the bottom of the table block are buttons for OrderTotal and TOTAL.

- a) Display the Layout Editor; ensure that the block is set to CONTROL.
- b) Select the Image Item tool.
- c) Click and drag a box and place it so that it matches the screenshot.
- d) Display the Property Palette for the image.
- e) Change the name to PRODUCT_IMAGE.
- f) Set these properties for the image item:
 - i) Keyboard Navigable: No
 - ii) Sizing Style: Adjust
 - iii) Database Item: No

Practice 10-3: Creating Push Buttons

In this practice, you create iconic buttons in the control blocks of both the Orders and the Customers forms. At this point, the buttons have no functionality, but in a later lesson you add PL/SQL triggers to these buttons to perform programmed actions when clicked.

- 1) In the CONTROL block of the Orders form, create an iconic button called Product_Lov_Button. Use the list file (do not include the .ico or .gif extension). Set the Keyboard Navigable property and the Mouse Navigate property to No.**
 - a) Display the Layout Editor and ensure that the block is set to CONTROL.
 - b) Select the Button tool.
 - c) Create a push button and place it close to Product_Id.
 - d) Set Name to PRODUCT_LOV_BUTTON.
 - e) Set the Iconic property to Yes, and set the Icon Filename to list.
 - f) Set the Keyboard Navigable property to No.
 - g) Set the Mouse Navigate property to No.
 - h) Set Width and Height to 17.
Note: If you completed Practice 3 successfully, you should now be able to see the flashlight icon on the Product_Lov_Button button in the Layout Editor.
- 2) In the CONTROL block of the Customers form, create an iconic button. Use the list file (do not include the .ico or .gif extension). Name the push button ACCOUNT_MGR_LOV_BUTTON, and place it next to Account_Mgr_Id. Set the Keyboard Navigable property and the Mouse Navigate property to No.**
 - a) In the Customers form, display the Layout Editor and ensure that the block is set to CONTROL.
 - b) Select the Button tool.
 - c) Create a push button and place it close to Account_Mgr_Id.
 - d) Resize the push button to a Width of 17 and a Height of 17.
 - e) Set Name to ACCOUNT_MGR_LOV_BUTTON.
 - f) Set Keyboard Navigable to No.
 - g) Set Mouse Navigate to No.
 - h) Set Iconic to Yes.
 - i) Set Icon File Name to list.
Note: If you completed Practice 3 successfully, you should now be able to see the flashlight icon on the Account_Mgr_Lov_Button button in the Layout Editor.

Practice 10-4: Creating Calculated Items

In this practice, you create two calculated items in the Orders form by entering calculation properties in existing items. The Item_Total calculation is formula-based, whereas Order_Total is a sum of all the item totals.

- 1) To display item total information, set the following properties for the Item_Total item in the ORDER_ITEMS block:**
 - a) Change the Item Type to Display Item.
Set the Calculation Mode property to Formula.
Set the Formula property to:
`nvl (:ORDER_ITEMS.quantity, 0) *`
`nvl (:ORDER_ITEMS.unit_price, 0)`
 - b) In the Object Navigator, select the Item_Total item in the ORDER_ITEMS block, and open the Property Palette.
 - c) Change Item Type to Display Item.
 - d) Set the Calculation Mode property to Formula.
 - e) Set the Formula property to: `nvl (:ORDER_ITEMS.quantity, 0) *`
`nvl (:ORDER_ITEMS.unit_price, 0)`
- 2) Make CONTROL.Total in the Orders form a summary item and display summaries of the Item_Total values in the ORDER_ITEMS block. Ensure that you have set the Query All Records property to Yes for the ORDER_ITEMS block.**
 - a) For the Total item in the Control block of the Orders form, set the Calculation Mode property to Summary, and the Summary Function property to Sum.
 - b) Set the Summarized Block property to ORDER_ITEMS and Summarized Item property to Item_Total.
 - c) Set the Query All Records property to Yes for the ORDER_ITEMS block; that is necessary for summary items to compute the sum of all the records.
- 3) Run the Orders form and test the changes you have made.**
 - a) Ensure that the button is displayed and that the icon appears. The button is not yet functional.
 - b) Execute a query and ensure that the new items do not cause an error. (Did you remember to switch off the Database Item property for items that do not correspond to columns in the base table?)
 - c) Ensure that the numbers reflect the format masks that you set.
 - d) Ensure that the following items exist and that you cannot tab to them with the keyboard (all of these items should be blank):
 - i) ORDERS.Customer_Name
 - ii) ORDERS.Sales_Rep_Name
 - iii) ORDERS.Items.Description

Practice 10-4: Creating Calculated Items (continued)

- iv) ORDER_ITEMS.Image_Description
- v) CONTROL.Product_Image
- e) Ensure that you cannot navigate to the following items either with keyboard or mouse: ORDER_ITEMS.Item_Total and CONTROL.Order_Total.
- f) Check that the calculations for Item_Total and Order_Total function correctly. Change some of the values in Quantity to ensure that recalculations of Item_Total and Order Total are performed.
- g) Exit the form and close the browser window when you are finished.

Practice 10-5: Creating a Bean Area Item

In this practice, you create a bean area item in the Customers form. At this point it does not function; you add functionality in the practices for a later lesson.

- 1) In the Customers form, create a bean area in the CONTROL block and name it Colorpicker. This bean has no visible component on the form, so set it to display as a tiny dot in the upper-left corner of the canvas. Ensure that users cannot navigate to the bean area item with either the keyboard or mouse. You will not set an implementation class; in Lesson 20, you learn how to programmatically instantiate the JavaBean at run time.**
 - a) Display the Layout Editor and ensure that the block is set to CONTROL.
 - b) Select the Bean Area tool and click in the upper-left corner of the canvas.
 - c) Set NAME to COLORPICKER.
 - d) Set X Position and Y Position to 0.
 - e) Set Width and Height to 1.
 - f) Set Keyboard Navigable and Mouse Navigate to No.
- 2) Run the Customers form and test the changes you have made.**
 - a) Ensure that the button appears with the icon. You are not able to test the bean area or the function of the button yet; you test these when you later add functionality to these items.
 - b) Exit the form and close the browser window when you are finished.

Practices for Lesson 11

In the practices for this lesson, you customize windows in your form modules. You resize the windows to make them more suitable for presenting canvas contents. You also create a new window to display the CV_INVENTORIES canvas.

Practice 11-1: Setting Window Properties

In this practice, you change the size, position, name, and title of the window in the Customers form. You also modify the name and title of the window in the Orders form.

- 1) Modify the window in the Customers form. Change the name of the window to WIN_CUSTOMER, and change its title to Customer Information. Check that the size and position are suitable. Run the form to test the changes.**
 - a) Set the Name property of the existing window to WIN_CUSTOMER.
 - b) Change the Title property to Customer Information.
 - c) In the Layout Editor, look at the lowest and right-most positions of objects on the CV_CUSTOMER canvas, and plan the height and width for the window.
 - d) Change the height and width of the window in the Property Palette.
 - e) The suggested size is Width 440 and Height 250, but this may be different depending on how you have arranged items on the canvas.
 - f) The suggested X, Y positions are 10, 10, respectively..
Note: Another way to change the size of the window is to select View > Show View and then resize the viewport in the Layout Editor.
 - g) Run the form and ensure that window title, size, and position are as you defined.
- 2) Modify the window in the Orders form. Ensure that the window is called WIN_ORDER. Also change its title to Orders and Items. Check that the size and position are suitable.**
 - a) Set the Name property of the existing window to WIN_ORDER.
 - b) Set the Title property to Orders and Items.
 - c) In the Layout Editor, look at the lowest and right-most positions of objects on the CV_ORDER canvas, and plan the height and width for the window.
 - d) Change the height and width of the window in the Property Palette.
 - e) The suggested size is Width 450, Height 275, but this may be different depending on how you have arranged items on the canvas.
 - f) The suggested X, Y positions are 10, 10, respectively.
Note: Another way to change the size of the window is to select View > Show View and then resize the viewport in the Layout Editor.

Practice 11-2: Creating a New Window

In this practice, you cause the contents of the INVENTORIES block to display in a new window.

- 1) In the Orders form, create a new window called WIN_INVENTORY that is suitable for displaying the CV_INVENTORY canvas. Use the rulers in the Layout Editor to help you plan the height and width of the window. Set the window title to Stock Levels. Place the new window in a suitable position relative to WIN_ORDER. Ensure that the window does not display when the user navigates to an item on a different window.**
 - a) Create a new window called WIN_INVENTORY.
 - b) Set the Title property to Stock Levels.
 - c) In the Layout Editor, look at the lowest and right-most positions of objects on the CV_Inventory canvas, and plan the height and width for the window.
 - d) Look at the CV_Order canvas to plan the X and Y positions for the window. Set the size and position in the Property Palette.
The suggested size is Width 200 and Height 160.
The suggested position is X 160, Y 100.
 - e) Set “Hide on Exit” to Yes.
- 2) Associate the CV_INVENTORY canvas with the WIN_INVENTORY window.**
Set the Window property to WIN_INVENTORY for the CV_INVENTORY canvas.
- 3) Run the Orders form to test the changes.**
 - a) Click Run Form to run the form.
 - b) Execute a query.
 - c) Ensure that window titles, sizes, and positions are as you defined.
 - d) Ensure that the INVENTORIES block is displayed in WIN_INVENTORY when you navigate to this block.
 - e) Also make sure that the WIN_INVENTORY window is hidden when you navigate to one of the other blocks.

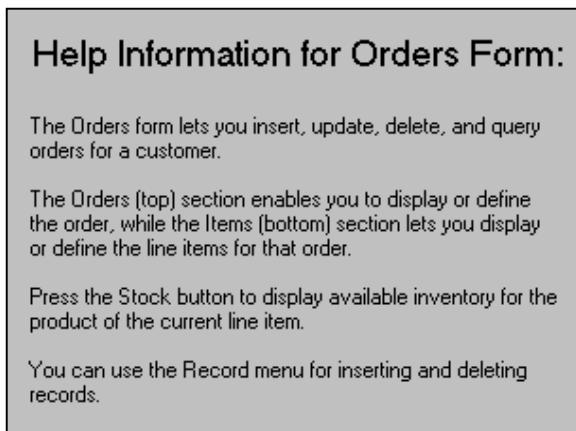
Practices for Lesson 12

In the practices for this lesson, you create different types of canvases: stacked canvas, toolbar canvas, and tab canvas.

Practice 12-1: Creating a Stacked Canvas

In this practice, you create a stacked canvas in the Orders form to provide help text.

- 1) Create a stacked canvas named CV_HELP to display help in the WIN_ORDER window of the Orders form. Suggested *visible* size is Viewport Width 250, Viewport Height 225 (points). Select a contrasting color for the canvas. Place some application help text on this canvas, similar to the following screenshot:**



- a) Create a new canvas.
 - b) If the Property Palette is not already displayed, click the new canvas object in the Object Navigator and select Tools > Property Palette.
 - c) Set Canvas Type to Stacked.
 - d) Name the canvas CV_HELP.
 - e) Assign it to the WIN_ORDER window.
 - f) Set the Viewport Width and Height and the Background Color properties.
 - g) Display the stacked canvas in the Layout Editor, and create some boilerplate text objects with help information about the form.
- 2) Position the view of the stacked canvas so that it appears in the center of WIN_ORDER. Ensure that it will not obscure the first enterable item.**
Do this by planning the top-left position of the view in the Layout Editor, while showing CV_ORDER. Define the Viewport X and Viewport Y Positions in the Property Palette. You can resize the view in the Layout Editor.
- a) In the Layout Editor, display the CV_ORDER canvas, select View > Stacked Views from the menu, and select CV_HELP from the list. You can see both canvases in this way.
 - b) Change Viewport X Position and Viewport Y Position properties in the Property Palette for CV_HELP canvas. Suggested coordinates: 100, 30.
 - c) You can also move or resize the view of the CV_HELP canvas as you see it displayed on the CV_ORDER canvas.

Practice 12-1: Creating a Stacked Canvas (continued)

- 3) Organize CV_HELP so that it is the last canvas in sequence.
Do this in the Object Navigator. (This ensures the correct stacking order at run time.)**

Drag the CV_HELP canvas so that it is the last canvas displayed under the Canvases node in the Object Navigator.

- 4) Run your form and test the changes. Note that the stacked canvas is initially displayed until you navigate to an item behind it.**

No formal solution. You may need to adjust the size of the canvas because of the Java fonts used at run time.

- 5) Switch off the Visible property of CV_HELP, and then create a push button in the CONTROL block to hide the Help information when it is no longer needed.
You will add the code later. Display this push button on the CV_HELP canvas.**

- a) Set the Visible Property to No for the CV_HELP canvas.

- b) Create a push button in the CONTROL block with the following properties:

Name: Hide_Help_Button

Label: Hide Help

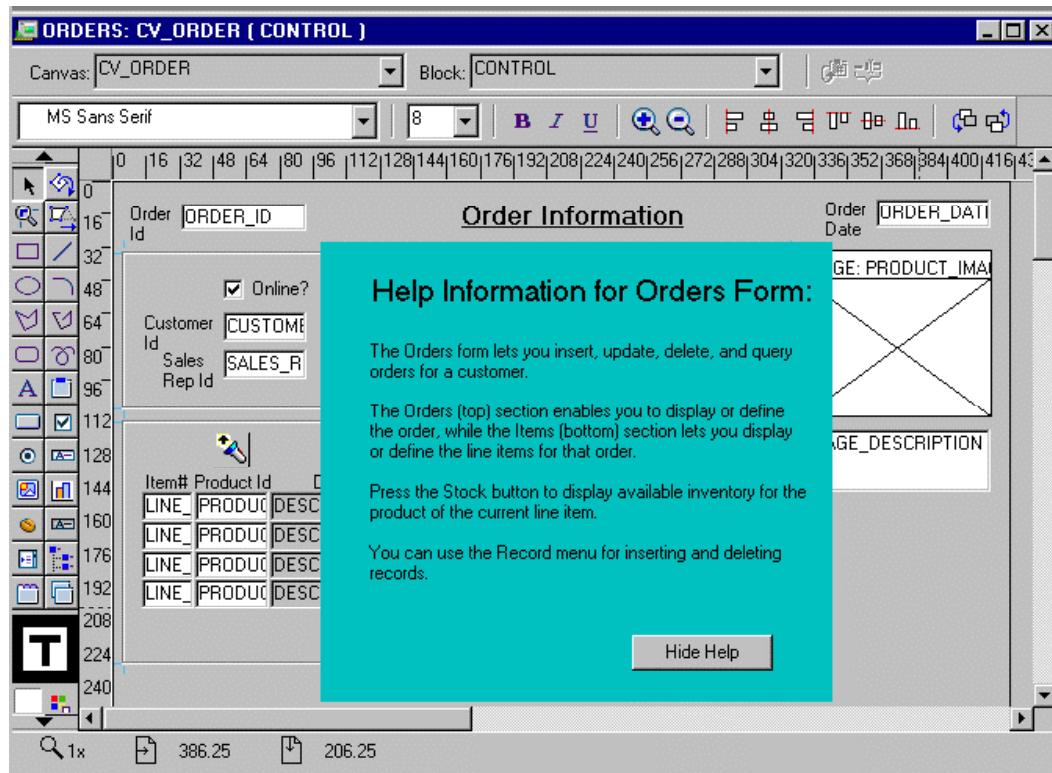
Mouse Navigate: No

Keyboard Navigable: No

Canvas: CV_HELP

Width, Height: 65, 16

X, Y Position: 150, 170



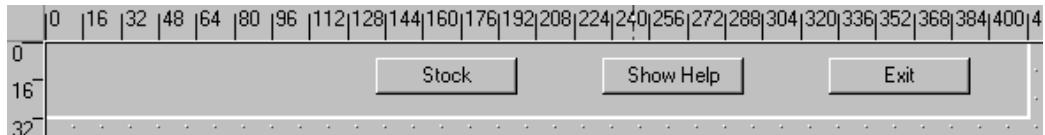
Practice 12-2: Creating a Toolbar Canvas

In this practice, you create a horizontal toolbar canvas for the Orders form.

- 1) In the Orders form, create a horizontal toolbar canvas called Toolbar in the WIN_ORDER window, and make it the standard toolbar for that window. The suggested height is 30.**
 - a) Create a new canvas.
 - b) Set Name to TOOLBAR and Height to 30.
 - c) Set Canvas Type to Horizontal Toolbar and Window to WIN_ORDER.
 - d) In the Property Palette of the WIN_ORDER window, set Horizontal Toolbar Canvas to Toolbar.
- 2) Save, compile, and run the form to test.**
Notice that the toolbar now uses part of the window space, so that the bottom of the form is truncated. Adjust the design-time window size accordingly.
Set the Height property of the WIN_ORDER window to add 30 to the height to accommodate the horizontal toolbar.
- 3) Create three push buttons in the CONTROL block, as in the following table:**

Push Button Name	Label
Stock_Button	Stock
Show_Help_Button	Show Help
Exit_Button	Exit

- a) Set the following properties on the buttons:

Mouse Navigate: No	Keyboard Navigable: No
Canvas: Toolbar	Background Color: White
Height: 16	Width: 50
- b) Place the buttons on the Toolbar canvas. Resize the Toolbar if needed.
Suggested positions for the push buttons are shown in the illustration.
A screenshot of the Oracle Forms Layout Editor. It shows a toolbar canvas with a height of 32 pixels. Three push buttons are placed on the toolbar: 'Stock' at position 16, 'Show Help' at position 288, and 'Exit' at position 384. The toolbar has a background color of white and a height of 16 pixels. The overall window height is 400 pixels, with a 32-pixel gap between the toolbar and the main form area.
- c) In the Layout Editor of the Toolbar canvas, with the CONTROL block selected, use the Button tool to create three push buttons.
- d) Select all three buttons, invoke the Property Palette, and set the Mouse Navigate, Keyboard Navigable, Height, Width, and Background Color properties.
- e) Then select each button individually and set the Name and Label properties.

Practice 12-2: Creating a Toolbar Canvas (Continued)

- f) In the Layout Editor, resize and reposition the buttons. Although gray in the Layout Editor, the buttons will be white at run time. Change the Width of the Toolbar if needed.
- 4) **Run the form to test the appearance of the canvases.**

No formal solution

Practice 12-3: Creating a Tab Canvas

In this practice, you create a tab canvas in the Customers form to organize and display the customer information on tab pages.

- 1) In the Layout Editor of the Customers form, delete the frame object that surrounds the CUSTOMERS block.**

Create a tab canvas named TAB_CUSTOMER (you may need to first enlarge the content canvas).

In the Property Palette, set the Background Color property to gray, the Corner Style property to Square, and the Bevel property to None.

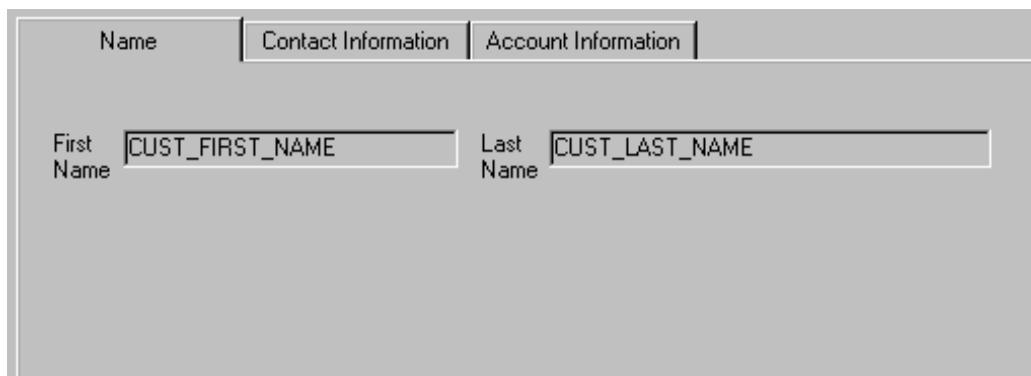
- a) In the Layout Editor, select the frame that surrounds the CUSTOMERS block and delete it.**
- b) Select the content canvas and resize it if necessary to make it large enough to accommodate the tab canvas you are about to create.**
- c) Click Tab Canvas in the toolbar. Create a tab canvas by drawing a rectangle on the content canvas.**
- d) In the Object Navigator, select the new tab canvas and open the Property Palette.**
- e) Set the Name property to TAB_CUSTOMER.**
- f) Set the Background Color property to gray, the Corner Style property to Square, and the Bevel property to None.**

- 2) Create another tab page in addition to the two tab pages created by default. Name the tab pages Name, Contact, and Account. Label them as Name, Contact Information, and Account Information.**

- a) In the Object Navigator, expand this tab canvas and create one more tab page for a total of three tab pages.**
- b) Set the tab pages Name properties as Name, Contact, and Account.**
- c) Set the Label properties as Name, Contact Information, and Account Information.**

- 3) Design the tab pages according to the following screenshots. Set the item properties to make them display on the relevant tab pages.**

- a) Place the First_Name and Last_Name items on the Name tab page.**



Practice 12-3: Creating a Tab Canvas (continued)

- i) In the Object Navigator, select the Cust_Last_Name and Cust_First_Name items of the CUSTOMERS block and open the Property Palette for this multiple selection.
 - ii) Set the Canvas property to TAB_CUSTOMER.
 - iii) Set the Tab Page property to the Name tab page.
 - iv) In the Layout Editor, arrange the items according to the screenshot.
- 4) Place the following items on the Contact tab page of the TAB_CUSTOMER canvas: Cust_Address_Street_Address, Cust_Address_City, Cust_Address_State_Province, Cust_Address_Postal_Code, Cust_Address_Country_Id, Cust_Email, and Phone Numbers. Arrange them as shown in the following screenshot:**

Name	Contact Information	Account Information
Address <input type="text" value="CUST_ADDRESS_STREET_ADDRESS"/>		
City <input type="text" value="CUST_ADDRESS_CITY"/>	State Province <input type="text" value="CUST_ADDF"/>	
Zip <input type="text" value="CUST_ADDF"/>	Country Id <input type="text" value="CUS"/>	
Email <input type="text" value="CUST_EMAIL"/>		Phone Numbers <input type="text" value="PHONE_NUMBERS"/>

- i) In the Object Navigator, select the listed items of the CUSTOMERS block.
 - ii) Open the Property Palette for this multiple selection.
 - iii) Set the Canvas property to TAB_CUSTOMER.
 - iv) Set the Tab Page property to the CONTACT tab page.
 - v) In the Layout Editor, arrange the items according to the screenshot.
 - vi) For the Phone_Numbers item, open the Property Palette and change the Prompt to Phone Numbers, the Prompt Attachment Edge to Top, and the Prompt Attachment Offset to 0.
- 5) Place the following items on the Account tab page of the TAB_CUSTOMER canvas: Account_Mgr_Id and Credit_Limit from the CUSTOMERS block, and Account_Mgr_LOV_Button from the CONTROL block.**
- a) In the Object Navigator, select the listed items of the CUSTOMERS block.
 - b) Open the Property Palette for this multiple selection.
 - c) Set the Canvas property to TAB_CUSTOMER.
 - d) Set the Tab Page property to the ACCOUNT tab page.
 - e) In the Layout Editor, arrange the items according to the screenshot and create a rectangle to surround the Credit_Limit radio buttons.

Practice 12-3: Creating a Tab Canvas (continued)

- f) In the Object Navigator, select the text object under the Graphics node of the CV_CUSTOMER canvas that corresponds with the Credit Limit boilerplate text.
- g) Drag the text object to the Account tab page.
- h) In the Object Navigator, select the text object under the Graphics node of the CV_CUSTOMER canvas that corresponds with the Credit Limit boilerplate text.
- i) Drag the text object to the Account tab page.
- j) In the Layout Editor, arrange the items according to the screenshot and create a rectangle to surround the Credit Limit radio buttons. The content canvas should now resemble the following screenshot:

The screenshot shows the Oracle Forms Layout Editor interface. At the top, there is a toolbar with various icons. Below the toolbar, the title bar displays 'ID CUSTOMER' and the main title 'Customer Information' in bold. The form has a tabular structure with three tabs: 'Name', 'Contact Information', and 'Account Information'. The 'Account' tab is currently selected, indicated by a highlighted border. Within the 'Account' tab, there are several input fields and a button labeled 'ACCOUNT'. Below these fields, there is a rectangular container containing three radio buttons labeled 'Credit Limit' with options 'Low', 'Medium', and 'High'. The 'Medium' option is currently selected, as indicated by its checked state. The entire container is enclosed in a thin black border.

- 6) Run the form to test the functionality of the tabs.

No formal solution

Practices for Lesson 13

There are no practices for this lesson.

Practices for Lesson 14

This practice focuses on how to use the When-Button-Pressed and When-Window-Closed triggers.

Practice 14-1: Using Triggers to Invoke LOVs

In this practice, you write two When-Button-Pressed triggers that invoke lists of values that you designed earlier. In the Customers form, invoke the Account Manager LOV and in the Orders form, you invoke the Products LOV.

- 1) In the Customers form, write a trigger to display the Account_Mgr_Lov when the Account_Mgr_Lov_Button is selected. To create the When-Button-Pressed trigger, use the Smart Triggers feature. Find the relevant built-in in the Object Navigator under built-in packages, and use the “Paste Name and Arguments” feature.**

- a) Right-click the Account_Mgr_Lov_Button in the Object Navigator
 - b) Select SmartTriggers in the pop-up menu, and select the When-Button-Pressed trigger from the list.
 - c) When-Button-Pressed on CONTROL.Account_Mgr_Lov_Button:

```
IF SHOW_LOV('account_mgr_lov') THEN  
    NULL;  
END IF;
```

- 2) Save, compile, and run the form. Test to see that the LOV is invoked when you press the Account_Mgr_Lov_Button.**

No formal solution

- 3) In the Orders form, write a trigger to display the Products_Lov when the Product_Lov_Button is clicked.**

When-Button-Pressed on CONTROL.Product_Lov_Button:

```
IF SHOW_LOV('products_lov') THEN  
    NULL;  
END IF;
```

- 4) Write a trigger that exits the form when the Exit_Button is clicked.**

When-Button-Pressed on CONTROL.Exit_Button:

```
EXIT_FORM;
```

- 5) Save, compile, and run the form. Test to see that the LOV is invoked when you press the Product_Lov_Button and that the form exits when you press the Exit_Button.**

No formal solution

Practice 14-2: Using Triggers to Control Windows

In this practice, you write triggers to exit the form and to show or hide certain windows. You use both When-Button-Pressed and When-Window-Closed triggers.

- 1) Code a trigger in the Customers form so that the form exits when the user attempts to close the window. Run the form to test.**
 - a) When-Window-Closed at the form level:

```
EXIT_FORM;
```
 - b) Run the form and test that the form exits when you close the Customer Information window.
- 2) In the Orders form, create a When-Button-Pressed trigger on CONTROL.Show_Help_Button that uses the SHOW_VIEW built-in to display the CV_HELP.**

When-Button-Pressed on CONTROL.Show_Help_Button:

```
SHOW_VIEW('CV_HELP');
```

- 3) Create a When-Button-Pressed trigger on CONTROL.Hide_Help_Button that hides the CV_HELP. Use the HIDE_VIEW built-in to achieve this.**

When-Button-Pressed on CONTROL.Hide_Help_Button

```
HIDE_VIEW('CV_HELP');
```

- 4) Create a When-Button-Pressed trigger on CONTROL.Stock_Button that uses the GO_BLOCK built-in to display the INVENTORIES block, and EXECUTE_QUERY to automatically execute a query.**

When-Button-Pressed on CONTROL.Stock_Button:

```
GO_BLOCK('INVENTORIES');
EXECUTE_QUERY;
```

- 5) Write a form-level When-Window-Closed trigger to hide the WIN_INVENTORY window if the user attempts to close it, and to exit the form if the user attempts to close the WIN_ORDER window.**

Hint: Use the :SYSTEM.trigger_block system variable to determine what block the cursor is in when the trigger fires.

When-Window-Closed trigger on form:

```
IF :SYSTEM.trigger_block = 'INVENTORIES' THEN
    GO_BLOCK('orders');
ELSE
    EXIT_FORM;
END IF;
```

- 6) Save and compile the form. Click Run Form to run the form and test the changes.**

Practice 14-2: Using Triggers to Control Windows (continued)

- a) The stacked canvas, CV_HELP, is displayed only if the current item will not be obscured. Ensure, at least, that the first entered item in the form is one that will not be obscured by CV_HELP.
(You might decide to advertise Help only while the cursor is in certain items, or move the stacked canvas to a position that does not overlay enterable items. The CV_HELP canvas, of course, could also be shown in its own window, if appropriate.)
- b) Ensure that the Stock Information window closes when you click its X, and that the form exits when you click the X or the Exit button in the Order Information window.

Practices for Lesson 15

In the practice for this lesson, you use the Forms Builder integrated debugger to troubleshoot a problem with an application.

Practice 15-1: Using the Forms Debugger

In this practice, you run a form in debug mode, set a breakpoint, step through code, and look at variable values.

- 1) Open your CUSTGXX.FMB file. In this form, create a procedure that is called List_Of_Values. Import code from the pr15_1.txt file:**

```
PROCEDURE list_of_values(p_lov IN VARCHAR2, p_text IN
    VARCHAR2) IS
    v_lov BOOLEAN;
BEGIN
    v_lov := SHOW_LOV(p_lov);
    IF v_lov = TRUE THEN
        MESSAGE('You have just selected a(n) ' || p_text);
    ELSE
        MESSAGE('You have just cancelled the List of Values');
    END IF;
END;
```

- a) Select the Program Units node in the Object Navigator and click Create.
 - b) In the New Program Unit window, enter the name List_Of_Values, and click OK.
 - c) In the PL/SQL Editor, select all the text and press Delete to delete it.
 - d) From the menu, select File > Import PL/SQL Text.
 - e) In the Import window, click the pop-up list in the “Files of type:” field to change the File type to All Files (*.*).
 - f) Select pr15_1.txt and click Open.
 - g) In the PL/SQL Editor, click Compile PL/SQL code, the icon at the upper-left of the window.
- 2) Modify the When-Button-Pressed trigger of CONTROL.Account_Mgr_LOV_Button in order to call this procedure. Misspell the parameter to pass the LOV name.**

When-Button-Pressed on CONTROL.Account_Mgr_LOV_Button:
LIST_OF_VALUES ('ACCOUNT_MGR_LO', 'Account Manager');
- 3) Compile your form and click Run Form to run it. Execute a query. Press the LOV button for the Account Manager. Notice that the LOV does not display, and you receive a message that “You have just cancelled the List of Values.”**

No formal solution
- 4) Now click Run Form Debug. Set a breakpoint in your When-Button-Pressed trigger, and investigate the call stack. Try stepping through the code to monitor its progress. Look at the Variables panel to see the value of the parameters you passed to the procedure, and the value of the p_lov variable in the procedure. How does this information help you to figure out where the code was in error?**
 - a) Click Run Form Debug.

Practice 15-1: Using the Forms Debugger (continued)

- b) In Forms Builder, open the PL/SQL Editor for the When-Button-Pressed trigger on the CONTROL.Account_Mgr_LOV_Button.
- c) Double-click the gray area to the left of the code to create a breakpoint.
- d) In the running form, execute a query and then click Account_Mgr_LOV_Button. The debugger takes control of the form because the breakpoint is encountered. The running form may now appear as a blank applet.
- e) In Forms Builder, the Debug Console is displayed. If the Stack panel and the Variables panel are not shown, click the appropriate icons of the Debug Console window to display them.
- f) In the Forms Builder toolbar, click Step Into. This adds the List_Of_Values procedure to the Stack panel. In addition, some variables now appear in the Variables panel.
- g) Resize the Value column of the Variables panel so that you can see the value of the variables. You can see that the p_lov value is incorrect.
- h) Click Go in the Forms Builder toolbar to execute the remaining code. Control returns to the running form.
- i) Click Stop to dismiss the debugger.
- j) Exit the form and the browser, and then correct the code in the When-Button-Pressed trigger. You can then run the form again to retest it.

Practices for Lesson 16

In the practices for this lesson, you create some additional functionality for a radio group. You also code interaction with a JavaBean. Finally, you add some triggers that enable interaction with buttons.

Practice 16-1: Responding to a Value Change

In this practice, you write a trigger to check whether the customer's credit limit has been exceeded.

- 1) In the Customers form, write a trigger that fires when the credit limit changes. The trigger should display a message warning if a customer's outstanding credit orders (those with an order status between 4 and 9) exceed the new credit limit. You can import the pr16_1.txt file.**

- a) Create a When-Radio-Changed trigger on CUSTOMERS.Credit_Limit.
- b) With the PL/SQL Editor open, select File > Import PL/SQL Text.
- c) In the Import window, click the pop-up list in the "Files of type:" field to change the File type to All Files (*.*).
- d) Select pr16_1.txt and click Open.
- e) In the PL/SQL Editor, click Compile PL/SQL code, the icon at the upper-left of the window.
- f) When-Radio-Changed on CUSTOMERS.Credit_Limit:

```
DECLARE v_unpaid_orders NUMBER;
BEGIN
    SELECT SUM(nvl(unit_price,0)*nvl(quantity,0))
    INTO v_unpaid_orders
    FROM orders o, order_items i
    WHERE o.customer_id = :customers.customer_id
    AND o.order_id = i.order_id
    -- Unpaid credit orders have status between 4 and 9
    AND (o.order_status > 3 AND o.order_status < 10);
    IF v_unpaid_orders > :customers.credit_limit THEN
        MESSAGE('This customer''s current orders exceed
                the new credit limit');
    END IF;
END;
```

- 2) Click Run Form to run the form and test the functionality.**

Hint: Most customers who have outstanding credit orders exceed the credit limits, so you should receive the warning for most customers. Customer 120 has outstanding credit orders of less than \$500, so you should not receive a warning when changing this customer's credit limit.

No formal solution

Practice 16-2: Invoking a JavaBean

In this practice, you code a button to enable users to choose a canvas color for a form.

- 1) In the Customers form, begin to implement a JavaBean for the ColorPicker bean area on the CONTROL block that will enable a user to choose a color from a color picker.**

Create a button on the CV_CUSTOMER canvas to enable the user to change the canvas color using the ColorPicker bean.

Set the following properties on the button:

Label: Canvas Color Mouse Navigate: No

Keyboard Navigable: No Background color: gray

The button should call a procedure named PickColor, with the imported text from the pr16_2.txt file.

The bean will not function at this point, but you will write the code to instantiate it in Practice 20.

- a) Create a procedure under the Program Units node of the Object Navigator. Name the procedure PickColor. With the PL/SQL Editor open, delete the existing text and select File > Import PL/SQL Text. The code in pr16_2.txt is:

```
PROCEDURE PickColor(pvcTarget in VARCHAR2) IS
    vcOldColor VARCHAR2(12 char);
    vcNewColor VARCHAR2(12 char);
    hCanvas CANVAS := FIND_CANVAS('cv_customer');
    hColorPicker ITEM := FIND_ITEM('control.colorpicker');
    vnPos1 NUMBER;
    vnPos2 NUMBER;

BEGIN
    -- First get the current Color for this target
    vcOldColor :=
        get_canvas_property(hCanvas,background_color);
    vnPos1 := instr(vcOldColor,'g',1);
    vnPos2 := instr(vcOldColor,'b',vnPos1 + 1);
    vcOldColor := substr(vcOldColor,2,vnPos1 - 2) ||
        ' '||substr(vcOldColor,vnPos1+1,vnPos2 - vnPos1 -1) ||
        ' '||substr(vcOldColor,vnPos2 + 1,length(vcOldColor)-
    vnPos2);
    -- now display the picker with that color as an initial
    -- value
    vcNewColor :=
        FBean.Invoke_char(hColorPicker,1,'showColorPicker',
        '"Select color
for'||pvcTarget||'"',''||vcOldColor||"');
    -- finally if vcColor is not null reset the Canvas
    property
        if (vcNewColor is not null) then
            vnPos1 := instr(vcNewColor,' ',1);
            vnPos2 := instr(vcNewColor,' ',vnPos1 + 1);
            vcNewColor := 'r'||substr(vcNewColor,1,vnPos1 - 1)|||
                'g'||substr(vcNewColor,vnPos1+1,vnPos2 - vnPos1 -
            1) ||
                'b'||substr(vcNewColor,vnPos2 +
```

Practice 16-2: Invoking a JavaBean (continued)

```
1, length(vcNewColor) -  
    vnPos2);  
set_canvas_property(hCanvas, background_color, vcNewColor);  
end if;  
END;
```

- b) Create a button in the CONTROL block called Color_Button on the CV_CUSTOMER canvas, to the right of the boilerplate text, setting its properties as shown above. Define a When-Button-Pressed trigger for the button:

```
PickColor('Canvas');
```

- 2) Save and compile the form. You will not be able to test the Color button yet, because the bean does not function until you instantiate it in Practice 20.**

No formal solution

Practice 16-3: Setting Properties of an Image Item

In this practice, you create a toolbar button to display and hide product images.

- 1) In the Orders form CONTROL block, create a new button called Image_Button and position it on the toolbar. Set the Label property to Image Off. Set the navigation, width, height, and color properties like the other toolbar buttons.**
 - a) Display the Layout Editor. Make sure the Toolbar canvas and the CONTROL block are selected.
 - b) Select the Button tool, then create a button and place it on the toolbar.
 - c) Set Name to Image_Button.
 - d) Set the Keyboard Navigable property to No.
 - e) Set the Mouse Navigate property to No; set Width and Height to 50 and 16, respectively.
 - f) Set the Label property to Image Off.
 - g) Set the Background Color property to white.
- 2) Import the pr16_3.txt file into a trigger that fires when the Image_Button is clicked. The file contains code that determines the current value of the visible property of the Product_Image item. If the current value is True, the visible property toggles to False for both the Product_Image item and the Image Description item. Finally, the label changes on the Image_Button to reflect its next toggle state. However, if the visible property is currently False, the visible property toggles to True for both the Product_Image item and the Image_Description item.**
 - a) Create a When-Button-Pressed trigger on CONTROL.Image_Button.
 - b) With the PL/SQL Editor open, select File > Import PL/SQL Text.
 - c) In the Import window, click the pop-up list in the “Files of type:” field to change the File type to All Files (*.*).
 - d) Select pr16_3.txt and click Open.

When-Button-Pressed on Control.Image_Button:

```
IF GET_ITEM_PROPERTY('control.product_image',
    VISIBLE)='TRUE' THEN
    SET_ITEM_PROPERTY('control.product_image', VISIBLE,
        PROPERTY_FALSE);
    SET_ITEM_PROPERTY('order_items.image_description',
        VISIBLE, PROPERTY_FALSE);
    SET_ITEM_PROPERTY('control.image_button', LABEL,
        'Image On');
ELSE
    SET_ITEM_PROPERTY('control.product_image', VISIBLE,
        PROPERTY_TRUE);
    SET_ITEM_PROPERTY('order_items.image_description',
        VISIBLE, PROPERTY_TRUE);
```

Practice 16-3: Setting Properties of an Image Item (continued)

```
SET_ITEM_PROPERTY('control.image_button',LABEL,'Image  
Off');  
END IF;
```

- 3) Save and compile the form. Click Run Form to run the form and test the functionality. Note: The image will not display in the image item at this point; you will add code to populate the image item in Practice 20.

Practices for Lesson 17

In the practices for this lesson, you create some alerts. These include a general alert for questions and a specific alert that is customized for credit limit.

Practice 17-1: Using a Customized Alert

In this practice, you define an alert for the sole purpose of informing the operator that the customer's credit limit has been exceeded.

- 1) Create an alert in the Customers form called Credit_Limit_Alert with one OK button. The message should read “This customer’s current orders exceed the new credit limit”.**
 - a) Create an alert.
 - b) Set Name to Credit_Limit_Alert.
 - c) Set Title to Credit Limit.
 - d) Set Alert Style to Caution.
 - e) Set Button1 Label to OK.
 - f) Set Message to “This customer’s current orders exceed the new credit limit.”
 - g) Remove the labels for the other buttons.
- 2) Alter the When-Radio-Changed trigger on Credit_Limit to show the Credit_Limit_Alert instead of the message when a customer’s credit limit is exceeded.**

When-Radio-Changed on CUSTOMERS.Credit_Limit (changed or added lines are in bold):

```
DECLARE
  n NUMBER;
  v_unpaid_orders NUMBER;
BEGIN
  SELECT SUM(nvl(unit_price,0)*nvl(quantity,0))
  INTO v_unpaid_orders
  FROM orders o, order_items i
  WHERE o.customer_id = :customers.customer_id
  AND o.order_id = i.order_id
  -- Unpaid credit orders have status between 4 and 9
  AND (o.order_status > 3 AND o.order_status < 10);
  IF v_unpaid_orders > :customers.credit_limit THEN
    n := SHOW_ALERT('credit_limit_alert');
  END IF;
END;
```

- 3) Save and compile the form. Click Run Form to run the form and test the changes.**

Test as in Practice 16-1, step 2. The message should display in an alert rather than on the status line as it previously did.

Practice 17-2: Using a Generic Alert

In this practice, you define a generic alert and programmatically set its properties to ask the operator to confirm that the form should terminate.

- 1) Create a generic alert in the Orders form called Question_Alert that allows Yes and No replies.**
 - a) Create an alert.
 - b) Set Name to QUESTION_ALERT.
 - c) Set Title to Question.
 - d) Set Alert Style to Stop.
 - e) Set Button1 Label to Yes.
 - f) Set Button2 Label to No.
- 2) Alter the When-Button-Pressed trigger on CONTROL.Exit_Button to use the Question_Alert to ask the operator to confirm that the form should terminate. (You can import the text from pr17_1.txt.)**

When-Button-Pressed on CONTROL.Exit_Button:

```
SET_ALERT_PROPERTY('Question_Alert', ALERT_MESSAGE_TEXT,  
    'Do you really want to leave the form?');  
SET_ALERT_PROPERTY('Question_Alert', TITLE,  
    'Exit Confirmation');  
IF SHOW_ALERT('Question_Alert') = ALERT_BUTTON1 THEN  
    EXIT_FORM;  
END IF;
```

- 3) Save and compile the form. Click Run Form to run the form and test the changes.**

No formal solution

Practices for Lesson 18

In this practice, you create two query triggers to populate non-base-table items. You also change the default query interface to enable case-sensitive and exact match queries.

Practice 18-1: Populating Nonbase Table Items

In this practice, you use Post-Query triggers to populate nonbase table items.

- 1) In the Orders form, write a trigger that populates Customer_Name and the Sales_Rep_Name for every row fetched by a query on the ORDERS block. You can import the text from pr18_1.txt.**

Post-Query on the ORDERS block:

```
BEGIN
    IF :orders.customer_id IS NOT NULL THEN
        SELECT cust_first_name || ' ' || cust_last_name
        INTO :orders.customer_name
        FROM customers
        WHERE customer_id = :orders.customer_id;
    END IF;
    IF :orders.sales_rep_id IS NOT NULL THEN
        SELECT first_name || ' ' || last_name
        INTO :orders.sales_rep_name
        FROM employees
        WHERE employee_id = :orders.sales_rep_id;
    END IF;
END;
```

- 2) Write a trigger that populates the Description for every row fetched by a query on the ORDER_ITEMS block. You can import the text from pr18_2.txt.**

Post-Query on the ORDER_ITEMS block:

```
BEGIN
    IF :order_items.product_id IS NOT NULL THEN
        SELECT translate(product_name using char_cs)
        INTO :order_items.description
        FROM products
        WHERE product_id = :order_items.product_id;
    END IF;
END;
```

Practice 18-2: Using the ONETIME_WHERE Clause

In this practice, you use a ONETIME_WHERE clause to restrict a query only the first time it is executed.

- 1) Change the When-Button-Pressed trigger on the Stock_Button in the CONTROL block so that users will be able to execute a second query on the INVENTORIES block that is not restricted to the current Product_Id in the ORDER_ITEMS block. You can import the text from pr18_3.txt.**
 - a) In the Property Palette for Inventories.Product_Id, remove the value in the “Copy Value from Item” property.
 - b) When-Button-Pressed on Stock_Button:

```
SET_BLOCK_PROPERTY('inventories',ONETIME_WHERE,  
'product_id='||:order_items.product_id);  
GO_BLOCK('inventories');  
EXECUTE_QUERY;
```

Practice 18-3: Determining the Mode of the Running Form

In this practice, you determine whether a form is in Enter-Query mode before firing a trigger.

- 1) In the Orders form, ensure that the Exit_Button has no effect in Enter-Query mode.**

Set Fire in Enter-Query Mode property to No for the When-Button-Pressed trigger.

- 2) Create a button named Query_Button in the Control block and place it on the CV_Inventory canvas to the right of the Product Id (you may need to move the Product Id to make room for it.)**

Set its label to Enter Query, and set its other properties as you have for previous buttons. Write a trigger that places the form into Enter Query mode the first time the button is clicked and that executes the query the next time. Toggle the button label to Execute Query when the block is in query mode.

When-Button-Pressed on Query_Button:

```
IF :SYSTEM.mode = 'NORMAL' THEN
    SET_ITEM_PROPERTY('query_button',LABEL,'Execute Query');
    ENTER_QUERY;
ELSE
    SET_ITEM_PROPERTY('query_button',LABEL,'Enter Query');
    EXECUTE_QUERY;
END IF;
```

- 3) Click Run Form to run the form and test all the changes you have made to the Orders form in this set of practices.**

No formal solution

- 4) In the Customers form, ensure that the When-Radio-Changed trigger for the Credit_Limit item does not fire when in Enter-Query mode.**

- a) Open the Property Palette for the When-Radio-Changed trigger on the Credit_Limit item.**
- b) Set the “Fire in Enter-Query Mode” property to No.**

Practice 18-4: Enabling User Control of Queries

In this practice, you add functionality for the user to control certain aspects of a query.

- 1) Adjust the default query interface in the Customers form. Add a check box called CONTROL.Case_Sensitive to the form so that the user can specify whether or not a query for a customer name should be case-sensitive. Place the check box on the Name page of the TAB_CUSTOMER canvas. You can import the pr18_5.txt file into the When-Checkbox-Changed trigger. Set the Initial Value property to Y, and the Checked/Unchecked properties to Y and N. Set the Mouse Navigate property to No.

Note: If the background color looks different from that of the canvas, click in the Background Color property and click Inherit on the Property Palette toolbar.

- a) Open the Layout Editor and check that the canvas is TAB_CUSTOMER and the block is CONTROL. Click the Name tab.
- b) Using the Checkbox tool, place a check box on the canvas.
- c) Open the Property Palette for the new check box.
- d) Set the Name to CASE_SENSITIVE, the Initial Value to Y, “Value when Checked” to Y, “Value when Unchecked” to N, Keyboard Navigable to No, and Mouse Navigate to No.
- e) Set the Label to: Perform case sensitive query on name?
- f) In the Layout Editor, resize the check box so that the label fully displays.
- g) When-Checkbox-Changed trigger on the CONTROL. Case_Sensitive item (check box):

```
IF NVL(:control.case_sensitive, 'Y') = 'Y' THEN
    SET_ITEM_PROPERTY('customers.cust_first_name',
        CASE_INSENSITIVE_QUERY, PROPERTY_FALSE);
    SET_ITEM_PROPERTY('customers.cust_last_name',
        CASE_INSENSITIVE_QUERY, PROPERTY_FALSE);
ELSE
    SET_ITEM_PROPERTY('customers.cust_first_name',
        CASE_INSENSITIVE_QUERY, PROPERTY_TRUE);
    SET_ITEM_PROPERTY('customers.cust_last_name',
        CASE_INSENSITIVE_QUERY, PROPERTY_TRUE);
END IF;
```

- 2) Add a check box called CONTROL.Exact_Match to the form so that the user can specify whether or not a query condition for a customer name should exactly match the table value. (If a nonexact match is allowed, the search value can be part of the table value.) Set the label to: Exact match on query?

Set the Initial Value property to Y, and the Checked/Unchecked properties to Y and N. Set the Keyboard Navigable and Mouse Navigate properties to No. You can import the pr18_6.txt file into the When-Checkbox-Changed Trigger.

Note: If the background color looks different from that of the canvas, click in the Background Color property and click Inherit on the Property Palette toolbar.

- a) Create the check box as in Practice 18-6 and set its properties.

Practice 18-4: Enabling User Control of Queries (Continued)

- b) Pre-Query trigger on the CUSTOMERS block:

```
IF NVL( :control.exact_match, 'Y' ) = 'N' THEN
    IF :customers.cust_first_name IS NOT NULL THEN
        :customers.cust_first_name := '%' ||
            :customers.cust_first_name || '%';
    END IF;
    IF :customers.cust_last_name IS NOT NULL THEN
        :customers.cust_last_name := '%' ||
            :customers.cust_last_name || '%';
    END IF;
END IF;
```

- 3) Modify the properties of the CONTROL block so that the check boxes can be checked or unchecked at run time.

Hint: The CONTROL block contains a single new record.

Set the block property Insert Allowed to Yes.

- 4) Run the form to test the changes, being sure to also test that the When-Radio-Changed trigger for the Credit_Limit item does not fire when in Enter-Query mode.

No formal solution

Practices for Lesson 19

In the practices for this lesson, you implement validation in three different ways:

- By using an LOV for validation
- By coding validation triggers
- By implementing a PJC

Practice 19-1: Using an LOV for Validation

In this practice, you validate the Account_Mgr_Id with an LOV.

- 1) In the Customers form, cause the Account_Mgr_Lov to be displayed whenever the user enters an Account_Mgr_Id that does not exist in the database.**

Set the Validate from List property to Yes for the Account_Mgr_Id item in the CUSTOMERS block.

- 2) Save and compile the form. Click Run Form to run the form and test the functionality.**

No formal solution

Practice 19-2: Using Validation Triggers

In this practice, you use triggers for validation as well as to return data to nonbase table items.

- 1) In the Orders form, write a validation trigger to check that if the Order_Mode is online, the Order_Status indicates a CREDIT order (values between 4 and 10). You can import the text from pr19_1.txt.**

When-Validate-Record on the ORDERS block (shows error if an online order is not a credit order):

```
IF :orders.order_mode = 'online' and
   :orders.order_status not between 4 and 10 THEN
   MESSAGE('Online orders must be CREDIT orders');
   RAISE FORM_TRIGGER_FAILURE;
END IF;
```

- 2) In the Orders form, create triggers to write the correct values to the Customer_Name whenever validation occurs on Customer_Id, and to the Sales_Rep_Name when validation occurs on Sales_Rep_Id. Fail the triggers if the data is not found. You can import text from pr19_2.txt and pr19_3.txt.**

- a) When-Validate-Item on ORDERS.Customer_Id:**

```
SELECT cust_first_name || ' ' || cust_last_name
  INTO :orders.customer_name
    FROM customers
   WHERE customer_id = :orders.customer_id;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    MESSAGE('Invalid customer id');
    RAISE FORM_TRIGGER_FAILURE;
```

- b) When-Validate-Item on ORDERS.Sales_Rep_Id:**

```
SELECT first_name || ' ' || last_name
  INTO :orders.sales_rep_name
    FROM employees
   WHERE employee_id = :orders.sales_rep_id
     AND job_id = 'SA_REP';
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    MESSAGE('Invalid sales rep id');
    RAISE FORM_TRIGGER_FAILURE;
```

- 3) Create another validation trigger on ORDER_ITEMS.Product_Id to derive the name of the product and suggested wholesale price, and write them to the Description item and the Price item. Fail the trigger and display a message if the product is not found. You can import the text from pr19_4.txt.**

When-Validate-Item on ORDER_ITEMS.Product_Id:

```
SELECT product_name, list_price
  INTO :order_items.description,
       :order_items.unit_price
```

Practice 19-2: Using Validation Triggers (continued)

```
FROM products
WHERE product_id = :order_items.product_id;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        MESSAGE('Invalid product id');
        RAISE FORM_TRIGGER_FAILURE;
```

- 4) Save and compile the form. Click Run Form to run the form and test the changes.

No formal solution

Practice 19-3: Using a Pluggable Java Component

In this practice, you begin to implement a PJC to validate input on the client.

- 1) Perform client-side validation on the ORDER_ITEMS.Quantity item using a Pluggable Java Component to filter the keystrokes and allow only numeric values. The full path to the PJC class is `oracle.forms.demos.KeyFilter` (this is case-sensitive), to be used as the Implementation Class for the item. You will set the filter for the item in the next practice, so the validation is not yet functional.**

Note: Do not test the validation on the Quantity item yet because it will not function until after you set the filter on the item in Practice 20.

- a) Open the Property Palette for the ORDER_ITEMS.Quantity item.
- b) Set the Implementation Class property to:
`oracle.forms.demos.KeyFilter`

Practices for Lesson 20

In the practices for this lesson, you use navigation triggers to perform the following tasks:

- Register the implementation class for a bean area item at form startup.
- Set properties on a PJC at form startup.
- Execute a query at form startup.
- Populate an image item when navigating to each record of a block.

Practice 20-1: Initializing a Form at Startup

In this practice, you perform initialization tasks when a form starts up.

- 1) In the Customers form, register the ColorPicker bean (making its methods available to Forms) when the form first opens, and also execute a query on the CUSTOMERS block. You can import the code from `pr20_1.txt`.**

When-New-Form-Instance trigger on the Customers form:

```
FBean.Register_Bean('control.colorpicker',1,  
                     'oracle.forms.demos.beans.ColorPicker');  
GO_BLOCK('customers');  
EXECUTE_QUERY;
```

- 2) Save, compile, and click Run Form to run the form and test the Canvas Color button. You should be able to invoke the ColorPicker bean from the Canvas Color button now that the bean has been registered at form startup.**

No formal solution

- 3) When the Orders form first opens, set a filter on the ORDER_ITEMS.Quantity Pluggable Java Component, and execute a query. Note that you must set the filter for each visible row in the block. You can import the code for the trigger from `pr20_2.txt`.**

When-New-Form-Instance at the form level:

```
SET_CUSTOM_PROPERTY('order_items.quantity',1,'FILTER_TYPE',  
'NUMERIC');  
SET_CUSTOM_PROPERTY('order_items.quantity',2,'FILTER_TYPE',  
'NUMERIC');  
SET_CUSTOM_PROPERTY('order_items.quantity',3,'FILTER_TYPE',  
'NUMERIC');  
SET_CUSTOM_PROPERTY('order_items.quantity',4,'FILTER_TYPE',  
'NUMERIC');  
SET_CUSTOM_PROPERTY('order_items.quantity',5,'FILTER_TYPE',  
'NUMERIC');  
SET_CUSTOM_PROPERTY('order_items.quantity',6,'FILTER_TYPE',  
'NUMERIC');  
GO_BLOCK('orders');  
EXECUTE_QUERY;
```

Practice 20-2: Initializing a Record

In this practice, you initialize a record when navigation to it occurs.

- 1) In the Orders form, write a trigger that fires as the cursor arrives in each record of the ORDER_ITEMS block to populate the Product_Image item with a picture of the product, if one exists. First, create a procedure called get_image to populate the image, and then call that procedure from the appropriate trigger. You can import the code for the procedure from pr20_3.txt.**

- a) get_image procedure:

```
PROCEDURE get_image IS
    filename VARCHAR2(250);
BEGIN
    filename := to_char(:order_items.product_id) || '.jpg';
    READ_IMAGE_FILE(filename, 'jpeg', 'control.product_image');
END;
```

- b) When-New-Record-Instance on the ORDER_ITEMS block:

```
get_image;
```

- 2) Define the same trigger type and code on the ORDERS block.**

When-New-Record-Instance on the ORDERS block:

```
get_image;
```

- 3) Is there another trigger where you might also want to place this code?**

When-Validate-Item on ORDER_ITEMS.Product_Id is a candidate for the code.

- 4) Save and compile the form. Click Run Form to run the form and test the changes.**

Try entering alphabetic characters into Order_Items.Quantity.

- 5) Notice that you receive an error if the image file does not exist. Code a trigger to gracefully handle the error by populating the image item with a default image called blank.jpg. You can import the code from pr20_4.txt.**

On-Error trigger on the Orders form:

```
IF ERROR_CODE = 47109 THEN
    READ_IMAGE_FILE('blank.jpg', 'jpeg',
                    'control.product_image');
ELSE
    MESSAGE(ERROR_TYPE || '-' || TO_CHAR(ERROR_CODE) ||
           ': ' || ERROR_TEXT);
END IF;
```

- 6) The image item has a lot of blank space when the image does not take up the entire area. To make it look better, set its Background Color of both the CONTROL.Product_Image item and the CV_ORDER canvas to the same value, such as r0g75b75. Set the Bevel for the Product_Image item to None.**

Practice 20-2: Initializing a Record (continued)

With both the CONTROL.Product_Image item and the CV_ORDER canvas selected in the Object Navigator, open the Property Palette. Set the Background Color for both objects to r0g75b75. With just the Product_Image item selected, open the Property Palette and set Bevel to None.

- 7) **Click Run Form to run your form again and test the changes.**

No formal solution

Practices for Lesson 21

In this practice, you add transactional triggers to the Orders form to automatically provide sequence numbers to records at save time. You also customize commit messages and the login screen in the Customers form.

Practice 21-1: Using Transactional Triggers to Populate Primary Keys

In this practice, you use Pre-Insert triggers to populate primary keys. You use a sequence to populate ORDERS.Order_Id. You then assign a value to ORDER_ITEMS.Line_Item_Id that is one greater than the largest existing line item ID for that order.

- 1) In the Orders form, write a transactional trigger on the ORDERS block that populates ORDERS.Order_Id with the next value from the ORDERS_SEQ sequence. You can import the code from pr21_1.txt.**

Pre-Insert on the ORDERS block:

```
SELECT orders_seq.NEXTVAL INTO :orders.order_id
   FROM sys.dual;
EXCEPTION
  WHEN OTHERS THEN
    MESSAGE('Unable to assign order id');
    RAISE FORM_TRIGGER_FAILURE;
```

- 2) In the ORDERS block, set the Enabled property for the Order_Id item to No. Set the Required property for the Order_Id item to No. To ensure that the data remains visible, set the Background Property to gray.**

In the Property Palette, set the Enabled and Required properties to No for ORDERS.Order_Id. Set Background Color to gray.

- 3) Create a similar trigger on the ORDER_ITEMS block that assigns the Line_Item_Id when a new record is saved. Do not use a sequence, but determine the next line order ID for that order programmatically. Set the properties for the item as you did on ORDERS.Order_Id. You can import the code from pr21_2.txt.**

Pre-Insert on the ORDER_ITEMS block:

```
SELECT NVL(MAX(line_item_id),0) + 1
  INTO :order_items.line_item_id
   FROM order_items
 WHERE :order_items.order_id = order_id;
EXCEPTION
  WHEN OTHERS THEN
    MESSAGE('Unable to assign line item id');
    RAISE FORM_TRIGGER_FAILURE;
```

Set the Required and Enabled properties to No and the Background Color to gray for ORDER_ITEMS.Line_Item_Id.

- 4) Save and compile the form. Click Run Form to run the form and test the changes.**
 - a) Insert a new order. Use a customer ID of 104 and a sales rep ID of 150.
 - b) Save the record to ensure that the order ID and line item IDs are properly assigned.

Practice 21-2: Using Transactional Triggers to Log Transaction Information

In this practice, you use transactional triggers to compile a specialized commit message that you display to the user after the transaction is complete.

- 1) Open the Customers form module. Create three global variables called `GLOBAL.insert`, `GLOBAL.update`, and `GLOBAL.delete`. These variables indicate the number of inserts, updates, and deletes, respectively. You need to write Post-Insert, Post-Update, and Post-Delete triggers to initialize and increment the value of each global variable. You may import the code from `pr21_3.txt`, `pr21_4.txt`, and `pr21_5.txt`.**

- a) Post-Insert at the form level:**

```
DEFAULT_VALUE('0', 'GLOBAL.insert');
:GLOBAL.insert := TO_CHAR(TO_NUMBER(:GLOBAL.insert)+1);
```

- b) Post-Update at the form level:**

```
DEFAULT_VALUE('0', 'GLOBAL.update');
:GLOBAL.update := TO_CHAR(TO_NUMBER(:GLOBAL.update)+1);
```

- c) Post-Delete at the form level:**

```
DEFAULT_VALUE('0', 'GLOBAL.delete');
:GLOBAL.delete := TO_CHAR(TO_NUMBER(:GLOBAL.delete)+1);
```

- 2) Create a procedure called `HANDLE_MESSAGE`. Import the `pr21_6.txt` file. This procedure receives two arguments. The first one is a message number, and the second is a message line to be displayed. This procedure uses the three global variables to display a customized commit message and then erases the global variables.**

```
PROCEDURE handle_message( message_number IN NUMBER,
message_line IN VARCHAR2 ) IS
BEGIN
    IF message_number IN ( 40400, 40406, 40407 ) THEN
        DEFAULT_VALUE( '0', 'GLOBAL.insert' );
        DEFAULT_VALUE( '0', 'GLOBAL.update' );
        DEFAULT_VALUE( '0', 'GLOBAL.delete' );
        MESSAGE('Save Ok: ' ||
:GLOBAL.insert || ' records inserted, ' ||
:GLOBAL.update || ' records updated, ' ||
:GLOBAL.delete || ' records deleted !!!! ');
        ERASE('GLOBAL.insert');
        ERASE('GLOBAL.update');
        ERASE('GLOBAL.delete');
    ELSE
        MESSAGE( message_line );
    END IF;
END ;
```

Practice 21-2: Using Transactional Triggers to Log Transaction Information (continued)

- 3) Call the procedure when an error occurs. Pass the error code and an error message to be displayed. You can import the code from `pr21_7.txt`.
Call the procedure when a message occurs. Pass the message code and a message to be displayed. You can import the code from `pr21_8.txt`.

- a) On-Error at the form level:

```
handle_message( error_code, 'ERROR: ' || ERROR_TYPE || '-'  
|| TO_CHAR(ERROR_CODE) || ': ' || ERROR_TEXT );
```

- b) On-Message at the form level:

```
handle_message( message_code, MESSAGE_TYPE || '-' ||  
TO_CHAR(MESSAGE_CODE) || ': ' || MESSAGE_TEXT );
```

Practice 21-3: Controlling Login Behavior

In this practice, you use a trigger to limit the number of times a user can attempt to log in to the application.

- 1) In the Customers form, write an On-Logon trigger to control the number of connection tries. Use the LOGON_SCREEN built-in to simulate the default login screen and LOGON to connect to the database. You can import the pr21_9.txt file.**

On-Logon at the form level:

```
DECLARE
    connected BOOLEAN := FALSE;
    tries NUMBER := 3;
    un VARCHAR2(30);
    pw VARCHAR2(30);
    cs VARCHAR2(30);
BEGIN
    SET_APPLICATION_PROPERTY(CURSOR_STYLE, 'DEFAULT');
    WHILE connected = FALSE and tries > 0 LOOP
        LOGON_SCREEN;
        un := GET_APPLICATION_PROPERTY(USERNAME);
        pw := GET_APPLICATION_PROPERTY(PASSWORD);
        cs := GET_APPLICATION_PROPERTY
            (CONNECT_STRING);
        LOGON(un, pw || '@' || cs, FALSE);
        IF FORM_SUCCESS THEN
            connected := TRUE;
        END IF;
        tries := tries - 1;
    END LOOP;
    IF NOT CONNECTED THEN
        MESSAGE('Too many tries!');
        RAISE FORM_TRIGGER_FAILURE;
    END IF;
END;
```

- 2) Click Run Form to run the form and test all of the changes. Note that you cannot delete customers that have orders because of the foreign key relationship. Some of the customers without orders are 110 through 115.**

No formal solution

Note: The login screen is necessary when running the form by entering a URL in the browser. You should test its functionality here, but because logging in again is unnecessary when running from within Forms Builder (it uses the Forms Builder connection,) in the future you can just click Cancel to avoid having to enter login credentials.

Practices for Lesson 22

In this practice, you use properties and variables in the Orders form to provide flexible use of its code. You also make use of object IDs.

Practice 22-1: Using Object IDs

In this practice, you use built-ins to look up object IDs, and then reference objects in code by their IDs instead of their names.

- 1) In the Orders form, alter the code called by the triggers that populate the Product_Image item when the image item is displayed.**

Add a test in the code to check Product_Image. Perform the trigger actions only if the image is currently displayed. Use the GET_ITEM_PROPERTY built-in function. The code is contained in pr22_1.txt.

The `get_image` procedure is called by When-New-Record-Instance on the ORDERS and ORDER_ITEMS blocks. Modify the procedure under the Program Units node as follows (note the change in the call to `READ_IMAGE_FILE` denoted by italicized plain text):

```
PROCEDURE get_image IS
    product_image_id ITEM :=
        FIND_ITEM('control.product_image');
    image_button_id ITEM:=
        FIND_ITEM('control.image_button');
    filename VARCHAR2(250);
BEGIN
    IF GET_ITEM_PROPERTY(product_image_id,VISIBLE)='TRUE'
    THEN
        filename := to_char(:order_items.product_id) ||'.jpg';
        READ_IMAGE_FILE(filename,'jpeg',product_image_id);
    END IF;
END;
```

- 2) Alter the When-Button-Pressed trigger on Image_Button so that object IDs are used.**

Use a FIND_object function to obtain the IDs of each item referenced by the trigger. Declare variables for these IDs, and use them in each item reference in the trigger. The code is contained in pr22_2.txt.

When-Button-Pressed on CONTROL.image_button:

```
DECLARE
    product_image_id ITEM :=
        FIND_ITEM('control.product_image');
    image_desc_id ITEM :=
        FIND_ITEM('order_items.image_description');
    image_button_id ITEM :=
        FIND_ITEM('control.image_button');
BEGIN
    IF GET_ITEM_PROPERTY(product_image_id, VISIBLE)='TRUE'
    THEN
        SET_ITEM_PROPERTY(product_image_id, VISIBLE,
                          PROPERTY_FALSE);
        SET_ITEM_PROPERTY(image_desc_id,VISIBLE,
                          PROPERTY_FALSE);
        SET_ITEM_PROPERTY(image_button_id,LABEL,'Image On');
    ELSE
        SET_ITEM_PROPERTY(product_image_id, VISIBLE,
```

Practice 22-1: Using Object IDs (continued)

```
        PROPERTY_TRUE) ;
    SET_ITEM_PROPERTY(image_desc_id,VISIBLE,PROPERTY_TRUE) ;
    SET_ITEM_PROPERTY(image_button_id,LABEL,'Image Off') ;
END IF;
END;
```

Practice 22-2: Writing Generic Code by Using System Variables

In this practice, you use system variables to write generic code that you can use with any form.

- 1) In the Orders form, create a button called Blocks_Button in the CONTROL block and place it on the Toolbar canvas. Label the button Show Blocks. Set its navigation and color properties to be the same as the other toolbar buttons.**
The code for the button should print a message showing what block the user is currently in. It should keep track of the block and item where the cursor was located when the trigger was invoked (`:SYSTEM.cursor_block` and `:SYSTEM.CURSOR_ITEM`). It should then loop through the remaining navigable blocks of the form and print a message giving the names (`SYSTEM.current_block`) of all the navigable blocks in the form. Finally, it should navigate back to the block and item where the cursor was located when the trigger began to fire. Be sure to set the Mouse Navigate property of the button to No. You may import the code for the trigger from `pr22_3.txt`.

- a) Create a button on the TOOLBAR canvas and set its properties:

Name: Blocks_Button	Label: Show Blocks
Mouse Navigate: No	Keyboard Navigable: No
Height, Width: 16, 60	Background Color: white

- b) When-Button-Pressed on Blocks_Button:

```
DECLARE
    vc_startblk VARCHAR2(30) := :SYSTEM.cursor_block;
    vc_startitm VARCHAR2(30) := :SYSTEM.cursor_item;
    vc_otherblks VARCHAR2(80) := NULL;
BEGIN
    MESSAGE('You are in the ' || vc_startblk || ' block.');
    NEXT_BLOCK;
    WHILE :SYSTEM.current_block != vc_startblk LOOP
        vc_otherblks := vc_otherblks || ' ' ||
                        :SYSTEM.current_block;
        NEXT_BLOCK;
    END LOOP;
    message('Other block(s) in the form:' || vc_otherblks);
    GO_BLOCK(vc_startblk);
    GO_ITEM(vc_startitm);
END;
```

- 2) Save, compile, and run the form to test these features.**

Ensure that the images are still displayed correctly and that the Show Blocks functionality works.

- 3) The trigger code above is generic, so it will work with any form. Implement the same functionality for the Customers form.**

- a) Open the Customers form and define a similar Blocks_Button, labeled Show Blocks, in the CONTROL block, and place it just under the Color button on the CV_CUSTOMER canvas.

Practice 22-2: Writing Generic Code by Using System Variables (continued)

- b) Change the Background Color to gray.
 - c) In the Object Navigator, drag the When-Button-Pressed trigger you created for the Blocks_Button of the Orders form to the Triggers node under Blocks_Button of the Customers form. Choose to copy rather than subclass.
- 4) Run the Customers form to test the button.**
- No formal solution. The code should print messages about the blocks in the CUSTOMERS form, with no code changes.

Practices for Lesson 23

In the practices for this lesson, you use an object group and an object library to copy Forms Builder objects from one form to another. You also create a property class and use it to set multiple properties for several objects. You set SmartClasses in the object library and use these classes in the form module.

Practice 23-1: Using an Object Group

In this practice, you create an object group and use it in a new form module.

- 1) In the Orders form, create an object group, called Stock_Objects, consisting of the INVENTORIES block, CV_INVENTORY canvas, and WIN_INVENTORY window. Save the form.**
 - a) Select the Object Groups node and click the Create icon. Rename the group STOCK_OBJECTS.
 - b) Drag the INVENTORIES block, CV_INVENTORY canvas, and WIN_INVENTORY window to the Object Group Children node.
 - c) Save the form.
- 2) Create a new form module and copy the Stock_Objects object group into it.**
 - a) Select the Forms node and click the Create icon.
 - b) Drag Stock_Objects from the ORDERS form to your new module under the Object Groups node. Select the Copy option. Notice that this creates a data block, a canvas, and a window in your new form.

Practice 23-2: Using a Property Class

In this practice, you create a property class and apply it to different types of objects.

- 1) In the new form module, create a property class called ClassA. Include the following properties and settings:**

Font Name:	Arial
Format Mask:	99,999
Font Size:	8
Justification:	Right
Delete Allowed:	No
Background Color:	DarkRed
Foreground Color:	Gray

- a) Select the Property Classes node and click the Create icon.
 - b) In the Property Palette for the property class, set the Name to CLASSA.
 - c) Add the properties listed by clicking the Add Property icon and selecting from the list displayed.
 - d) Set the properties to the values listed above.
- 2) Apply ClassA to CV_INVENTORY and the Quantity_on_Hand item.**
 - a) In the Property Palette for each of the objects, select the Subclass Information property, and click More.
 - b) In the Subclass Information dialog box, select the Property Class option and set the Property Class Name list item to ClassA.
 - c) You can examine the Property Palettes of the two objects to see which properties of each were inherited from the property class (denoted by the bent arrow icons.)
- 3) Name the form module Stock, and save it as STOCKXX.fmb.**
No formal solution
- 4) Make the canvas background color a variant property by setting it to gray.**
 - a) In the Property Palette for the canvas, change the Background Color to gray.
 - b) Notice the red X that appears over the icon.
 - c) Notice that the canvas is now the same color as the foreground color, causing some of the text labels to not be visible.
- 5) Reset the canvas background color to the value defined in the property class.**
 - a) Select the Background Color property in the Property Palette and click Inherit.
 - b) Notice that the red X disappears from the icon for the property.
- 6) Save, compile, and run the form to test it.**
No formal solution

Practice 23-3: Using an Object Library

In this practice, you create an object library and use its objects in a new form. You also mark some of the objects as smart classes to facilitate reuse.

1) Create an object library and name it `Summit`.

Create two tabs in the object library called Personal and Corporate.

Add the CONTROL block, the Toolbar, and the Question_Alert from the Orders form to the Personal tab of the object library.

Save the object library as `summitXX.olb`.

- a) Select the Object Libraries node in the Object Navigator, and click Create.
- b) Rename this object library SUMMIT.
- c) Select the Library Tabs node in the Object Navigator.
- d) Click Create twice to create two tabs.
- e) Set the Name and Label properties for the first tab as Personal and for the second tab as Corporate.
- f) Open the object library by double-clicking its icon in the Object Navigator.
- g) From the Orders form, drag the CONTROL block, the Toolbar, and the Question_Alert to the Personal tab of the Object Library.
- h) Save the Summit object library as `summitXX.olb`.

2) Use the objects in the object library in a new form by performing the following steps:

- a) **Create a new form, and create a data block based on the DEPARTMENTS table, including all columns. Use the Form layout style. Name the form Departments.**
- b) **Drag the Toolbar canvas, CONTROL block, and Question_Alert from the object library to appropriate nodes in the new form, and select to subclass the objects. (For proper behavior, the DEPARTMENTS block must precede the CONTROL block in the Object Navigator.)**
- c) **Some items are not applicable to this form, but you cannot delete subclassed objects, so set the Canvas property for the following items to NULL: Image_Button, Stock_Button, Show_Help_Button, Product_Lov_Button, Hide_Help_Button, and Product_Image, Total.**
- d) **The code of some of the triggers does not apply to this form. Set the code for the When-Button-Pressed triggers for the above buttons to: NULL;**
- e) **For the Total item, set the Calculation Mode and Summary Function properties to None, and set the Summarized Block property to NULL.**
- f) **For the Query_Button, set its Canvas to the canvas that the data block is displayed on, and then adjust its position on the canvas.**
- g) **Use Toolbar as the Horizontal Toolbar canvas for this form.**

Practice 23-3: Using an Object Library (continued)

- h) Set the Window property to WINDOW1 for the Toolbar canvas.**
- i) Set the Horizontal Toolbar Canvas property to TOOLBAR for the window.**

Follow the practice steps; for a solution, see the 23dept.fmb file in the \soln directory.

3) Save this form as DEPTGXX, compile, and run the form to test it.

No formal solution

Notice that the color of the Exit button is white; you will change this shortly in the Object Library to see how it affects subclassed objects.

4) Change the Exit button of the Object Library's CONTROL block to have a gray background. Run the Departments form again to see that the Exit button is now gray.

- a) Create a new form module and drag the CONTROL block into it from the Object Library. Use the Copy option.**
- b) In the form, change the Background Color of the Exit button to gray.**
- c) Drag the CONTROL block back into the Object Library. Answer Yes to the Alert: "An object with this name already exists. Replace it with the new object?"**
- d) Run the Departments form again. You should see the Exit button is now gray, rather than white as it was before.**

5) Create two buttons and a date item as SmartClasses in the object library by performing the following steps:

- a) In the new form, create two sample buttons in the Control block, one for wide buttons and one for medium buttons, by means of width.**
- b) Name the buttons Wide_Btn and Medium_Btn.**
- c) Set their heights to 16 and their Widths to 60 and 40, respectively.**
- d) Set the Keyboard Navigable and Mouse Navigate properties to No.**
- e) Create a sample nonbase table date item named Date_Field.**
- f) Set the width and the format mask to your preferred standard (be sure the width is enough to display the date in the specified format.) For example, you could set Width to 100 and Format Mask to: fmMonth" "dd", "yyyy**
- g) Drag these three items to the Corporate tab of your object library. Mark these items as SmartClasses by selecting each one in the object library and selecting Edit > SmartClass from the menu.**

6) Create another new form and name it Employees. Create a new data block in the form based on the Employees table, using the Form layout and including all columns. Apply the SmartClasses in your form. Place the Toolbar canvas in the new form, using the form's existing window, and display the buttons on the toolbar. Set the Toolbar canvas to be the horizontal toolbar for the form and the window.

Practice 23-3: Using an Object Library (continued)

- a) In the Employees form, right-click Hire_Date and select SmartClasses > Date_Field.
- b) Create two buttons and set their SmartClass to Wide_Button and Medium_Button.
- c) Subclass Toolbar from the Personal tab of the object library to the form, and change its Window property to this form's window.
- d) Set the form and window properties to use the Toolbar canvas as the horizontal toolbar.
- e) Change the buttons' canvas to the toolbar and position them appropriately.
- f) Run the form and execute a query.
- g) Ensure that the hire date displays in your defined format mask and that the buttons show the correct display properties. See the 23Employees.fmb file in the \soln directory for an example.
- h) You can delete the new unnamed form module if you wish, because you will not be using it. You created it only to use in editing the Object Library.

Practice 23-4: Reusing PL/SQL Code

In this practice, you move common code routines into PL/SQL program units and a PL/SQL library.

- 1) In the Orders form, note the similarity of the code in the Post-Query trigger of the ORDERS block and in the When-Validate-Item triggers of the Orders.Customer_Id and Orders.Sales_Rep_Id items. Move the similar code to PL/SQL program units and call the program units from the triggers.
 - a) In the PL/SQL editor, open the Post-Query trigger for the Orders block. Copy the code of the first IF statement.
 - b) In the Object Navigator, select the Program Units node and click Create.
 - c) Enter the procedure name get_customer_name and click OK.
 - d) Paste the copied code into the body of the program unit.
 - e) In the PL/SQL editor, open the When-Validate-Item trigger of ORDERS.Customer_Id.
 - f) Copy the exception clause into the new program unit and click Compile. The code should be as follows:

```
PROCEDURE get_customer_name IS
BEGIN
    IF :orders.customer_id IS NOT NULL THEN
        SELECT cust_first_name || ' ' || cust_last_name
        INTO :orders.customer_name
        FROM customers
        WHERE customer_id = :orders.customer_id;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        MESSAGE('Invalid customer id');
        RAISE FORM_TRIGGER_FAILURE;
END;
```

- g) In the PL/SQL editor, open the Post-Query trigger for the Orders block. Copy the code of the second IF statement.
- h) In the Object Navigator, select the Program Units node and click Create.
- i) Enter the procedure name get_sales_rep_name and click OK.
- j) Paste the copied code into the body of the program unit.
- k) In the PL/SQL editor, open the When-Validate-Item trigger for the ORDERS.Sales_Rep_Id item.
- l) Copy the exception clause into the new program unit and click Compile. The code should be as follows:

```
PROCEDURE get_sales_rep_name IS
BEGIN
    IF :orders.sales_rep_id IS NOT NULL THEN
        SELECT first_name || ' ' || last_name
```

Practice 23-4: Reusing PL/SQL Code (continued)

```
        INTO :orders.sales_rep_name
        FROM employees
       WHERE employee_id = :orders.sales_rep_id
         AND job_id = 'SA_REP';
      END IF;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      MESSAGE('Invalid sales rep id');
      RAISE FORM_TRIGGER_FAILURE;
END;
```

- m) In the When-Validate-Item trigger for the ORDERS.Customer_Id item, replace the existing code with:
`get_customer_name;`
- n) In the When-Validate-Item trigger for the ORDERS.Sales_Rep_Id item, replace the existing code with:
`get_sales_rep_name;`
- o) In the Post-Query trigger of the ORDERS block, replace the existing code with:
`get_customer_name;`
`get_sales_rep_name;`
- 2) **The When-New-Form-Instance triggers of most forms use code that navigates to the first block in the form and then executes a trigger. Create a new PL/SQL library named `Summit_Lib`.**

Move the code to a library procedure named `init_query`, attach the library to the Orders and Customers forms, and replace the code in the When-New-Form-Instance triggers with a call to that procedure. You can also change the When-Button-Pressed trigger on the Stock button of the Orders form. You can import the code for the procedure from `pr23_1.txt`.

Hint: Pass the block name as an argument to the procedure.

- a) In the Object Navigator, select the PL/SQL Libraries node and click Create.
- b) In the new library, create a procedure named `init_query`.
- c) Add a VARCHAR2 argument named `block`.
- d) Code the procedure to navigate to the specified block and execute a query. The code should read as follows:

```
PROCEDURE init_query (block VARCHAR2) IS
BEGIN
  go_block(block);
  execute_query;
END;
```

- e) Save the new library as `Summit_Lib`.
- f) Attach the library to the Orders form:
 - i) Select the Attached Libraries node and click Create.
 - ii) Browse to select the `Summit_Lib.dll` file and click Attach.

Practice 23-4: Reusing PL/SQL Code (continued)

- iii) In the alert asking if you want to remove the path, click Yes (the library should be in the FORMS_PATH of the Forms Services environment for this to work.)
- g) In the When-New-Form-Instance trigger of the Orders form, replace the two lines of code that navigate to the Orders block and execute a query with a call to the library procedure, passing the name of the block as an argument (see the last line of code):

```
-- set filter for each visible row in the Order_Items block
SET_CUSTOM_PROPERTY('order_items.quantity',1,'FILTER_TYPE',
'NUMERIC');
SET_CUSTOM_PROPERTY('order_items.quantity',2,'FILTER_TYPE',
'NUMERIC');
SET_CUSTOM_PROPERTY('order_items.quantity',3,'FILTER_TYPE',
'NUMERIC');
SET_CUSTOM_PROPERTY('order_items.quantity',4,'FILTER_TYPE',
'NUMERIC');
SET_CUSTOM_PROPERTY('order_items.quantity',5,'FILTER_TYPE',
'NUMERIC');
SET_CUSTOM_PROPERTY('order_items.quantity',6,'FILTER_TYPE',
'NUMERIC');
init_query('orders');
```

- h) Be sure that the trigger compiles correctly.
- i) Change the When-Button-Pressed trigger on Control.Stock_Button to:
- ```
SET_BLOCK_PROPERTY('inventories',ONETIME_WHERE,
'product_id=' || :order_items.product_id);
init_query('inventories');
```
- j) Attach the library to the Customers form and make a similar replacement in the When-New\_Form-Instance trigger:
- ```
FBean.Register_Bean('control.colorpicker',1,
'oracle.forms.demos.beans.ColorPicker');
init_query('customers');
```
- k) Run the Orders form and ensure that the query executes correctly upon form startup and that the Stock button displays the inventory correctly. Make sure that the customer name and sales rep name are populated correctly.
- l) Run the Customers form and ensure that the query executes correctly upon form startup.

Practices for Lesson 24

In the practices for this lesson, you use WebUtil to perform various functions on the client machine.

Practice 24-1: Integrating WebUtil into a Form

In this practice, you load the WEBUTIL object group into the Orders form.

- 1) In the Orders form, integrate the WebUtil objects and library.**
 - a) Open the webutil.olb object library from the <ORACLE_HOME>\forms directory.
 - b) In the Object Navigator, double-click the icon for the WEBUTIL object library to open it.
 - c) Select the WEBUTIL object group from the object library and drag it to the Object Groups node of the Orders form. Choose to Subclass, rather than Copy, the objects.
 - d) In the Object Navigator, drag the WEBUTIL block to the last position under the Blocks node.
 - e) For the steps to attach a PL/SQL library to a form see Practice 23-4, step 2(f). Attach the WEBUTIL library (the webutil.PLL file) from the <ORACLE_HOME>\forms directory, choosing to remove the hard-coded path.

Practice 24-2: Reading a Client Image File

In this practice, you load a client image into your form.

- 1) Change the Exit button in the CONTROL block. Rename it New_Image_Btn. Relabel it New Image. Delete the current code for the button and write code to enable the user to choose a new JPEG image to display in the Product_Image item.**

Hint: You will need to use **CLIENT_GET_FILENAME** and **CLIENT_IMAGE.READ_IMAGE_FILE**. You can import the code from **pr24_1.txt**.

- a) Open the Property Palette for the CONTROL.Exit_Button button. Change its name to NEW_IMAGE_BTN and its Label to New Image.
 - b) Open the When-Button-Pressed trigger for the button and replace its code with the following code imported from pr24_1.txt:

```
DECLARE
    v_file VARCHAR2(250) := 
    client_get_file_name('''','','JPEG Files|*.jpg',
        'Select a product image',open_file,TRUE);
    it_image_id ITEM := 
    FIND_ITEM('control.product_image');
BEGIN
    client_image.read_image_file(v_file,'',it_image_id);
END;
```

- 2) Run the form to test it. Try to load the 2056.jpg image in the lab directory or Sample.jpg in My Pictures.

Note: Because the image item is not a base table item, the new image is not saved when you exit the form. Also, because of triggers you have defined, navigating to another block, or to another record and then back, reloads the original image. You can code around this if you had a need to do so.

Practice 24-3: Interacting with the Client Environment

In this practice, you obtain the value of a client environment variable.

- 1) Display a message on the status line that contains the value of the PATH environment variable. You can import the code from `pr24_2.txt`.**

- a) From the Corporate tab of the Summit object library, drag MEDIUM_BTN to the CONTROL Block, choosing to Copy rather than Subclass.
 - b) Change the label to Test, the Canvas to Toolbar, and add the following code in a When-Button-Pressed trigger:

```
DECLARE
    v_message VARCHAR2(1000);
BEGIN
    CLIENT_TOOL_ENV.GETVAR('PATH',v_message);
    MESSAGE('Path is: '||v_message);
END;
```

- 2) Run the form to test the button.**

No formal solution

Practice 24-4: Creating a Client File

In this practice, you create a file on the client and open it.

1) Recode the Test button to:

- a) Create a file called `hello.txt` in the `d:\temp` directory that contains the text “Hello World”.
- b) Invoke Notepad to display this file, and delete the file after displaying it. You can import the code from `pr24_3.txt`.

When-Button-Pressed on the Test button:

```
DECLARE
    v_dir VARCHAR2(250) := 'd:\temp';
    ft_tempfile CLIENT_TEXT_IO.FILE_TYPE;
BEGIN
    ft_tempfile := CLIENT_TEXT_IO.FOPEN(v_dir ||
        '\hello.txt', 'w');
    CLIENT_TEXT_IO.PUT_LINE(ft_tempfile, 'Hello World');
    CLIENT_TEXT_IO.FCLOSE(ft_tempfile);
    SYNCHRONIZE;
    CLIENT_HOST('cmd /c notepad ' || v_dir || '\hello.txt');
    CLIENT_HOST('cmd /c del ' || v_dir || '\hello.txt');
END;
```

2) Run the form to test the button.

No formal solution

Practices for Lesson 25

In the practices for this lesson, you define a multiple-form application and pass values between forms.

Practice 25-1: Calling One Form from Another

In this practice, you define a button to open the Orders form from the Customers form.

- 1) In the Customers form, create a CONTROL block button called Orders_Button and set appropriate properties. Place it on the CV_CUSTOMER canvas below the Customer_Id item.**

- a) From the Summit object library, drag Medium_Btn into the CONTROL block of the Customers form, choosing Copy rather than Subclass.
 - b) Set Name to ORDERS_BUTTON.
 - c) Set Label to Orders.
 - d) Set Canvas to CV_CUSTOMER.
 - e) Open CV_CUSTOMER and position the button.

- 2) Add a code to the button to call the Orders form.**

When-Button-Pressed on CONTROL.Orders_Button (use the file name, *not* the form name, as the argument):

```
OPEN_FORM('ordgxx');
```

Note: Be sure to change the name of the form to open.

- 3) Run the Customers form to test the button.**

No formal solution

- 4) Change the window location of the Orders form, if required, so that it does not completely cover the Customers form when both are open.**

- a) Set the X and Y position properties in WIN_ORDER to larger values, such as 100.
 - b) Recompile the Orders form by selecting Program > Compile Module (this regenerates the .fmx file.)

Practice 25-2: Sharing Data between Forms

Currently there is no coordination between the calling Customers form and the called Orders form. In this practice, you define a PL/SQL package for the purpose of coordinating the open forms so that both show data for the same customer.

- 1) Open the Summit_Lib PL/SQL library and add a package named `Summit_Vars` with a single numeric variable, `CustId`, for the customer ID.**
 - a) With Summit_Lib open in the Object Navigator, select its Program Units subnode and click Create.
 - b) In the New Program Unit dialog box, select the Package Spec option and name the package `Summit_Vars`.
 - c) In the PL/SQL editor, add the following declaration:
`custid NUMBER;`
 - d) Compile the package and save the library.
- 2) Ensure that the library is attached to the Orders and Customers forms.**

It should already be attached. If you have saved the library under a different name, detach the previous library and attach the new one.
- 3) Add code to ensure that queries on the ORDERS block are restricted by the value of the PL/SQL variable if that variable has a value assigned to it.**
 - a) When-Button-Pressed on CONTROL.Orders_Button in Customers form:
`summit_vars.custid := :customers.customer_id;
OPEN_FORM('ordgxx',ACTIVATE,NO_SESSION,SHARE_LIBRARY_DATA);`

Note: Be sure to change the name of the form to be opened.
 - b) Orders form Pre-Query on the ORDERS block:
`:orders.customer_id := summit_vars.custid;`
- 4) Save, compile, and run the Orders form to test that it works as a stand alone form.**
- 5) Run the Customers form and test the button to open the Orders form.**
 - a) The Orders form should initially display the orders for the customer that is displayed in the Customers form.
 - b) Notice what happens when you leave both forms open and navigate to a different customer in the Customers form. When you navigate back to the Orders form, it still displays the orders for the original customer, not for the one that is currently selected in the Customers form. If you click Orders again, another Orders form opens. You fix these issues next.

Practice 25-3: Coding for Subsequent Queries

Currently if you leave both forms open and navigate to a different customer in the Customers form, the Orders form still displays the orders for the previous customer. In this practice, you write triggers to refresh the Orders form to display orders for the current customer. You also assign the current customer ID to a new order.

- 1) Alter the Orders_Button trigger in Customers so that it does not open more than one instance of the Orders form, but uses GO_FORM to pass control to Orders if the form is already running. Use the FIND_FORM built-in for this purpose. You can import the code from pr25_1.txt and then change the first argument of the OPEN_FORM command to the file name of your form.**

When-Button-Pressed on CONTROL.Orders_Button in the Customers form:

```
summit_vars.custid:= :customers.customer_id;
IF ID_NULL(FIND_FORM('orders')) THEN
    OPEN_FORM('ordgxx',ACTIVATE,NO_SESSION,
              SHARE_LIBRARY_DATA);
ELSE
    GO_FORM('orders');
END IF;
```

- 2) Add code to the Customers form so that the value of the custid PL/SQL variable is updated when navigating to another record.**

When-New-Record-Instance on CUSTOMERS block:

```
summit_vars.custid := :customers.customer_id;
```

- 3) If you navigate to a second customer record and click the Orders button or navigate to the Orders form, the Orders form still displays the records for the previous customer. Write a trigger to reexecute the query in the Orders form in this situation. You can import the code from pr25_2.txt.**

When-Form-Navigate trigger on the Orders form:

```
IF summit_vars.custid IS NOT NULL
AND :orders.customer_id != summit_vars.custid THEN
    init_query('orders');
END IF;
```

- 4) In the Orders form, write a When-Create-Record trigger on the ORDERS block that uses the value of the custid PL/SQL variable as the default value for ORDERS.Customer_Id.**

When-Create-Record on the ORDERS block:

```
:orders.customer_id := summit_vars.custid;
```

- 5) Save and compile the Orders form. Save, compile, and run the Customers form to test the functionality.**

No formal solution

- 6) If you have time, you can modify the appearance of the Orders form to make it easier to read, similar to the screenshot below.**

Practice 25-3: Coding for Subsequent Queries (continued)

Order Order Information Order
Id Date

Online? Order Status

Customer: ID Name

Sales Rep: ID Name



- a) Modify item prompts, widths, and colors and add boilerplate text and rectangles.
- b) With the rectangles selected, select Layout > Send to Back.

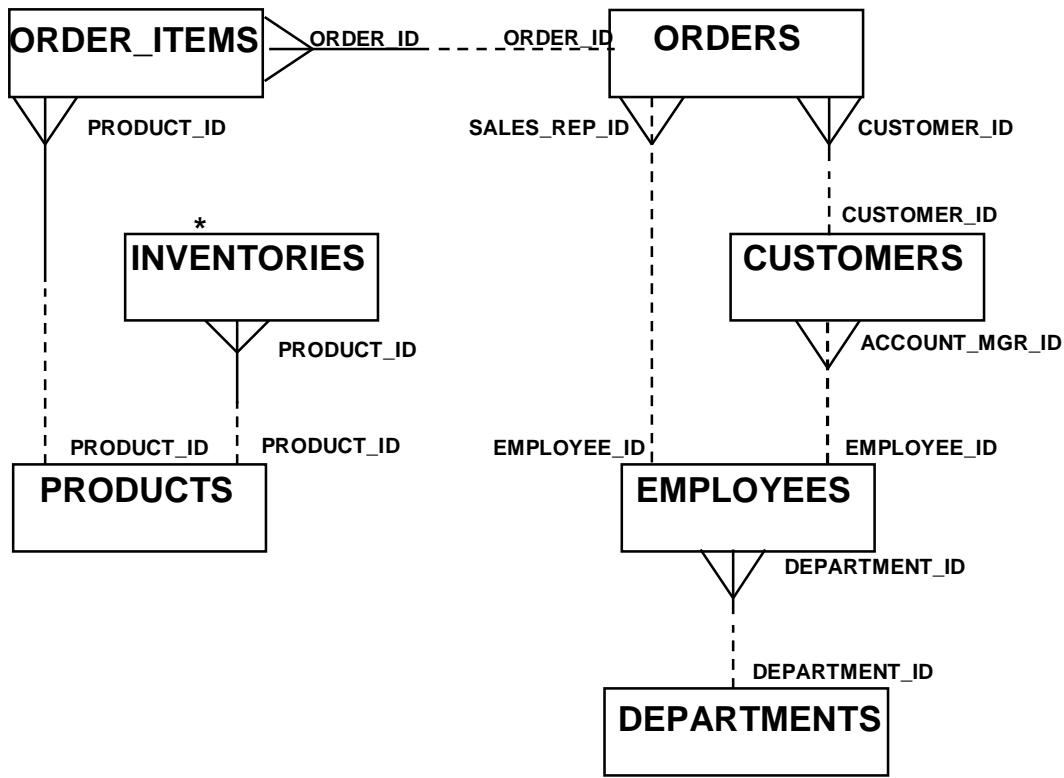
B

Table Descriptions

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Summit Office Supply Database Diagram



Unique occurrences in the Inventories table are identified by PRODUCT_ID and WAREHOUSE_ID.

The graphic shows that:

- The Orders table has a one-to-many relationship with Order_Items (ORDER_ID).
- The Products table has a one-to-many relationship with Order_Items (PRODUCT_ID) and Inventories (PRODUCT_ID.)
- The Employees table has a one-to-many relationship with Customers (EMPLOYEE_ID and ACCOUNT_MGR_ID) and Orders (EMPLOYEE_ID and SALES REP_ID.)
- The Departments table has a one-to-many relationship with Employees (DEPARTMENT_ID.)

Note: What follows is not a complete list of schema objects, but only those relevant to the Summit Office Supply application.

CUSTOMERS Description

```
SQL> desc customers
Name          Null?    Type
-----  -----
CUSTOMER_ID      NOT NULL NUMBER(6)
CUST_FIRST_NAME  NOT NULL VARCHAR2(20)
CUST_LAST_NAME   NOT NULL VARCHAR2(20)
CUST_ADDRESS           CUST_ADDRESS_TYP
PHONE_NUMBERS        VARCHAR2(30)
NLS_LANGUAGE         VARCHAR2(3)
NLS_TERRITORY        VARCHAR2(30)
CREDIT_LIMIT         NUMBER(9,2)
CUST_EMAIL          VARCHAR2(30)
ACCOUNT_MGR_ID       NUMBER(6)
```

```
SQL> desc cust_address_typ
Name          Null?    Type
-----  -----
STREET_ADDRESS        VARCHAR2(40)
POSTAL_CODE          VARCHAR2(10)
CITY                 VARCHAR2(30)
STATE_PROVINCE        VARCHAR2(10)
COUNTRY_ID           CHAR(2)
```

Related object creation statements

```
CREATE TYPE cust_address_typ
AS OBJECT
( street_address      VARCHAR2(40)
, postal_code        VARCHAR2(10)
, city                VARCHAR2(30)
, state_province     VARCHAR2(10)
, country_id          CHAR(2)
);

CREATE TABLE customers
( customer_id        NUMBER(6)
, cust_first_name    VARCHAR2(20) CONSTRAINT cust_fname_nn NOT
NULL
, cust_last_name     VARCHAR2(20) CONSTRAINT cust_lname_nn NOT
NULL
, cust_address        cust_address_typ
, phone_numbers      varchar2(30)
, nls_language        VARCHAR2(3)
```

CUSTOMERS Description (continued)

```
, nls_territory      VARCHAR2(30)
, credit_limit       NUMBER(9,2)
, cust_email         VARCHAR2(30)
, account_mgr_id    NUMBER(6)
, CONSTRAINT          customer_credit_limit_max
                      CHECK (credit_limit <= 5000)
, CONSTRAINT          customer_id_min
                      CHECK (customer_id > 0)) ;

CREATE UNIQUE INDEX customers_pk
  ON customers (customer_id) ;

ALTER TABLE customers
ADD ( CONSTRAINT customers_pk
      PRIMARY KEY (customer_id)) ;

ALTER TABLE customers
ADD ( CONSTRAINT customers_account_manager_fk
      FOREIGN KEY (account_mgr_id)
      REFERENCES employees(employee_id)
      ON DELETE SET NULL) ;

CREATE SEQUENCE customers_seq
START WITH 982
INCREMENT BY 1
NOCACHE
NOCYCLE ;
```

Sample record (1 out of 319 customers):

CUSTOMER_ID	CUST_FIRST_NAME	CUST_LAST_NAME
-----	-----	-----
CUST_ADDRESS(STREET_ADDRESS, POSTAL_CODE, CITY, STATE_PROVINCE, COUNTRY_ID)		
-----	-----	-----
PHONE_NUMBERS		NLS_NLS_TERRITORY
CREDIT_LIMIT		
-----	-----	-----
CUST_EMAIL		ACCOUNT_MGR_ID
-----	-----	-----
101 Constantin	Welles	
CUST_ADDRESS_TYP('514 W Superior St', '46901', 'Kokomo', 'IN', 'US')		
+1 317 123 4104	us	AMERICA
100		
<u>Constantin.Welles@ANHINGA.COM</u>		

DEPARTMENTS Description

```
SQL> desc departments
Name          Null?    Type
-----  -----
DEPARTMENT_ID      NOT NULL NUMBER(4)
DEPARTMENT_NAME    NOT NULL VARCHAR2(30)
MANAGER_ID         NUMBER(6)
LOCATION_ID        NUMBER(4)
```

Related object creation statements

```
CREATE TABLE departments
  ( department_id      NUMBER(4)
  , department_name    VARCHAR2(30)
  CONSTRAINT dept_name_nn NOT NULL
  , manager_id         NUMBER(6)
  , location_id        NUMBER(4)
  ) ;

CREATE UNIQUE INDEX dept_id_pk
ON departments (department_id) ;

ALTER TABLE departments
ADD ( CONSTRAINT dept_id_pk
      PRIMARY KEY (department_id)
  , CONSTRAINT dept_loc_fk
      FOREIGN KEY (location_id)
      REFERENCES locations (location_id)
  ) ;
```

Note: The Locations table is not documented here because it is not used in the Summit Office Supply application.

```
Rem Useful for any subsequent addition of rows to
      departments table
```

```
Rem Starts with 280
```

```
CREATE SEQUENCE departments_seq
  START WITH      280
  INCREMENT BY   10
  MAXVALUE       9990
  NOCACHE
  NOCYCLE;
```

DEPARTMENTS Description (continued)

```
SQL> select * from departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		1700
140	Control And Credit		1700
150	Shareholder Services		1700
160	Benefits		1700
170	Manufacturing		1700
180	Construction		1700
190	Contracting		1700
200	Operations		1700
210	IT Support		1700
220	NOC		1700
230	IT Helpdesk		1700
240	Government Sales		1700
250	Retail Sales		1700
260	Recruiting		1700
270	Payroll		1700

27 rows selected.

EMPLOYEES Description

```
SQL> desc employees
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

Related object creation statements

```
CREATE TABLE employees
(
    employee_id      NUMBER(6)
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25)
    CONSTRAINT      emp_last_name_nn NOT NULL
    , email          VARCHAR2(25)
    CONSTRAINT      emp_email_nn NOT NULL
    , phone_number   VARCHAR2(20)
    , hire_date      DATE
    CONSTRAINT      emp_hire_date_nn NOT NULL
    , job_id         VARCHAR2(10)
    CONSTRAINT      emp_job_nn NOT NULL
    , salary          NUMBER(8,2)
    , commission_pct NUMBER(2,2)
    , manager_id     NUMBER(6)
    , department_id   NUMBER(4)
    , CONSTRAINT      emp_salary_min
                      CHECK (salary > 0)
    , CONSTRAINT      emp_email_uk
                      UNIQUE (email)
) ;
```

EMPLOYEES Description (continued)

```
CREATE UNIQUE INDEX emp_emp_id_pk
ON employees (employee_id) ;
ALTER TABLE employees
ADD ( CONSTRAINT      emp_emp_id_pk
                  PRIMARY KEY (employee_id)
, CONSTRAINT      emp_dept_fk
                  FOREIGN KEY (department_id)
                  REFERENCES departments
, CONSTRAINT      emp_job_fk
                  FOREIGN KEY (job_id)
                  REFERENCES jobs (job_id)
, CONSTRAINT      emp_manager_fk
                  FOREIGN KEY (manager_id)
                  REFERENCES employees
) ;
ALTER TABLE departments
ADD ( CONSTRAINT dept_mgr_fk
                  FOREIGN KEY (manager_id)
                  REFERENCES employees (employee_id)
) ;
Rem Useful for any subsequent addition of rows to
employees table
Rem Starts with 207
CREATE SEQUENCE employees_seq
START WITH      207
INCREMENT BY   1
NOCACHE
NOCYCLE;
```

Sample records (2 out of 107 employees):

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	
100 AD_PRES	Steven	King	SKING	515.123.4567	17-JUN-87 90
101 AD_VP	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89 100 90

INVENTORIES Description and Data

```
SQL> desc inventories
```

Name	Null?	Type
PRODUCT_ID	NOT NULL	NUMBER(6)
WAREHOUSE_ID	NOT NULL	NUMBER(3)
QUANTITY_ON_HAND	NOT NULL	NUMBER(8)

Related object creation statements:

```
CREATE TABLE inventories
  ( product_id          NUMBER(6)
    , warehouse_id       NUMBER(3) CONSTRAINT
        inventory_warehouse_id_nn NOT NULL
    , quantity_on_hand   NUMBER(8)
  CONSTRAINT inventory_qoh_nn NOT NULL
    , CONSTRAINT inventory_pk PRIMARY KEY (product_id, warehouse_id)
  ) ;

ALTER TABLE inventories
ADD ( CONSTRAINT inventories_warehouses_fk
      FOREIGN KEY (warehouse_id)
      REFERENCES warehouses (warehouse_id)
      ENABLE NOVALIDATE
    ) ;

ALTER TABLE inventories
ADD ( CONSTRAINT inventories_product_id_fk
      FOREIGN KEY (product_id)
      REFERENCES product_information (product_id)
    ) ;
```

Sample records (11 out of 1112 inventory items):

PRODUCT_ID	WAREHOUSE_ID	QUANTITY_ON_HAND
1733	1	106
1734	1	106
1737	1	106
1738	1	107
1745	1	108
1748	1	108
2278	1	125
2316	1	131
2319	1	117
2322	1	118
2323	1	118

PRODUCTS Description and Data

```
SQL> desc products
Name          Null?    Type
-----  -----
PRODUCT_ID      NOT NULL NUMBER( 6 )
LANGUAGE_ID           VARCHAR2( 3 )
PRODUCT_NAME        NVARCHAR2( 250 )
CATEGORY_ID         NUMBER( 2 )
PRODUCT_DESCRIPTION NVARCHAR2( 4000 )
WEIGHT_CLASS        NUMBER( 1 )
WARRANTY_PERIOD     NUMBER( 5 )
SUPPLIER_ID         NUMBER( 6 )
PRODUCT_STATUS      VARCHAR2( 20 )
LIST_PRICE          NUMBER( 8 , 2 )
MIN_PRICE           NUMBER( 8 , 2 )
CATALOG_URL         VARCHAR2( 50 )
```

Related object creation statements:

```
CREATE OR REPLACE VIEW products
AS
SELECT i.product_id
,       d.language_id
,       CASE WHEN d.language_id IS NOT NULL
          THEN d.translated_name
          ELSE TRANSLATE(i.product_name USING NCHAR_CS)
      END      AS product_name
,       i.category_id
,       CASE WHEN d.language_id IS NOT NULL
          THEN d.translated_description
          ELSE TRANSLATE(i.product_description USING NCHAR_CS)
      END      AS product_description
,       i.weight_class
,       i.warranty_period
,       i.supplier_id
,       i.product_status
,       i.list_price
,       i.min_price
,       i.catalog_url
FROM   product_information i
,       product_descriptions d
WHERE  d.product_id (+) = i.product_id
AND    d.language_id (+) = sys_context('USERENV', 'LANG');
```

PRODUCTS Description and Data (continued)

```
CREATE TABLE product_information
( product_id          NUMBER(6)
, product_name        VARCHAR2(50)
, product_description VARCHAR2(2000)
, category_id         NUMBER(2)
, weight_class        NUMBER(1)
, warranty_period    NUMBER(5)
, supplier_id         NUMBER(6)
, product_status      VARCHAR2(20)
, list_price          NUMBER(8,2)
, min_price           NUMBER(8,2)
, catalog_url         VARCHAR2(50)
, CONSTRAINT          product_status_lov
                      CHECK (product_status in ('orderable',
'planned','development','obsolete'))) ;

ALTER TABLE product_information
ADD ( CONSTRAINT product_information_pk PRIMARY KEY (product_id)) ;

CREATE TABLE product_descriptions
( product_id          NUMBER(6)
, language_id         VARCHAR2(3)
, translated_name     NVARCHAR2(50)
CONSTRAINT translated_name_nn NOT NULL
, translated_description NVARCHAR2(2000)
CONSTRAINT translated_desc_nn NOT NULL);

CREATE UNIQUE INDEX prd_desc_pk
ON product_descriptions(product_id,language_id) ;

ALTER TABLE product_descriptions
ADD ( CONSTRAINT product_descriptions_pk
      PRIMARY KEY (product_id, language_id));
```

Sample records (4 out of 288 products; only 3 columns shown):

PRODUCT_ID	PRODUCT_NAME	LIST_PRICE
1726	LCD Monitor 11/PM	259
2359	LCD Monitor 9/PM	249
3060	Monitor 17/HR	299
2243	Monitor 17/HR/F	350

ORDER_ITEMS Description

```
SQL> desc order_items;
Name           Null?    Type
-----  -----
ORDER_ID        NOT NULL NUMBER(12)
LINE_ITEM_ID    NOT NULL NUMBER(3)
PRODUCT_ID      NOT NULL NUMBER(6)
UNIT_PRICE      NUMBER(8,2)
QUANTITY        NUMBER(8)
```

Related object creation statements:

```
CREATE TABLE order_items
( order_id          NUMBER(12)
, line_item_id     NUMBER(3)  NOT NULL
, product_id       NUMBER(6)  NOT NULL
, unit_price       NUMBER(8,2)
, quantity         NUMBER(8)
) ;

CREATE UNIQUE INDEX order_items_pk
ON order_items (order_id, line_item_id) ;

CREATE UNIQUE INDEX order_items_uk
ON order_items (order_id, product_id) ;

ALTER TABLE order_items
ADD ( CONSTRAINT order_items_pk PRIMARY KEY (order_id, line_item_id)
);

ALTER TABLE order_items
ADD ( CONSTRAINT order_items_order_id_fk
FOREIGN KEY (order_id)
REFERENCES orders(order_id)
ON DELETE CASCADE enable novalidate) ;

ALTER TABLE order_items
ADD ( CONSTRAINT order_items_product_id_fk
FOREIGN KEY (product_id)
REFERENCES product_information(product_id)) ;
```

ORDER_ITEMS Description (continued)

```
CREATE OR REPLACE TRIGGER insert_ord_line
  BEFORE INSERT ON order_items
  FOR EACH ROW
DECLARE
  new_line number;
BEGIN
  SELECT (NVL(MAX(line_item_id),0)+1) INTO new_line
    FROM order_items
   WHERE order_id = :new.order_id;
  :new.line_item_id := new_line;
END;
```

Sample records (11 out of 665 order items):

ORDER_ID	LINE_ITEM_ID	PRODUCT_ID	UNIT_PRICE	QUANTITY
2355	1	2289	46	200
2356	1	2264	199.1	38
2357	1	2211	3.3	140
2358	1	1781	226.6	9
2359	1	2337	270.6	1
2360	1	2058	23	29
2361	1	2289	46	180
2362	1	2289	48	200
2363	1	2264	199.1	9
2364	1	1910	14	6
2365	1	2289	48	92

ORDERS Description and Data

```
SQL> desc orders
Name          Null?    Type
-----  -----
ORDER_ID      NOT NULL NUMBER(12)
ORDER_DATE    NOT NULL DATE
ORDER_MODE    VARCHAR2(8 )
CUSTOMER_ID   NOT NULL NUMBER(6 )
ORDER_STATUS  NUMBER(2 )
ORDER_TOTAL   NUMBER(8 ,2 )
SALES_REP_ID  NUMBER(6 )
PROMOTION_ID NUMBER(6 )
```

Related object creation statements:

```
CREATE TABLE orders
( order_id           NUMBER(12 )
, order_date         DATE
, order_mode        VARCHAR2(8 )
, customer_id       NUMBER(6 )
, order_status      NUMBER(2 )
, order_total       NUMBER(8 ,2 )
, sales_rep_id      NUMBER(6 )
, promotion_id      NUMBER(6 )
, CONSTRAINT order_mode_lov
                     CHECK (order_mode in ('direct','online'))
, constraint         order_total_min
                     check (order_total >= 0)) ;

CREATE UNIQUE INDEX order_pk
ON orders (order_id) ;

ALTER TABLE orders
ADD ( CONSTRAINT order_pk
      PRIMARY KEY (order_id)) ;

ALTER TABLE orders
ADD ( CONSTRAINT orders_sales_rep_fk
      FOREIGN KEY (sales_rep_id)
      REFERENCES employees(employee_id)
      ON DELETE SET NULL) ;
```

ORDERS Description and Data (continued)

```
ALTER TABLE orders
ADD ( CONSTRAINT orders_customer_id_fk
      FOREIGN KEY (customer_id)
      REFERENCES customers(customer_id)
      ON DELETE SET NULL) ;
```

Sample records (12 out of 105 orders):

ORDER_ID	ORDER_DAT	ORDER_MO	CUSTOMER_ID	ORDER_STATUS	ORDER_TOTAL
2458	16-AUG-99	direct	101	0	78279.6
2397	19-NOV-99	direct	102	1	42283.2
2454	02-OCT-99	direct	103	1	6653.4
2354	14-JUL-00	direct	104	0	46257
2358	08-JAN-00	direct	105	2	7826
2381	14-MAY-00	direct	106	3	23034.6
2440	31-AUG-99	direct	107	3	70576.9
2357	08-JAN-98	direct	108	5	59872.4
2394	10-FEB-00	direct	109	5	21863
2435	02-SEP-99	direct	144	6	62303
2455	20-SEP-99	direct	145	7	14087.5
2379	16-MAY-99	direct	146	8	17848.2

C

Introduction to Query Builder

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

What Is Query Builder?

- Easy-to-use data access tool
- Point-and-click graphical user interface
- Distributed data access
- Powerful query building



C - 2

Copyright © 2009, Oracle. All rights reserved.

What Is Query Builder?

Query Builder is the tool used in the LOV Wizard to build the query on which a record group is based.

Easy-to-Use Data Access Tool

Query Builder is an easy-to-use data access tool. It provides a logical and intuitive means to access information stored in networked, distributed databases for analysis and reporting.

Point-and-Click Graphical User Interface

Query Builder enables you to become productive quickly because its graphical user interface works in the same way as your other applications. A toolbar enables you to perform common operations quickly.

Distributed Data Access

Query Builder represents all database objects (tables, views, and so on) as graphical data sources, which look and work exactly the same way regardless of which database or account the data came from.

What Is Query Builder? (continued)

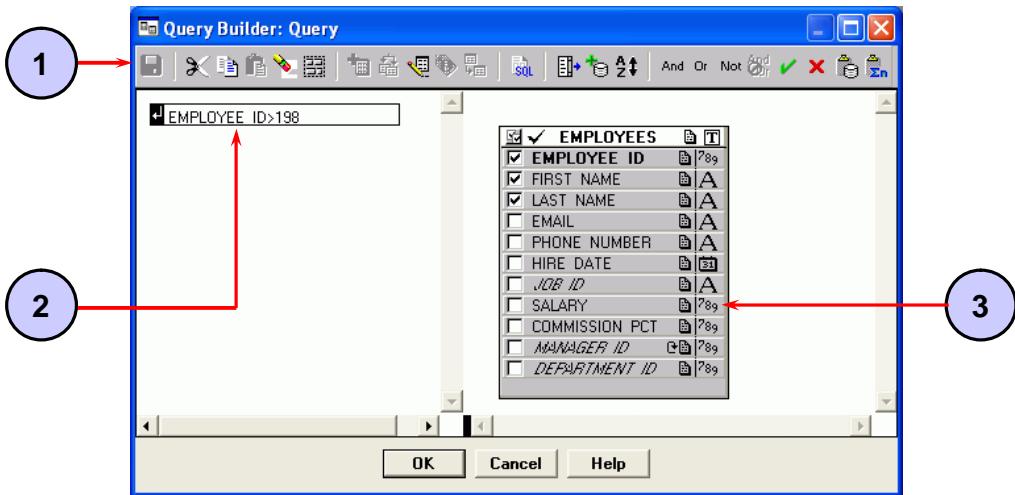
Query Builder is useful for the following reasons:

- Performing distributed queries on complex, enterprisewide databases is no more difficult than querying a single database.
- Locating database objects is easy because Query Builder uses a single hierarchical directory that lists all accessible data in your account, in other accounts, and in other databases.
- Graphical representation of tables and their relationships enables you to see the structure of your data.

With Query Builder, you can:

- Build queries by clicking the columns that you want to retrieve from the database. The browser generates the necessary SQL statements behind the scenes.
- Specify exactly which rows to retrieve from the database by using conditions, which consist of any valid SQL expression that evaluates to true or false
- Combine and nest conditions graphically using logical operators. You can disable conditions temporarily for *what-if* analysis.

Query Builder Window



ORACLE®

C - 4

Copyright © 2009, Oracle. All rights reserved.

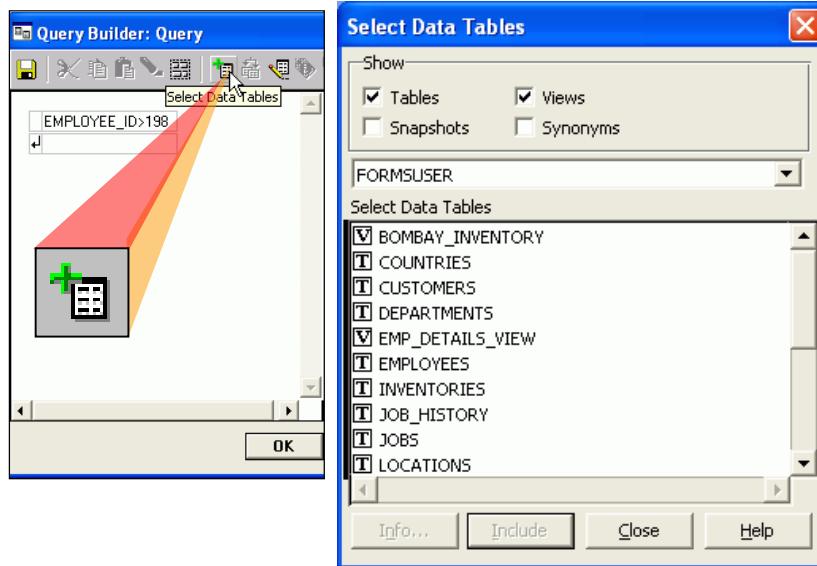
Query Builder Window

The Query Builder graphical user interface consists of one window, the Query Builder window, where you build your queries.

As shown in the screenshot, the Query Builder window comprises:

1. The Toolbar, which enables you to issue commands with a click. You can create conditions, add new data sources, or define new columns.
2. The Conditions panel, where you specify conditions to refine your queries
3. The Data Source panel, where you display and select tables and columns for a query

Building a New Query



ORACLE®

C - 5

Copyright © 2009, Oracle. All rights reserved.

Building a New Query

To build a query, you must select the tables you want to include and the columns you want to retrieve.

How to Include a Table

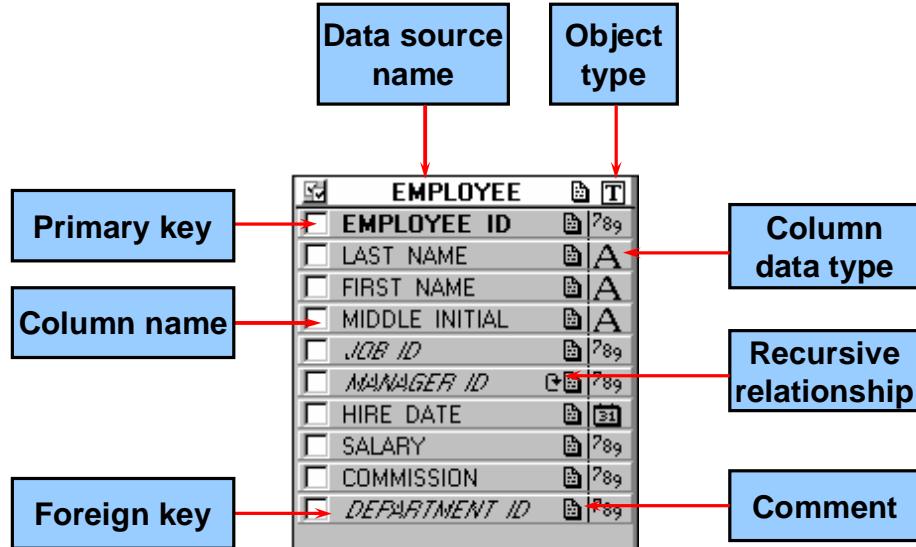
1. Click Select Data Tables on the toolbar. The Select Data Tables dialog box appears.
Note: When you open Query Builder to create a new query, the Select Data Tables dialog box is open by default.
2. Select the table name and then click Include, or simply double-click the desired table name.
The selected table appears in the Query window.
3. Click Close to close the dialog box.

You can, at any time, include additional tables in the query by following these steps.

How to Delete a Table

1. Select the data source in the Query Builder window.
2. Click Clear on the toolbar or press Delete.

Data Source Components



ORACLE®

C - 6

Copyright © 2009, Oracle. All rights reserved.

Data Source Components

Data Source name

An object that has been included in a query is referred to as a *data source*. It is displayed as a rectangular graphic in the Data Source panel of the Query window. The top part of the rectangle contains the table name and an icon representing the object type:

Object Type	Icon	Description
Table	█ T square with T inside	Stores data in the database
View	█ V square with V inside	Acts like a table when you execute a query, but is really a pointer to either a subset of a table, a combination of tables, or a join of two or more tables.
Synonym	█ S square with S inside	Another name for an object. Sometimes table names can be rather cryptic, such as <code>emp_em_con_tbl</code> . You can create a synonym that simply calls this table Contracts.
Alias	█ A square with A inside	Query Builder uses an alias name for a table when the table is used more than once in a query, mostly with self-joins. You can also rename a table.

Data Source Components (continued)

Columns

The body of the rectangle, as shown in the screenshot in the slide, contains column names listed vertically. To the right of the column name is an icon representing the data type.

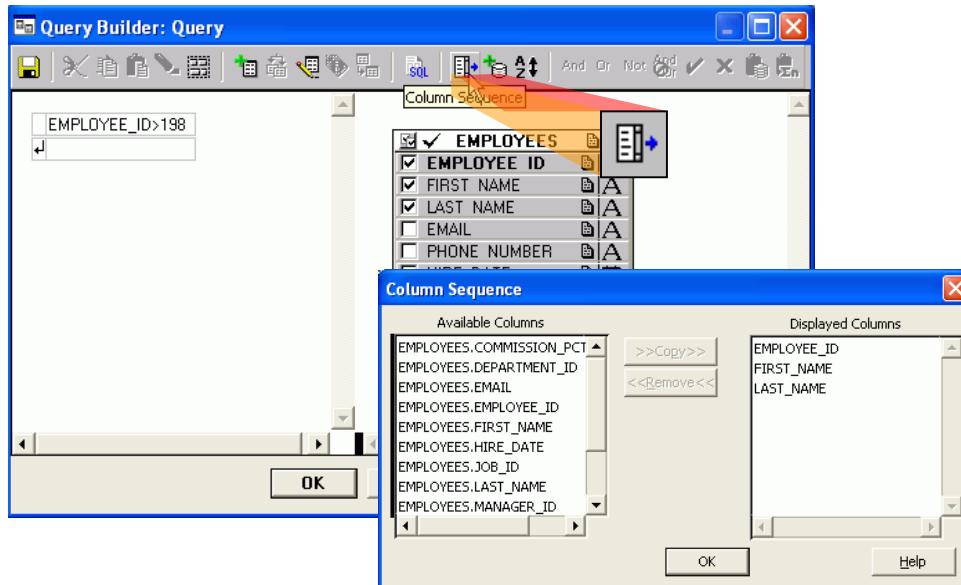
The screenshot shows data types of:

- **Number:** The icon is the numbers 7, 8, and 9.
- **Varchar2:** The icon is the letter A.
- **Date:** The icon is a small calendar page with a 31 inside.

Columns also provide additional information.

- **Primary keys** are displayed in bold (EMPLOYEE_ID in the screenshot).
- **Foreign keys** are displayed in italics (JOB_ID, MANAGER_ID, and DEPARTMENT_ID).
- **Recursive relationships** are indicated by a self-relationship icon, an arrow pointing back to itself (the one beside MANAGER_ID in the screenshot).
- **Comments**, if present, are indicated by an icon that looks like a dog-eared page; you can double-click the icon to read the comment.

Refining a Query



ORACLE®

C - 8

Copyright © 2009, Oracle. All rights reserved.

Refining a Query

Adding Columns to a Query

You can add a column to a query either by selecting the check box to the left of the column name, or by double-clicking the column name. To include all columns from any single table, double-click the table heading.

Removing Columns from a Query

You can remove columns from a query either by clearing the check box to the right of the column name, or by double-clicking the column name. To remove all columns from any single table, double-click the table heading.

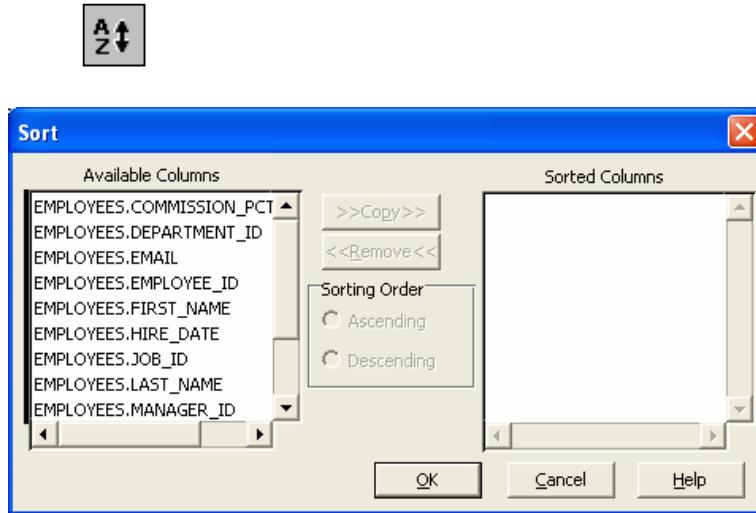
Changing the Column Position in a Query

By default, Query Builder places columns in the order in which you select them. You can resequence them by selecting the Column Sequence tool, whose icon displays a numbered list with an arrow as shown in the top screenshot in the slide.

The Column Sequence dialog box appears as shown in the slide. Column names are shown in the Displayed Columns list in the order of their appearance in the query. Drag any column to a new position.

Note: You can also use the Column Sequence dialog box to add columns to or remove them from a query.

Sorting Data



ORACLE®

C - 9

Copyright © 2009, Oracle. All rights reserved.

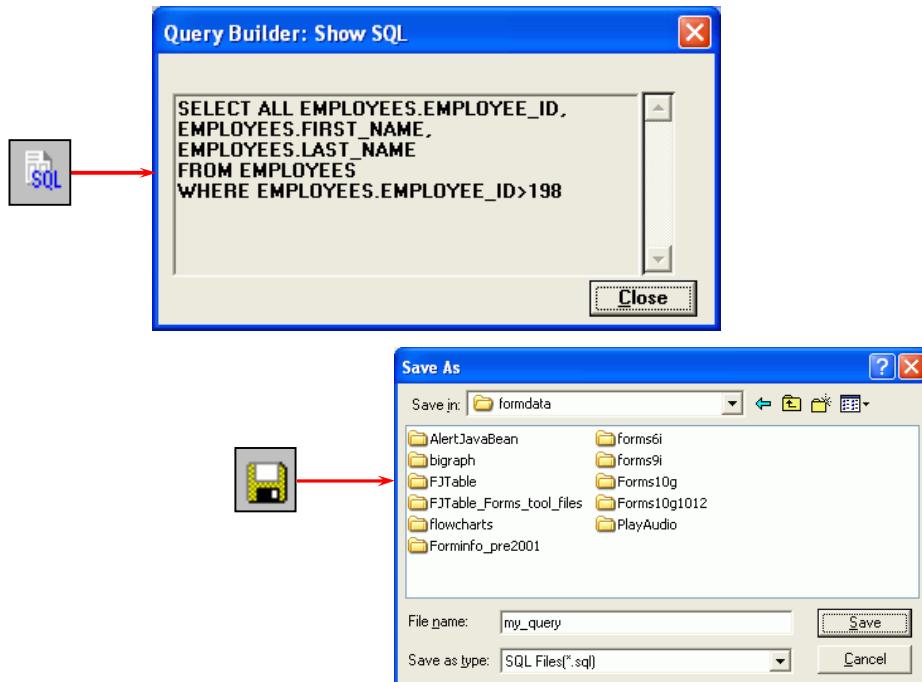
Sorting Data

By default, a query returns the data in no specific order. To sort the data, you must add an ORDER BY clause to the query.

How to Add an ORDER BY Clause to a Query

1. Select the Sort tool, whose icon looks like an A above a Z with a double-headed arrow next to it, as shown by the top screenshot in the slide. The Sort dialog box appears.
2. Select the column you want to sort from the Available Columns list.
3. Select Copy.
Query Builder adds the column to the Sorted Columns list and places an up arrow in the list box to indicate the default sort ascending order.
- Note:** You can sort by more than one column. Query Builder sorts according to the order in which columns appear in the Sorted Columns list.
4. You can change the sorting order by selecting the column name in the Sorted Columns list and selecting the desired option button.
5. Click OK to close the dialog box.

Viewing and Saving Queries



Viewing and Saving Queries

Viewing a Query

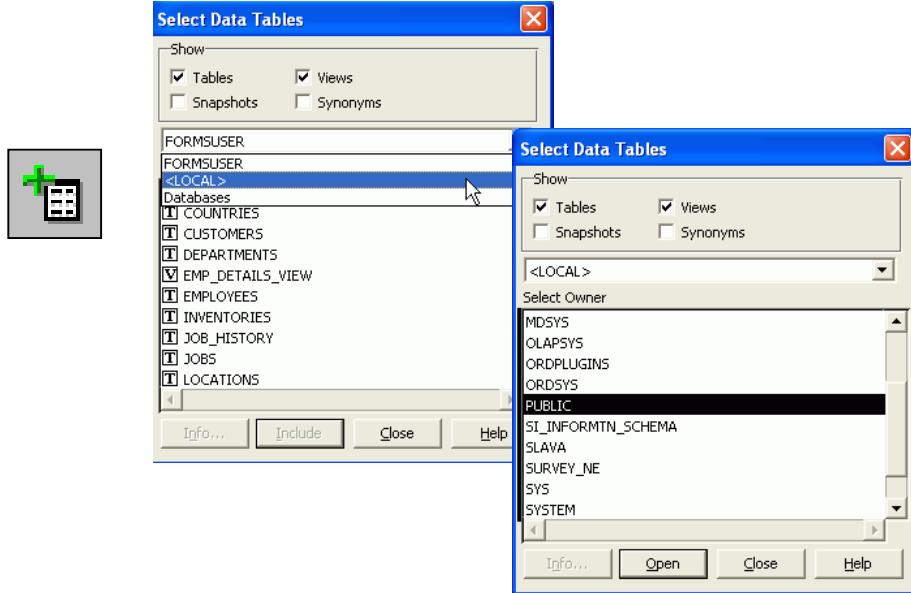
Click the Show SQL icon to view the query text that Query Builder will create. The Query Builder: Show SQL dialog box is shown in the slide.

How to Save a Query

You can save your query as a SQL statement to the file system.

1. Click the Save icon from the toolbar. The icon resembles a floppy disc. The Save As dialog box appears.
2. Enter a file name.
Note: If you do not enter a file extension, Query Builder automatically appends the .SQL file extension.
3. Select a destination.
4. Click OK.

Including Additional Tables



ORACLE®

C - 11

Copyright © 2009, Oracle. All rights reserved.

Including Additional Tables

Often, all the data needed to create a desired query cannot be found in a single table. With Query Builder, you can include multiple tables in a single query.

Remember that you can see the names of all the tables in your schema by clicking the Select Data Tables icon, which resembles a table with a green plus sign and is shown at the left of the slide. This displays the Select Data Tables dialog box. You can also use this dialog box to show only certain types of data sources, as well as data sources in other schemas and databases.

How to Find Tables in Other Schemas

If you do not find the table you are looking for in the Select Data Tables dialog box in your own schema, you can search other schemas.

1. Open the pop-up menu to display the name of the current database and the option databases.
2. Select the current database name. The list box displays a list of schemas that contain tables you can access.
3. Select the name of the schema in the list and then click Open, or simply double-click the schema name. This opens the schema and displays a list of objects in the schema.
4. Follow the normal procedure for including objects in the Query window.

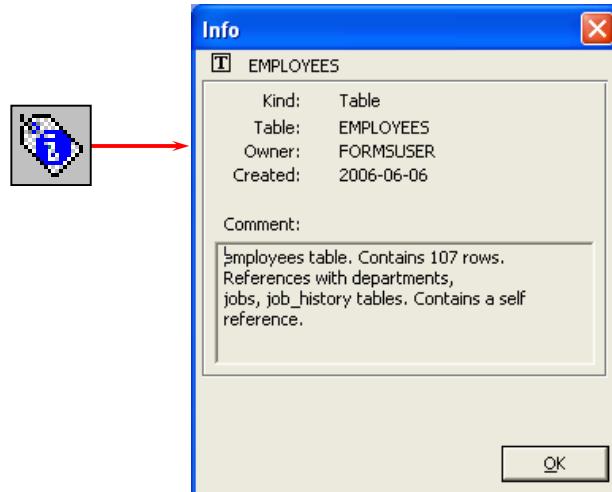
Including Additional Tables (continued)

How to Find Tables in Other Databases

If you do not find the table you are looking for in the Select Data Tables dialog box in your own schema, you can search other databases.

1. Open the pop-up menu to display the name of the current database and the option databases.
2. Select databases.
The list box displays a list of databases you can access.
3. Select the name of the database in the list and then click Open, or simply double-click the account name.
4. Follow the normal procedure for including tables in the Query window.

Viewing Comments



ORACLE®

C - 13

Copyright © 2009, Oracle. All rights reserved.

Viewing Comments

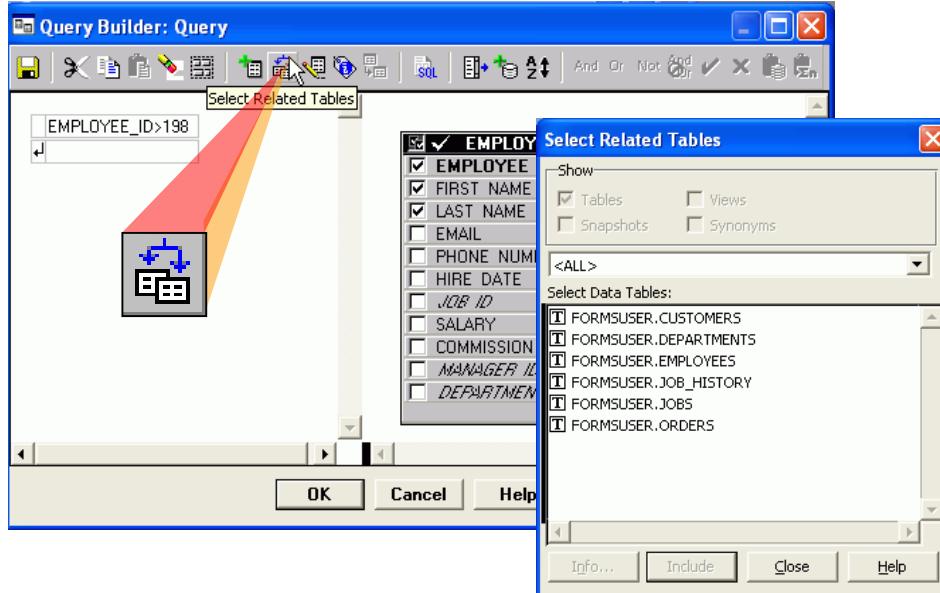
Sometimes, details about the kind of data stored in the database are not reflected by the table or column names. This can often make it difficult to decide which objects to include in your query. Query Builder features the Info dialog box to provide this kind of information for tables and columns.

To open the dialog box, follow these steps:

1. In the Data Source panel, select the table or column name.
2. Click the Get Info icon.
3. The Info window appears. It displays information about the object. The example in the slide, for the EMPLOYEES table, shows the owner, creation date, number of rows, and references that it contains.

Alternatively, you can double-click the Comment icon in each table or column in the Data Source panel.

Including Related Tables



Including Related Tables

To combine data from multiple tables into one query, Query Builder enables you to search for relationships between tables and to create user-defined relationships if they do not exist. Additionally, you can activate or deactivate relationships to suit your needs.

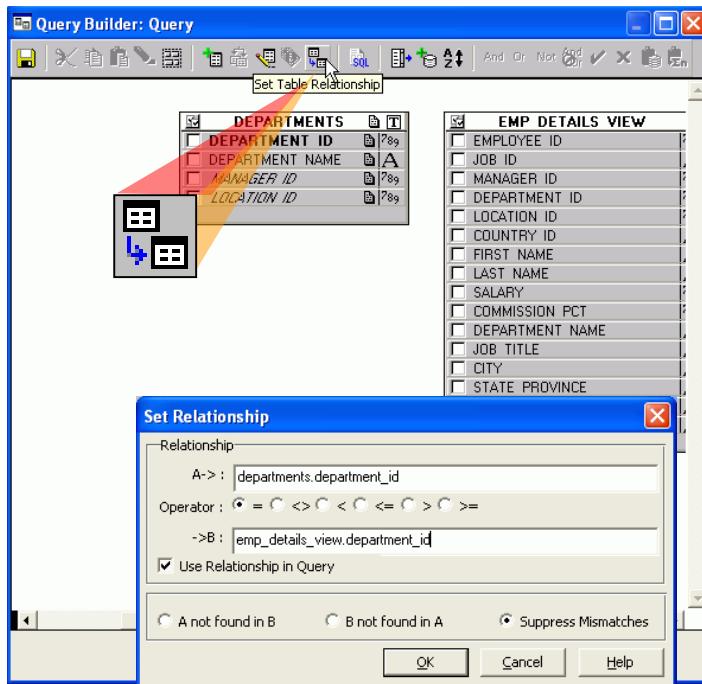
How to Find and Include Related Tables

1. Select the table in the Data Source panel.
2. Click the Select Related Tables icon. A list of tables that have relationships defined with the selected table appears.
3. Select the table name and click Include, or simply double-click the table name. The selected table appears in the Data Source panel. Relationships between the tables are identified by relationship lines, drawn from the primary keys in one table to the foreign keys in another.
4. Click Close to close the dialog box.

After you have included the table, you can retrieve its columns.

Note: When you select a foreign key column before selecting related tables, only the table to which the foreign key refers appears in the Select Related Tables dialog box.

Creating a User-Defined Relationship



ORACLE®

C - 15

Copyright © 2009, Oracle. All rights reserved.

Creating a User-Defined Relationship

When you create a user-defined relationship, Query Builder draws a relationship line connecting the related columns.

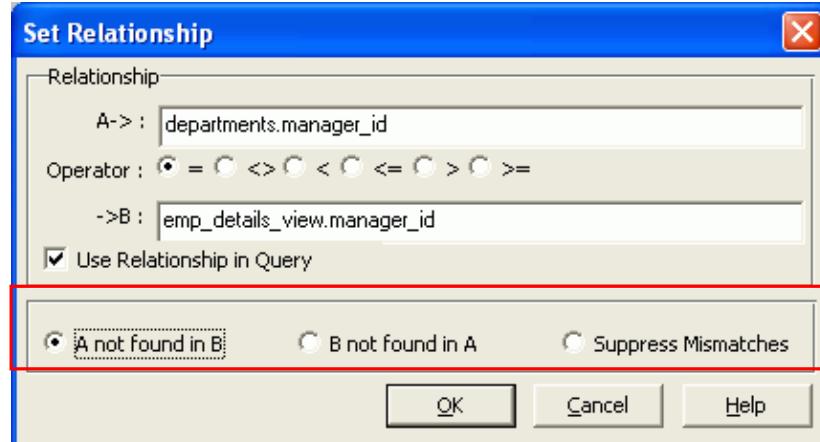
How to Create a Relationship

1. Click the Set Table Relationship icon.
The Set Relationship dialog box appears.
2. Enter the foreign key (A >) and primary key (B >) column names. Enter the complete table and column names (separate the table name from its column name with a period). The example in the screenshot shows the foreign key of `departments.department_id` and the primary key of `emp_details_view.department_id`.
3. Select the Operator option button to indicate `=`, `<>`, `<`, `<=`, or `>=`.
4. Select the check box indicating whether to use the relationship in the query.
5. Click OK to close the dialog box.

How to Create a Relationship (optional)

1. Select the column that you want to relate (foreign key).
2. Press and hold the mouse button, and drag the cursor to the related column in the second table (the primary key). You are drawing a relationship line as you do so.
3. After the target column is selected, release the mouse button to anchor the line.

Retrieving Unmatched Rows



ORACLE®

C - 16

Copyright © 2009, Oracle. All rights reserved.

Retrieving Unmatched Rows

Query Builder enables you to choose whether to retrieve any unmatched rows when you are using a relationship in a query. An unmatched row occurs when the relationship connects tables where there are values on one side that have no corresponding values on the other side.

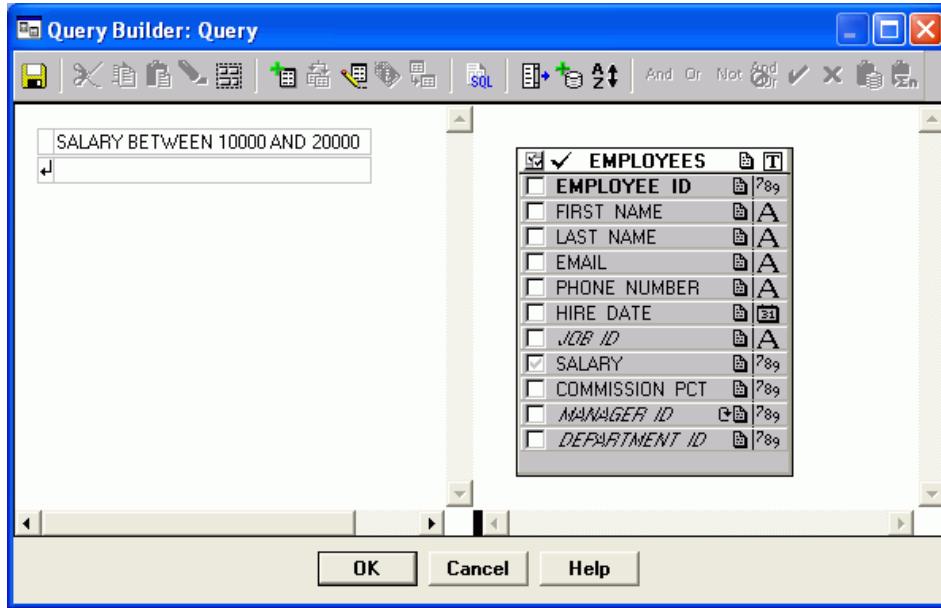
There are three types of relationships that you can choose from:

- Display records from table A not found in table B.
- Display records from table B not found in table A.
- Suppress any mismatched records (default).

How to Create an Unmatched Relationship

1. Click the relationship line that connects the tables.
Both the column names and the relationship line should be selected.
2. Click the Set Table Relationship icon on the toolbar.
The Set Relationship dialog box appears.
3. Select one of the three relationship option buttons.
4. Click OK.
An unmatched relationship icon is placed on the relationship line next to the column that returns unmatched rows.

Selecting Rows That Meet Conditions



ORACLE®

C - 17

Copyright © 2009, Oracle. All rights reserved.

Selecting Rows That Meet Conditions

Conditions Panel

The Query window contains two independently scrollable panels, the Conditions panel and the Data Source panel. The Data Source panel is where you include tables and columns. The Conditions panel is where you apply conditions. You enter conditions in the Condition field of the panel. The screenshot shows the EMPLOYEES table in the Data Source panel, while the Conditions panel shows the condition SALARY BETWEEN 10000 AND 20000.

How to Add Conditions to a Query

1. Activate the Conditions field.
2. Enter the text that describes the condition in one of the following ways:
 - Enter the conditions directly in the Condition field.
 - Click in the columns in the Data Source panel, and then enter the rest of the condition.
 - Select columns and functions, using the appropriate tools.

Note: Character and date values must be enclosed within single quotation marks.

Selecting Rows That Meet Conditions (continued)

3. Close and validate the condition. Query Builder automatically validates the condition when you close the Condition field. You can close the Condition field in any of the following ways:
 - Press Return.
 - Click in the Conditions panel outside the Condition field.
 - Click Accept on the toolbar.

Note: If a column is used in a condition but it is not displayed in the results window, a gray check mark appears to the left of the column name in the data source in the Query window.

Toolbar

The logical operators and the Accept and Cancel tools on the toolbar are active whenever the Condition field is active. You can insert an operator from the toolbar into the condition by clicking it.

Operators

Arithmetic:

- Perform calculations on numeric and date columns.
- Examples: +, -, x, and /

Logical:

- Combine conditions.
- Examples: AND, OR, and NOT

Comparison:

- Compare one expression with another.
- Examples: =, <>, <, IN, IS NULL, and BETWEEN . . . AND



Operators

An operator is a character or reserved word that is used to perform some operation in Query Builder, such as the + operator, which performs addition. Query Builder uses several types of operators.

Arithmetic Operators

Arithmetic (+, -, x, /) operators are used to perform calculations on numeric and date columns. In handling calculations, Query Builder first evaluates any multiplication or division, and then evaluates any addition or subtraction.

Logical Operators

Logical operators are used to combine conditions. They include:

- **AND:** Causes the browser to retrieve only data that meets all the conditions
- **OR:** Causes the browser to retrieve all data that meets at least one of the conditions
- **NOT:** Used to make a negative condition, such as NOT NULL

Comparison Operators

Comparison operators are used to compare one expression with another, such that the result will either be true or false. The browser returns all the data for which the result evaluates to true.

Operators (continued)

Operator	Usage
=	Equal
<>	Not equal
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
BETWEEN... AND...	Between two values
IN (LIST)	Equal to any member of the following list
IS NULL	Is a NULL value. A row without a value in one column is said to contain a NULL value.

Entering Multiple Conditions

AND	SAL BETWEEN 1000 AND 2000
	HIREDATE>='23-jan-86'
	<input type="button" value="↓"/>

AND	SAL BETWEEN 1000 AND 2000	
	OR	HIREDATE>='23-jan-86'
		DEPTNO=20
		<input type="button" value="↓"/>

ORACLE®

C - 21

Copyright © 2009, Oracle. All rights reserved.

Entering Multiple Conditions

There is no limit to the number of conditions you can include in a browser query. Multiple conditions are always combined using logical operators. You can add conditions to any query either before or after execution.

How to Add Conditions

The Conditions panel always displays a blank Condition field at the bottom of the list of conditions, as shown in the top screenshot in the slide. Use this field to enter multiple conditions.

1. Click in the empty Condition field to activate it.
2. Enter the new condition. Each time you add a condition, a new blank Condition field is created. The bottom screenshot shows the new condition.
Note: Pressing Shift + Return following the entry of each condition is the fastest way to create multiple conditions, because it moves the cursor and prompt down one line so that you can enter another condition.
3. Press Return to close and validate the condition.

How to Change Logical Operators

By default, Query Builder combines multiple conditions with the AND operator.

Entering Multiple Conditions (continued)

How to Change Logical Operators (continued)

To change the logical operator, perform the following steps:

1. Select the logical operator in the Conditions panel.
2. Click a new operator on the toolbar or select one from the Data menu.

How to Create Nested Conditions

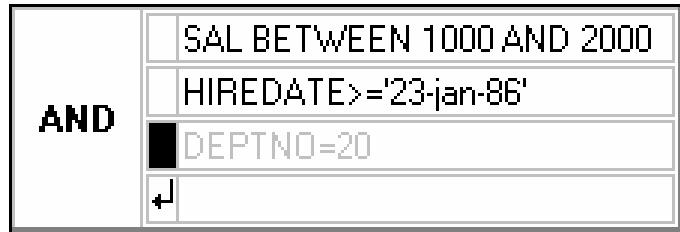
Query Builder enables you to combine logical operators to produce complex queries made up of multiple levels of conditions. These are referred to as nested conditions. To nest two or more conditions, perform the following steps:

1. Build each condition to be included in the nest.
2. Press and hold Shift and click in the box to the left of each condition to be nested.
3. Select the logical operator to combine the conditions—either AND or OR.

Query Builder draws a box around the highlighted conditions in the Conditions panel and combines them with the operator that you specified.

The bottom screenshot shows an OR operator added between the last two conditions.

Changing a Condition



ORACLE®

C - 23

Copyright © 2009, Oracle. All rights reserved.

Changing a Condition

If you change your mind about including one or more conditions in your query, you can delete, deactivate, or edit any of them in the Conditions panel.

How to Delete a Condition

1. Click inside the Condition field to activate it.
2. Select Clear from the toolbar or select Delete.
The condition is removed from the query.

How to Deactivate a Condition

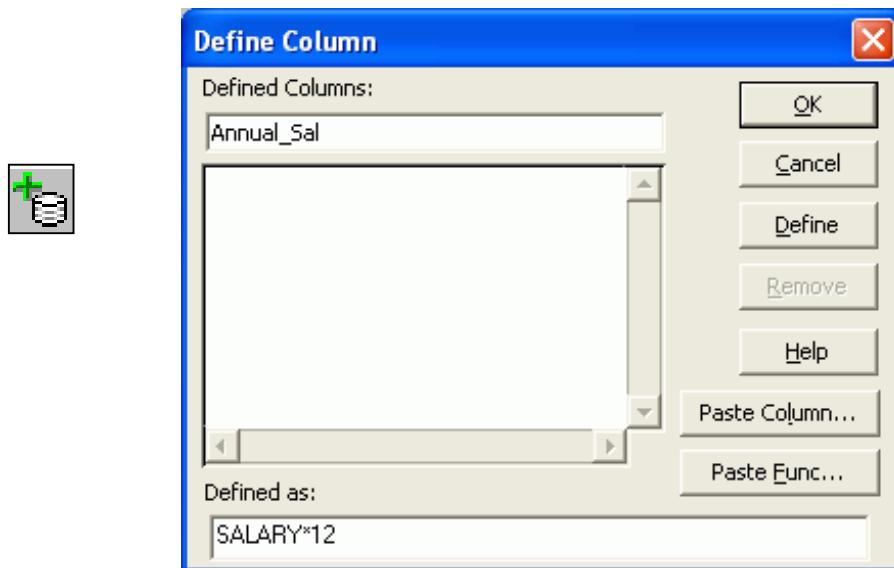
Query Builder enables you to temporarily deactivate a condition so that you can test different scenarios without having to re-create the conditions each time.

Double-clicking the box to the left of the condition or its operator acts as a toggle switch, alternately turning the condition on and off. Deactivated conditions remain in the Condition field but appear dimmed.

Additionally, you can turn conditions on and off by using the Data menu:

1. Select the box to the left of the condition or its operator.
Note: Press and hold the Shift to select multiple conditions.
2. Double-click in the box to deactivate the condition. The screenshot shows that the last condition was deactivated.

Defining Columns by Using an Expression



ORACLE®

C - 24

Copyright © 2009, Oracle. All rights reserved.

Defining Columns by Using an Expression

Besides retrieving columns of data stored in a table, Query Builder enables you to define new columns that are derived or calculated from the values in another column.

How to Define a Column

When you define a column, it exists only in your query, not in the database.

1. Select the table where you want to define a new column.
2. Click the Define Column icon.

The Defined Column dialog box appears, which displays a list of all columns currently defined in the query (if any).

3. Click in the Defined Column field to activate it, and then enter the name for your new column. The screenshot shows a column of Annual_Sal being defined.
4. Move your cursor to the Defined as field and enter the formula or expression that defines your column. The screenshot shows an expression of SALARY*12.
5. Click Define.

The new column is added to the list of defined columns in the dialog box and appears in the Data Source panel.

6. Click OK to close the dialog box.

Defining Columns by Using an Expression (continued)

How to Enter Expressions

You can enter information in the Defined as field in either of the following ways:

- Enter the expression directly.
- Click Paste Column.

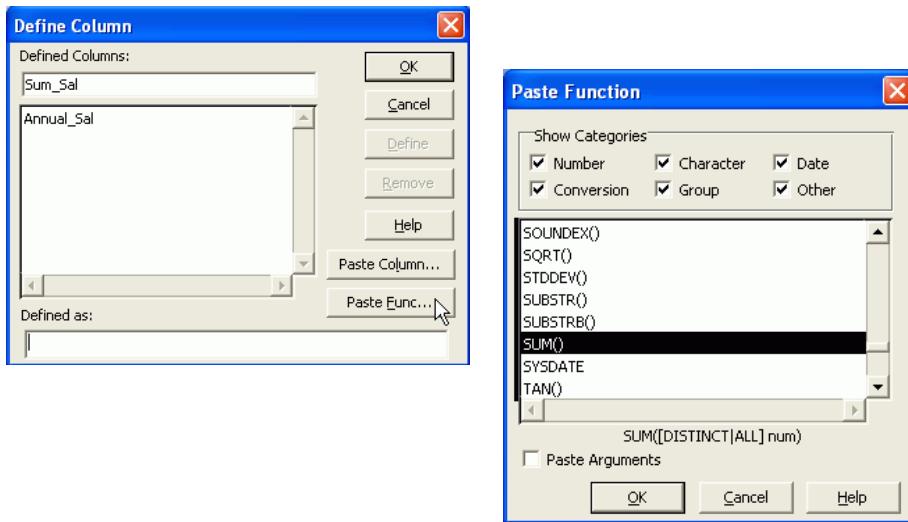
The Paste Column dialog box appears.

- Select the column from the displayed list.
- Click OK to paste the column into your expression and return to the Define Column dialog box.

How to Display and Hide Defined Columns

You display or hide defined columns from a query in exactly the same manner as you do ordinary columns.

Defining Columns by Using a Function



Defining Columns by Using a Function

The browser also enables you to define columns using a variety of built-in functions provided by the Oracle Server. You can enter a function directly in the Defined as field of the Define Column dialog box, or click Paste Function to select from a list.

What Is a Function?

A function is similar to an operator in that it performs a calculation and returns a result. Functions consist of a function name followed by parentheses, in which you indicate the arguments.

An argument is an expression that supplies information for the function to use. Functions usually include at least one argument, most commonly the name of the column on which the operation will be performed.

Single-Row Functions

- Return one value for every data row operated on
- Examples: INITCAP(), SUBSTR(), TRUNC()

Aggregate Functions

- Return a single row based on the input of multiple rows of data
- Examples: AVG(), COUNT(), SUM()

Defining Columns by Using a Function (continued)

How to Select a Function

1. Click in the Defined as field to activate it.
2. Click Paste Function. The Paste Function dialog box appears.
3. Select or deselect the Show Categories check boxes to view the desired list of functions.
4. Select the function from the displayed list.
5. Select the Paste Arguments check box (optional).

Note: If this check box is selected, a description or data type name of the arguments appropriate for the function is pasted into your expression. You can then replace the description with the actual arguments.

6. Click OK to paste the function into your expression and return to the Define Column dialog box. The screenshot shows the Sum() function being selected to be pasted into the Defined as field of the Define Column dialog box.

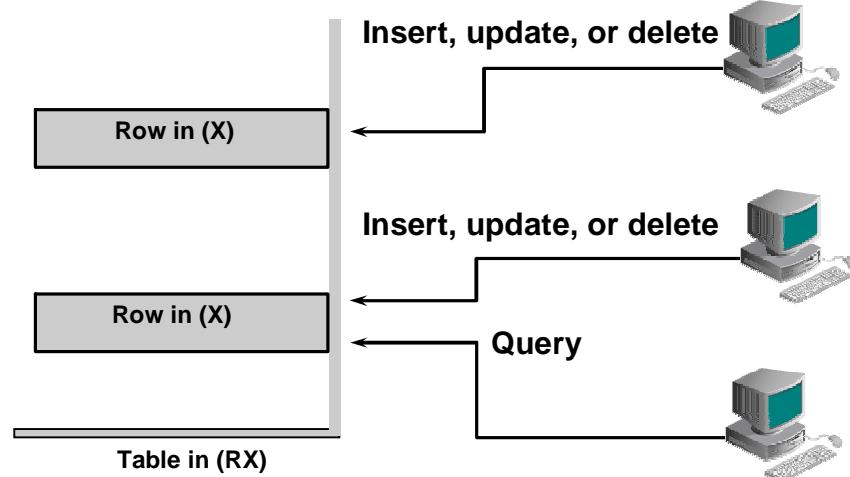
D

Locking in Forms

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Locking



ORACLE®

D - 2

Copyright © 2009, Oracle. All rights reserved.

Locking

In database applications, locking maintains the consistency and integrity of the data, where several users are potentially accessing the same tables and rows. Forms applications are involved in this locking process when they access database information.

Oracle Locking

Forms applications that connect to an Oracle database are subject to the standard locking mechanisms employed by the server. Here is a reminder of the main points:

- Oracle uses row-level locking to protect data that is being inserted, updated, or deleted.
- Queries do not prevent other database sessions from performing data manipulation. This also applies to the reverse situation.
- Locks are released at the end of a database transaction (following a rollback or commit).

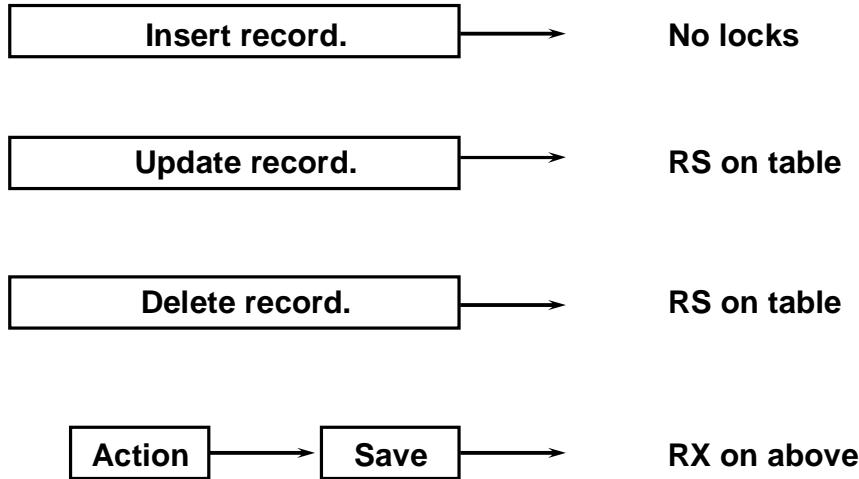
Locking (continued)

A session issues locks to prevent other sessions from performing certain actions on a row or table. The main Oracle locks that affect Forms applications are the following:

Lock Type	Description
Exclusive (X) row lock	Allows other sessions to only read the affected rows
Row Share (RS) table lock	Prevents the above lock (X) from being applied to the entire table; usually occurs because of SELECT_FOR_UPDATE
Row Exclusive (RX) table lock	Allows write operations on a table from several sessions simultaneously, but prevents (X) lock on entire table; usually occurs because of a DML operation

The graphic illustrates a table with a row-exclusive lock. A user can insert, update, or delete a row in that table with an exclusive lock, but while that lock is maintained other users can only query that particular row. However, other users can update, insert, or delete other rows in the table.

Default Locking in Forms



ORACLE®

D - 4

Copyright © 2009, Oracle. All rights reserved.

Default Locking in Forms

Forms initiates locking automatically when the operator inserts, updates, or deletes records in a base-table block. These locks are released when a save is complete.

Forms causes the following locks on Oracle base tables and rows, when the operator performs actions:

Operator Action	Locks
Insert a record	No locks
Update database items in a record*	Row Share (RS) on base table Exclusive (X) on corresponding row
Save	Row Exclusive (RX) on base tables during the posting process (Locks are released when actions are completed successfully.)

*Update of nondatabase items with the Lock Record property set to Yes also causes this.

The exclusive locks are applied to reserve rows that correspond to records that the operator is deleting or updating, so that other users cannot perform conflicting actions on these rows until the locking form has completed (Saved) its transaction.

Concurrent Updates and Deletes

- When users compete for the same record, normal locking protection applies.
- Forms tells the operator if another user has already locked the record.

ORACLE®

D - 5

Copyright © 2009, Oracle. All rights reserved.

Concurrent Updates and Deletes

What Happens When Users Compete for the Same Row?

Users who are only querying data are not affected here. If two users are attempting to update or delete the same record, integrity is protected by the locking that occurs automatically at row level.

Forms also keeps each user informed of these events through messages and alerts when:

- A row is already locked by another user. The user has the option of waiting or trying again later.
- Another user has committed changes since a record was queried. The user must requery before the user's own change can be applied.

Example: Two Users Accessing the Same Record

User A: Step 1

PERSONNEL							
Id	Last Name	First Name	Start Date	Title	Dept Id	Salary	
11	Magee	Colin	14-MAY-90	Sales Representat	31	1400	
12	Giljum	Henry	18-JAN-92	Sales Representat	32	1490	
13	Sedeghi	Yasmin	18-FEB-91	Sales Representat	33	1515	
14	Nguyen	Mai	22-JAN-92	Sales Representat	34	1525	
15	Dumas	Andre	09-OCT-91	Sales Representat	31	1450	

D - 6

Copyright © 2009, Oracle. All rights reserved.

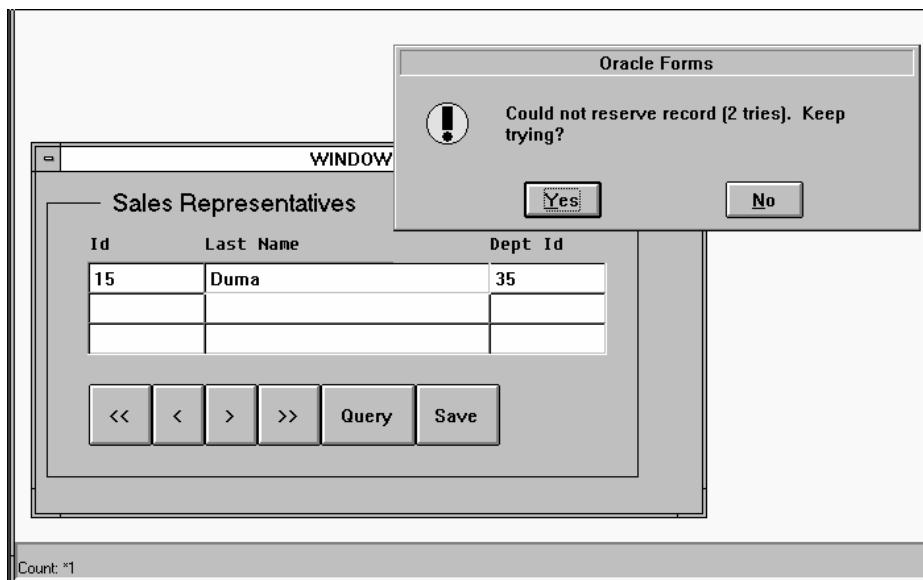
ORACLE®

Example: Two Users Accessing the Same Record

1. User A is running the Personnel application and queries the sales representatives. The record for employee 15, Dumas, is updated so that his department is changed to 31. User A does not save the change at this point in time. The row that corresponds to the changed record is now locked (exclusively).

Example: Two Users Accessing the Same Record

User B: Step 2



ORACLE®

D - 7

Copyright © 2009, Oracle. All rights reserved.

Example: Two Users Accessing the Same Record (continued)

2. User B is running the same application and has started a form that accesses the sales representatives. Employee Dumas still appears in department 35 in this form because User A has not yet saved the change to the database.
User B attempts to update the record by changing the sales representative's name to Duma. Because this action requests a lock on the row, and this row is already locked by User A, Forms issues an alert saying the attempt to reserve the record failed. The alert asks if the user wants to keep trying. This user can request additional attempts or reply No and try later.
User B replies No. This results in the fatal error message 40501, which confirms that the original update action has failed.

Example: Two Users Accessing the Same Record

User A: Step 3

PERSONNEL					
Id	Last Name	First Name	Start Date	Title	Dept Id
11	Magee	Colin	14-MAY-90	Sales Representative	31
12	Giljum	Henry	18-JAN-92	Sales Representative	32
13	Sedeghi	Yasmin	18-FEB-91	Sales Representative	33
14	Nguyen	Mai	22-JAN-92	Sales Representative	34
15	Dumas	Andre	09-OCT-91	Sales Representative	31

FRM-40400: Transaction complete: 1 records applied and saved.
Count: *5

ORACLE®

D - 8

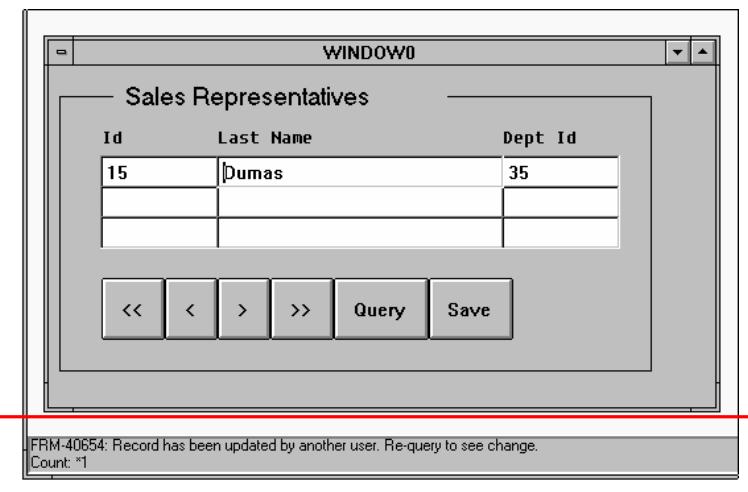
Copyright © 2009, Oracle. All rights reserved.

Example: Two Users Accessing the Same Record (continued)

3. Back in the Personnel form, User A now saves the change to Dumas' department, which applies the change to the database row and then releases the lock at the end of the transaction.

Example: Two Users Accessing the Same Record

User B: Step 4



ORACLE®

D - 9

Copyright © 2009, Oracle. All rights reserved.

Example: Two Users Accessing the Same Record (continued)

4. In the Sales Representatives form, User B can now alter the record. However, because the database row itself has now changed since it was queried in this form, Forms tells User B that it must be requeried before a change can be made. The screenshot shows the message: "FRM-40654: Record has been updated by another user. Re-Query to see change." After User B requeries, the record can then be changed.

Locking in Triggers

Achieved by:

- SQL data manipulation language
- SQL explicit locking statements
- Built-in subprograms
- Data manipulation language (DML) statements



Locking in Triggers

In addition to the default locking described earlier, database locks can occur in Forms applications because of actions that you include in triggers. These can be:

- **SQL DML:** Sometimes you may need to execute INSERT, UPDATE, and DELETE statements, which add to those that Forms does during the saving process (posting and committing). These trigger SQL commands cause implicit locks on the tables and rows that they affect.
- **SQL locking statements:** You can explicitly issue locks from a trigger by using the SELECT . . . FOR UPDATE statement. The LOCK TABLE statement is also allowed, though rarely necessary.
- **Built-in subprograms:** Certain built-ins allow you to explicitly lock rows that correspond to the current record (LOCK_RECORD) or to records fetched on a query (ENTER_QUERY and EXECUTE_QUERY).

To keep locking duration to a minimum, DML statements should be used only in transactional triggers. These triggers fire during the process of applying and saving the user's changes, just before the end of a transaction when locks are released.

Locking in Triggers (continued)

Locking by DML Statements

If you include DML statements in transactional triggers, their execution causes:

- Row exclusive lock on the affected table
- Exclusive lock on the affected rows

Because locks are not released until the end of the transaction, when all changes have been applied, it is advantageous to code DML statements as efficiently as possible, so that their actions are completed quickly.

Locking with Built-ins

- ENTER_QUERY (FOR_UPDATE)
- EXECUTE_QUERY (FOR_UPDATE)

ORACLE®

D - 12

Copyright © 2009, Oracle. All rights reserved.

Locking with Built-ins

Forms maintains a hidden item called ROWID in each base-table block. This item stores the ROWID value for the corresponding row of each record. Updates or deletes in triggers that apply to such rows can identify them most efficiently using this value, as in the following example:

```
UPDATE orders
      SET order_date = SYSDATE
      WHERE ROWID = :orders.rowid;
```

Locking with Built-in Subprograms

The following built-ins allow locking:

- **EXECUTE_QUERY (FOR_UPDATE) and ENTER_QUERY (FOR_UPDATE):** When called with the FOR_UPDATE option, these built-ins exclusively lock the rows fetched for their query. Care should be taken when reserving rows in this way because large queries cause locking on many rows.
- **LOCK_RECORD:** This built-in locks the row that corresponds to the current record in the form. This is typically used in an On-Lock trigger, where it has the same effect as Forms's default locking.

On-Lock Trigger

Example:

```
IF USER = 'MANAGER' THEN  
    LOCK_RECORD;  
ELSE  
    MESSAGE('You are not authorized to change  
records here');  
    RAISE form_trigger_failure;  
END IF;
```



On-Lock Trigger

This block-level trigger replaces the default locking that Forms normally carries out, typically when the user updates or deletes a record in a base-table block. The trigger fires before the change to the record is displayed. On failure, the input focus is set on the current item.

Use this trigger to:

- Bypass locking on a single-user system, therefore, speeding processing
- Conditionally lock the record or fail the trigger (Failing the trigger effectively fails the user's action.)
- Handle locking when directly accessing non-Oracle data sources

If this trigger succeeds, but its action does not lock the record, the row remains unlocked after the user's update or delete operation. Use the LOCK_RECORD built-in within the trigger if locking is not to be bypassed.

On-Lock Trigger (continued)

Example

The following On-Lock trigger on the block Stock only permits the user MANAGER to lock records for update or delete:

```
IF USER = 'MANAGER' THEN
    LOCK_RECORD;
    IF NOT FORM_SUCCESS THEN
        RAISE form_trigger_failure;
    END IF;
ELSE
    MESSAGE('You are not authorized to change records
here');
    RAISE form_trigger_failure;
END IF;
```

O

Oracle Object Features

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Oracle Scalar Data Types

- Automatically converted:
 - FLOAT
 - NLS types
 - NCHAR
 - NVARCHAR2
- Unsupported:
 - Time stamp
 - Interval

ORACLE®

E - 2

Copyright © 2009, Oracle. All rights reserved.

Oracle Data Types

In Oracle Forms Builder, these data types are automatically converted to existing Forms item data types. Three new scalar data types were introduced with Oracle8i:

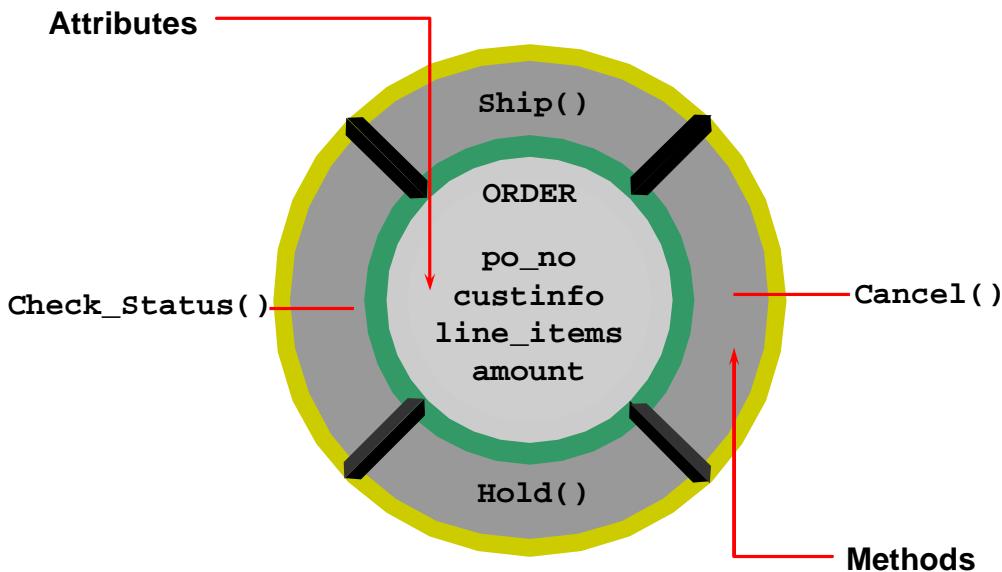
- NCHAR stores fixed-length (blank-padded if necessary) National Language Support (NLS) character data.
- NVARCHAR2 stores variable-length NLS character data.
- FLOAT is a subtype of NUMBER. However, you cannot specify a scale for FLOAT variables. You can specify only a binary precision, which is the total number of binary digits.

NLS Types in Oracle: In the Oracle database, data stored in columns of NCHAR or NVARCHAR2 data types are exclusively stored in Unicode, regardless of the database character set. This enables users to store Unicode in a database that may not use Unicode as the database character set. The Oracle database supports two Unicode encodings for the Unicode data types: AL16UTF16 and UTF8.

With NLS, number and date formats adapt automatically to the language conventions specified for a user session. Users around the world can interact with the Oracle server in their native languages. NLS is discussed in *Oracle Database Globalization Support Guide*.

Unsupported: Time stamp and interval data types (new with Oracle9i). You cannot use the Data Block Wizard to create a block when columns are of these data types. As an alternative, you can create blocks manually and then create DATETIME items referencing these database columns.

Object Types



ORACLE®

E - 3

Copyright © 2009, Oracle. All rights reserved.

Object Types

An object type is a user-defined composite data type. Oracle requires enough knowledge of a user-defined data type to interact with it. Thus, an object type can be similar to a record type and also to a package.

An object type must have one or more attributes and can contain methods. Interaction with attributes should be accomplished via the methods, as illustrated by the graphic in the slide. The order attributes are updated only through the `Cancel()`, `Ship()`, `Check_Status()`, and `Hold()` methods.

Attributes: An object type is similar to a record type in that it is declared to be composed of one or more subparts that are of predefined data types. Record types call these subparts *fields*, but object types call them *attributes*. Attributes define the object structure.

```
CREATE TYPE address_type AS OBJECT
  (address  VARCHAR2(30),
   city     VARCHAR2(15),
   state    CHAR(2),
   zip      CHAR(5));
CREATE TYPE phone_type AS OBJECT
  (country   NUMBER(2),
   area     NUMBER(4),
   phone    NUMBER(9));
```

Object Types (continued)

Just as the fields of a record type can be of other record types, the attributes of an object type can be of other object types. Such an object type is called *nested*.

```
CREATE TYPE address_and_phone_type AS OBJECT
  (address    address_type,
   phone      phone_type);
```

Object types are like record types in another sense: Both of them must be declared as types before the actual object or record can be declared.

Methods: An object type is also similar to a package. After an object is declared, its attributes are similar to package variables. Like packages, object types can contain procedures and functions. In object types, these subprograms are known as *methods*. A method describes the behavior of an object type.

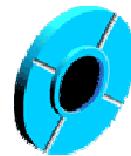
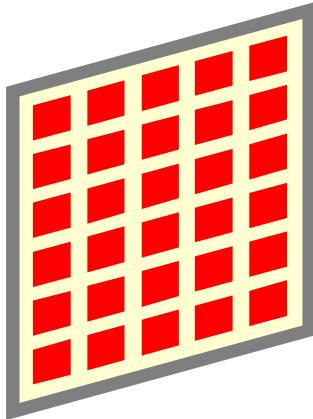
Like packages, object types can be declared in two parts: a specification and a body. As with package variables, attributes declared in the object type specification are public and those declared in the body are private. As with package subprograms, all methods are defined in the package body, but only those whose specification appears in the object type specification are public methods.

The following is an example of an object type:

```
CREATE TYPE dept_type AS OBJECT
  (dept_id      NUMBER(2),
   dname        VARCHAR2(14),
   loc          VARCHAR2(3),
   MEMBER PROCEDURE set_dept_id (d_id NUMBER),
   PRAGMA RESTRICT_REFERENCES (set_dept_id,
                                 RNDS,WNDS,RNPS,WNPS),
   MEMBER FUNCTION get_dept_id RETURN NUMBER,
   PRAGMA RESTRICT_REFERENCES (get_dept_id,
                                 RNDS,WNDS,RNPS,WNPS));
CREATE TYPE BODY dept_type AS
  MEMBER PROCEDURE set_dept_id (d_id NUMBER)
  IS
  BEGIN
    dept_id := d_id;
  END;
  MEMBER FUNCTION get_dept_id
  RETURN NUMBER
  IS
  BEGIN
    RETURN (dept_id);
  END;
END; \
```

Creating Oracle Objects: After you have declared an object type, you can create objects based on the type. Object types are not themselves objects; they are only blueprints for objects. The term “object” can be confusing because Oracle uses the term to refer to constructs within the database—for example, tables, views, procedures, and so on. This is a completely different usage.

Object Tables



Object table based on object type

ORACLE®

E - 5

Copyright © 2009, Oracle. All rights reserved.

Object Tables

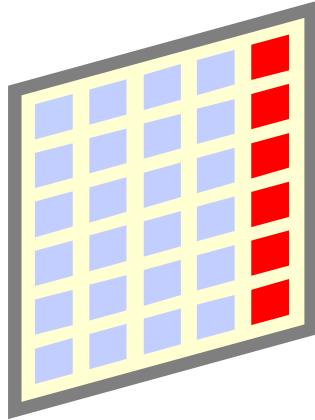
You can create an object by creating a table whose rows are objects of that object type. This type of table is illustrated by the graphic at the left of the slide, while the graphic at the right is the symbol for an object. Here is an example of an object table declaration:

```
CREATE TABLE o_dept OF dept_type;
```

SQL and PL/SQL treat object tables very similarly to relational tables, with the attribute of the object corresponding to the columns of the table. But there are significant differences. The most important difference is that rows in an object table are assigned object IDs (OIDs) and can be referenced using a REF type.

Note: REF types are discussed later.

Object Columns



Object column based on object type

ORACLE®

E - 6

Copyright © 2009, Oracle. All rights reserved.

Object Columns

Another construct that can be based on an object type is an object column in a relational table. Here is an example of a relational table creation statement with an object column:

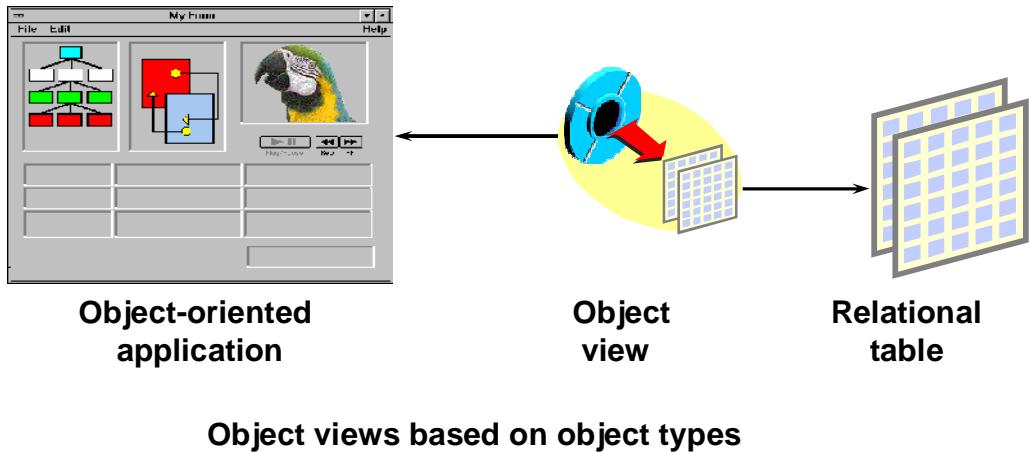
```
CREATE TABLE o_customer (
    custid      NUMBER (6) NOT NULL,
    name        VARCHAR2 (45),
    repid       NUMBER (4) NOT NULL,
    creditlimit NUMBER (9,2),
    address     address_type,
    phone       phone_type);
```

In the object table, the rows of a table are objects. In a relational table with an object column, illustrated by the graphic at the left of the slide, the column is an object. The table will usually have standard columns, as well as one or more object columns.

Object columns are not assigned OIDs and, therefore, cannot be referenced using object REF values.

Note: Object REFs are discussed later in this lesson.

Object Views



ORACLE®

E - 7

Copyright © 2009, Oracle. All rights reserved.

Object Views

Often, the most difficult part of adopting a new technology is the conversion process itself. For example, a large enterprise might have several applications accessing the same data stored in relational tables. If such an enterprise decided to start using object-relational technology, the enterprise would not convert all of the applications at once. It would convert the applications one at a time.

That presents a problem. The applications that have been converted need the data stored as objects, whereas the applications that have not been converted need the data stored in relational tables.

This dilemma is addressed by object views. Like all views, an object view transforms the way a table appears to a user, without changing the actual structure of the table. Object views make relational tables look like object tables. This allows the developers to postpone converting the data from relational structures to object-relational structures until after all of the applications have been converted. During the conversion process, the object-relational applications can operate against the object view; the relational applications can continue to operate against the relational tables.

Objects accessed through object views are assigned OIDs, and can be referenced using object REFs.

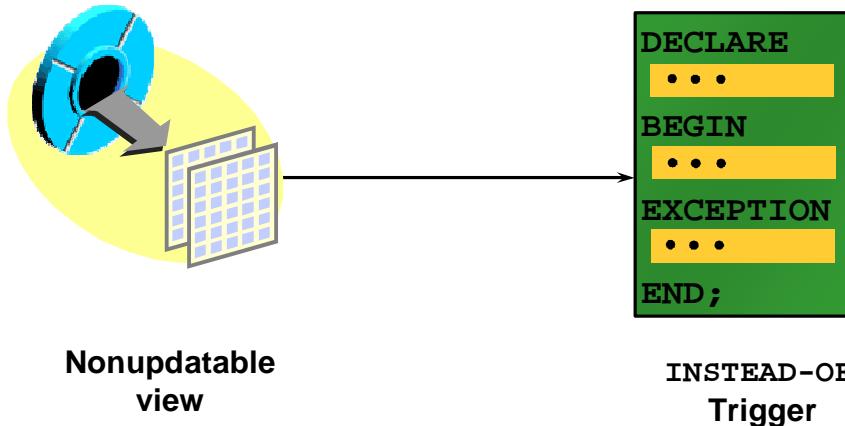
Object Views (continued)

The following is an example of an object view creation statement:

```
CREATE VIEW emp_view OF emp_type
  WITH OBJECT OID (eno)
  AS
    SELECT          e.empno, e.ename, e.sal, e.job
    FROM           emp e;
```

The graphic in the slide shows an object-oriented application at the left and a set of relational tables at the right. In the middle of the slide, the object view enables data stored in the relational tables to be used by the object-oriented application.

Using INSTEAD-OF Triggers



ORACLE®

E - 9

Copyright © 2009, Oracle. All rights reserved.

Using INSTEAD-OF Triggers

INSTEAD-OF Triggers: INSTEAD-OF triggers provide a transparent way of modifying views that cannot be modified directly through SQL DML statements (INSERT, UPDATE, and DELETE).

These triggers are called INSTEAD-OF triggers because, unlike other types of triggers, the Oracle server fires the trigger instead of executing the triggering statement. The trigger performs update, insert, or delete operations directly on the underlying tables.

Users write normal INSERT, DELETE, and UPDATE statements against the view, and the INSTEAD-OF trigger works invisibly in the background to make the right actions take place.

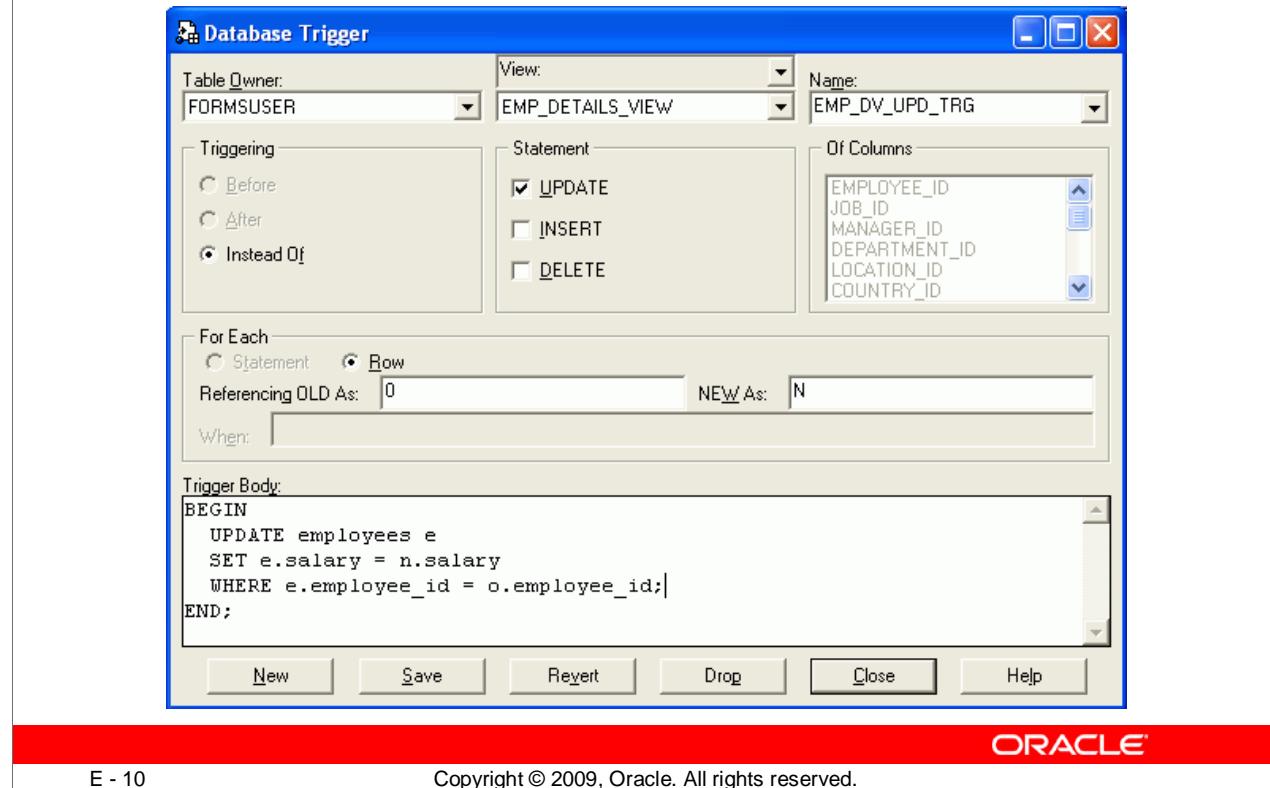
INSTEAD-OF triggers are activated for each row.

The graphic at the left of the slide illustrate a nonupdatable object view. Updates to the underlying tables of this view are performed by the INSTEAD-OF trigger represented by the graphic to the right of the slide.

Reference: For more information about INSTEAD-OF triggers, see *Oracle Application Developer's Guide—Fundamentals* or *Oracle Database Concepts*.

Note: Although INSTEAD-OF triggers can be used with any view, they are typically needed with object views.

Defining INSTEAD-OF Triggers



E - 10

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Defining INSTEAD-OF Triggers

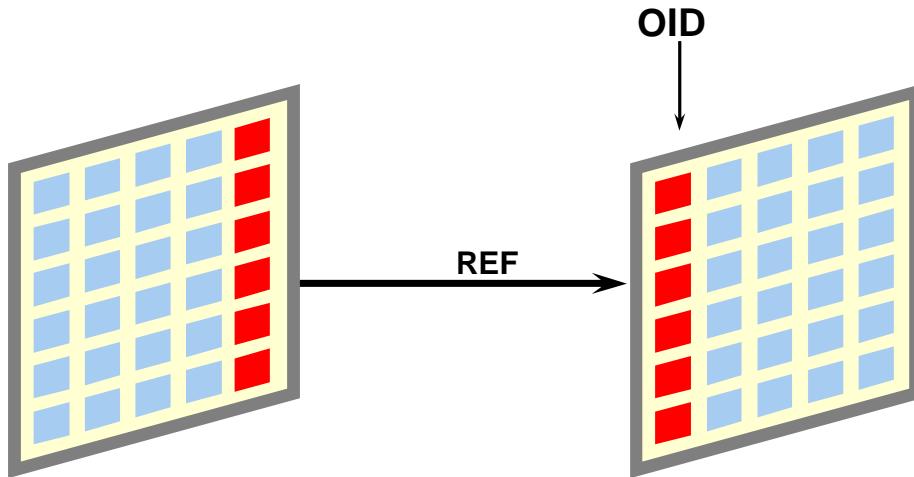
You create INSTEAD-OF database triggers through the trigger creation dialog box, just like any other database trigger. The screenshot shows the Database Trigger dialog box, with pop-up lists, option buttons, and check boxes to specify table owner, view, trigger name, triggering event (Before, After, or Instead Of), Statement (UPDATE, INSERT, and/or DELETE), column if applicable, for each statement or row, reference for old and new, when, and finally, the trigger body. In this example, the definition is for the EMP_DV_UPD_TRG trigger on the EMP_DETAILS_VIEW view in the FORMSUSER schema. It is an INSTEAD-OF-UPDATE trigger on the row, and reference old as O and new as N. The trigger body is:

```
BEGIN
  UPDATE employees e
  SET e.salary = n.salary
  WHERE e.employee_id = o.employee_id;
END;
```

INSTEAD-OF INSERT, UPDATE, and DELETE triggers enable you to directly insert, update, and delete against object views. They can also be used with any other type of view that does not allow direct DML.

When a view has an INSTEAD-OF trigger, the code in the trigger is executed in place of the triggering DML code.

References to Objects



ORACLE®

E - 11

Copyright © 2009, Oracle. All rights reserved.

Referencing Objects

Introduction

In relational databases, primary key values are used to uniquely identify records. In object-relational databases, OIDs provide an alternative method.

When a row in an object table or object view is created, it is automatically assigned a unique identifier called an object ID (OID).

Object REFs

With relational tables, you can associate two records by storing the primary key of one record in one of the columns (the foreign key column) of another.

In a similar way, you can associate a row in a relational table to an object by storing the OID of an object in a column of a relational table.

You can associate two objects by storing the OID of one object in an attribute of another. The stored copy of the OID then becomes a pointer, or reference (REF) to the original object. The attribute or column that holds the OID is of the REF data type.

In the slide, the relational table represented by the graphic at the left has one column that stores OIDs of the objects in the object table depicted at the right of the slide.

Referencing Objects (continued)

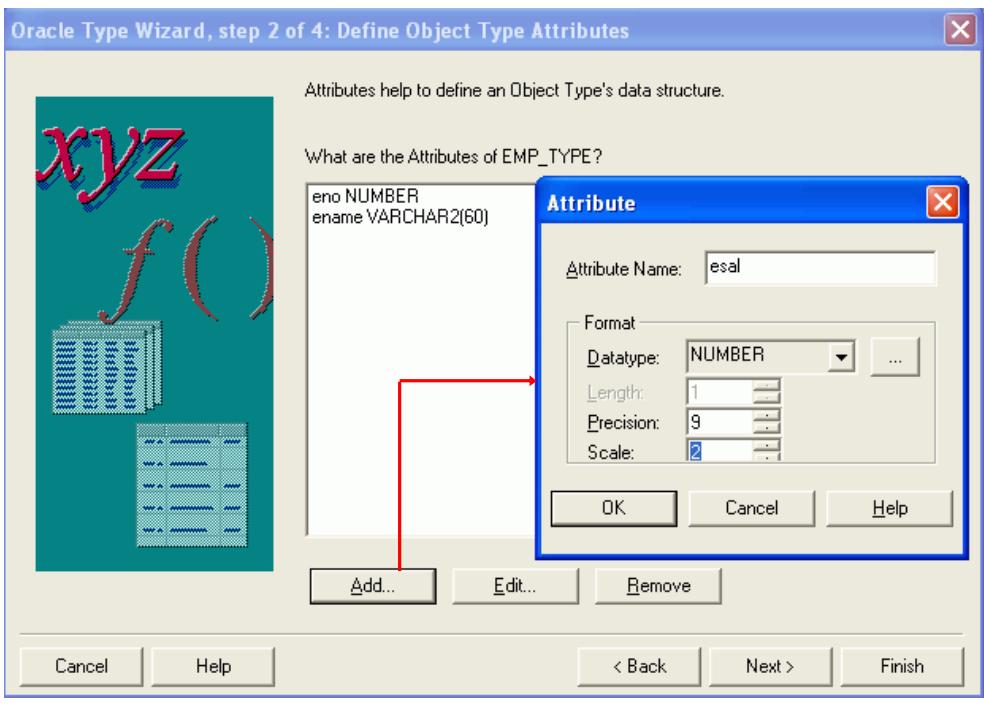
Note: Object columns are not assigned OIDs and cannot be pointed to by a REF.

The following is an example of a table declaration that includes a column with a REF data type:

```
CREATE TABLE o_emp
  ( empno          NUMBER( 4 )  NOT NULL,
    ename          VARCHAR2(10),
    job            VARCHAR2(10),
    mgr            NUMBER( 4 ),
    hiredate       DATE,
    sal             NUMBER( 7 , 2 ),
    comm            NUMBER( 7 , 2 ),
    dept           REF dept_type SCOPE IS o_dept) ;
```

Note: The REF is scoped here to restrict the reference to a single table, O_DEPT. The object itself is not stored in the table, only the OID value for the object is stored.

Oracle Type Wizard



E - 13

Copyright © 2009, Oracle. All rights reserved.

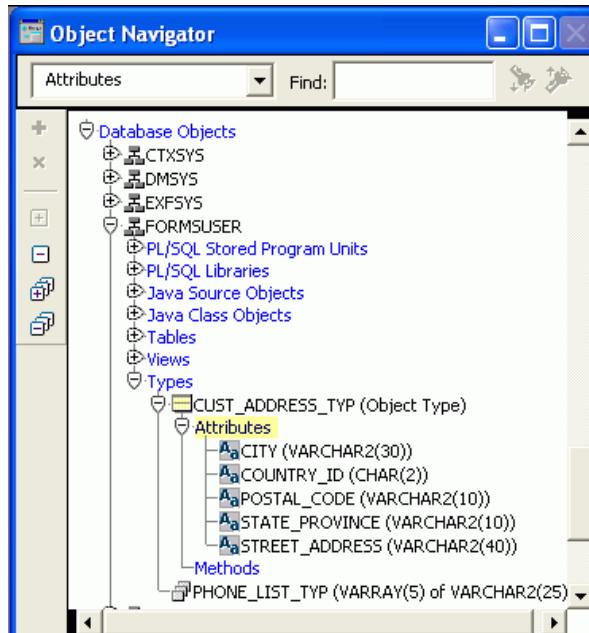
ORACLE®

Oracle Type Wizard

You can create object types by using the Oracle Type Wizard. The wizard enables you to define the attributes and methods. You invoke the Oracle Type Wizard from Forms Builder by selecting the Types node in a schema under Database Objects, and then by clicking the Create icon.

The screenshot shows one step in the Oracle Type Wizard that enables adding, editing, or removing attributes to the object. When the Add button is clicked, a dialog box enables you to name the attribute and specify its data type, length, precision, and scale.

Displaying Object Types in the Object Navigator



ORACLE®

Displaying Object Types in the Object Navigator

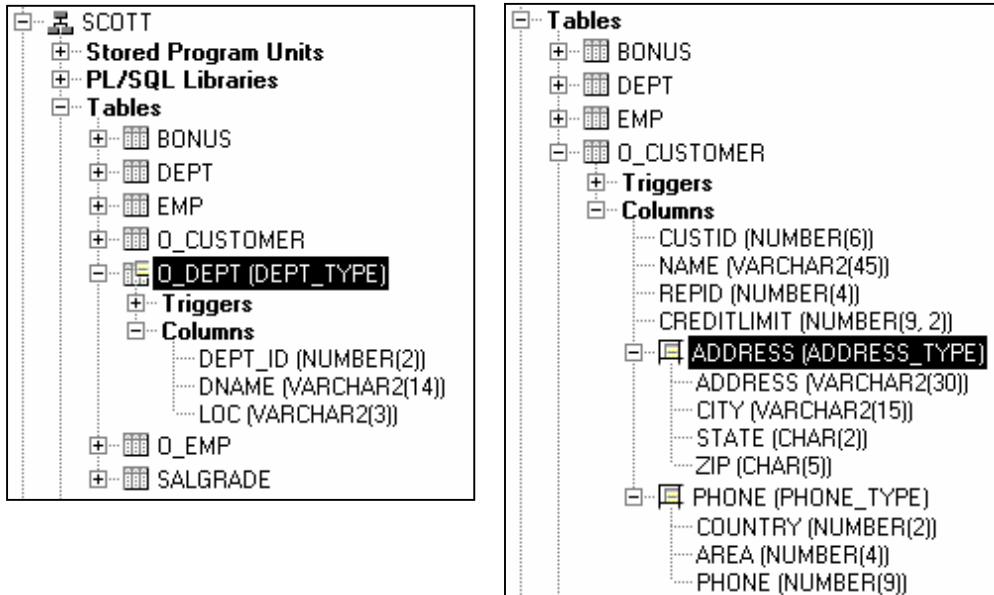
The Object Navigator lists declared types in the Database Objects section, along with tables, views, and other Oracle objects.

Both the attributes and the methods are listed under each type. Also, the nested types within a type are displayed in an indented sublevel.

This convention is used for nested object and object type displays throughout Oracle Developer.

The screenshot in the slide shows the CUST_ADDRESS_TYP (Object Type) node under the Database Objects > Schema_name > Types nodes in the Object Navigator. Subnodes include Attributes and Methods. There are no methods in this example, so that node is empty, but several attributes are shown: CITY, COUNTRY_ID, POSTAL_CODE, STATE_PROVINCE, and STREET_ADDRESS.

Displaying Object Tables and Columns in the Object Navigator



ORACLE®

Displaying Object Tables and Columns in the Object Navigator

Object Tables

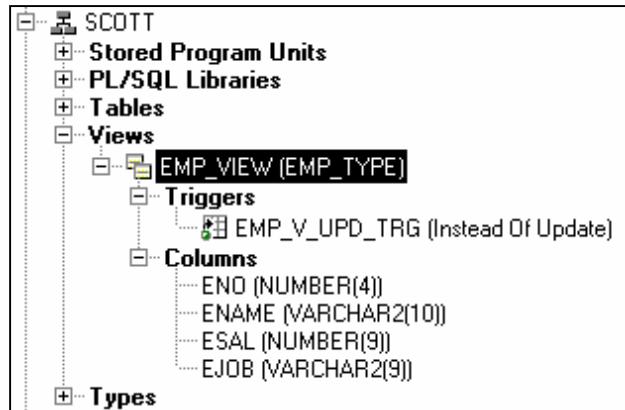
Object tables are displayed like relational tables, with the attributes of the object displayed like columns in a relational table. Also, the object table type name is displayed in parentheses after the name of the object table. The screenshot at the left of the slide shows an object table node of O_DEPT (DEPT_TYPE), and its Columns node shows the object attributes of DEPT_ID, DNAME, and LOC.

Object Columns

Object columns are displayed with the object type in parentheses after the column name and with the attributes of the type indented underneath the column name. The screenshot at the right shows a table named O_CUSTOMER that has six columns, two of which are object columns:

- ADDRESS (ADDRESS_TYPE), which has four attributes: ADDRESS, CITY, STATE, and ZIP
- PHONE (PHONE_TYPE), which has three attributes: COUNTRY, AREA, and PHONE

Displaying Object Views in the Object Navigator



ORACLE®

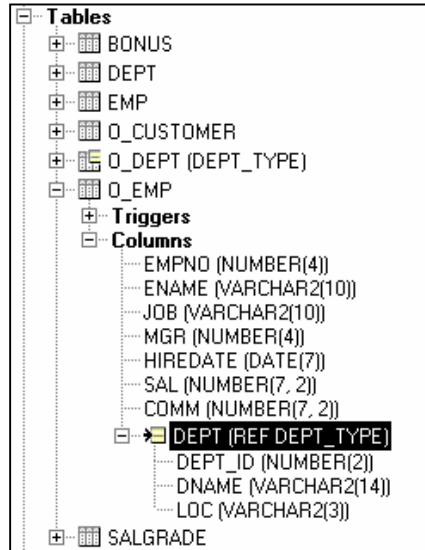
Displaying Object Views in the Object Navigator

Object views are displayed in the same way as any other view, except that the object type on which they are based is written in parentheses after the view name.

The screenshot shows that the EMP_VIEW (EMP_TYPE) object view appears under the Views node for the database schema. This view node contains two subnodes:

- Triggers: EMP_V_UPD_TRG (Instead of Update)
- Columns: ENO, ENAME, ESAL, and EJOB

Displaying Object REFs in the Object Navigator



ORACLE®

Displaying Object REFs in the Object Navigator

Object types that contain attributes of type REF, and relational tables that have columns of type REF, display the keyword REF before the name of the object type that is being referenced.

The attributes of the referenced object type are displayed indented under the column or attribute. In the screenshot in the slide, the O_EMP table is shown in the Object Navigator. Its last column is displayed as the node DEPT (REF DEPT_TYPE), with the attributes of the object displayed under that node (DEPT_ID, DNAME, and LOC).

Using the Layout Editor

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Using the Layout Editor

Common features:

- Moving and resizing objects and text
- Defining colors and fonts
- Importing and manipulating images and drawings
- Creating geometric lines and shapes
- Layout surface: Forms canvas view

ORACLE®

F - 2

Copyright © 2009, Oracle. All rights reserved.

Why Use the Layout Editor?

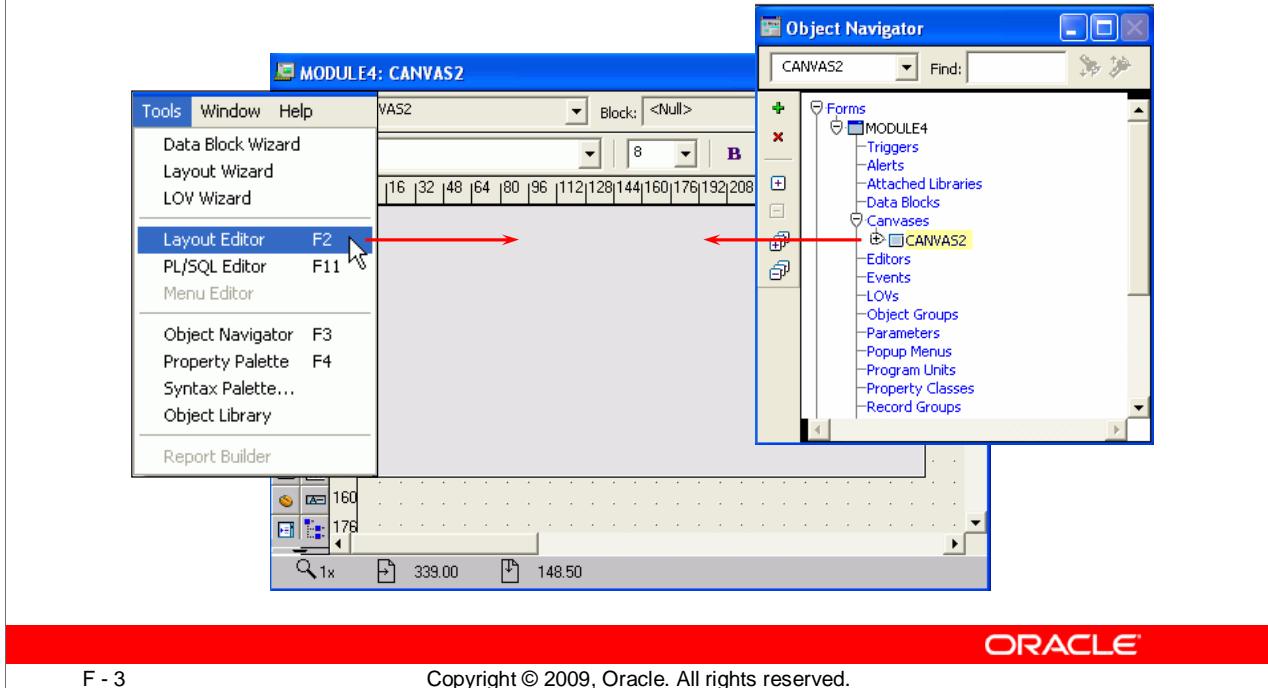
The Layout Editor is a graphical tool for defining the arrangement and appearance of visual objects. The Layout Editor opens windows in Forms Builder to present the surfaces on which you can arrange objects.

The following are the functions of the Layout Editor:

- Moving objects to new positions in the layout and aligning them with each other
- Resizing objects and text
- Defining the colors of text and visual objects
- Creating lines, boxes, and other geometric shapes
- Importing and manipulating images on the layout
- Changing the font style and weight of text
- Accessing the properties of objects that you see in the layout

You can use the Layout Editor to control the visual layout on canvas views in Oracle Forms Builder. A canvas is the surface on which you arrange a form's objects. Its view is the portion of that canvas that is initially visible in a window at run time. You can also see stacked or tabbed canvas views in the Layout Editor; their views might overlay others in the same window. You can also display stacked views in the Layout Editor.

Invoking the Layout Editor



Invoking the Layout Editor

You can invoke the Layout Editor from either the Builder menus or the Object Navigator. This applies whether you are doing so for the first time in a session or at a later stage. If you have minimized an existing Layout Editor window, you can also reacquire it in the way that you would any window.

Opening from the Object Navigator

Double-click a Canvas View icon in a form hierarchy, as shown in the screenshot at the right of the slide.

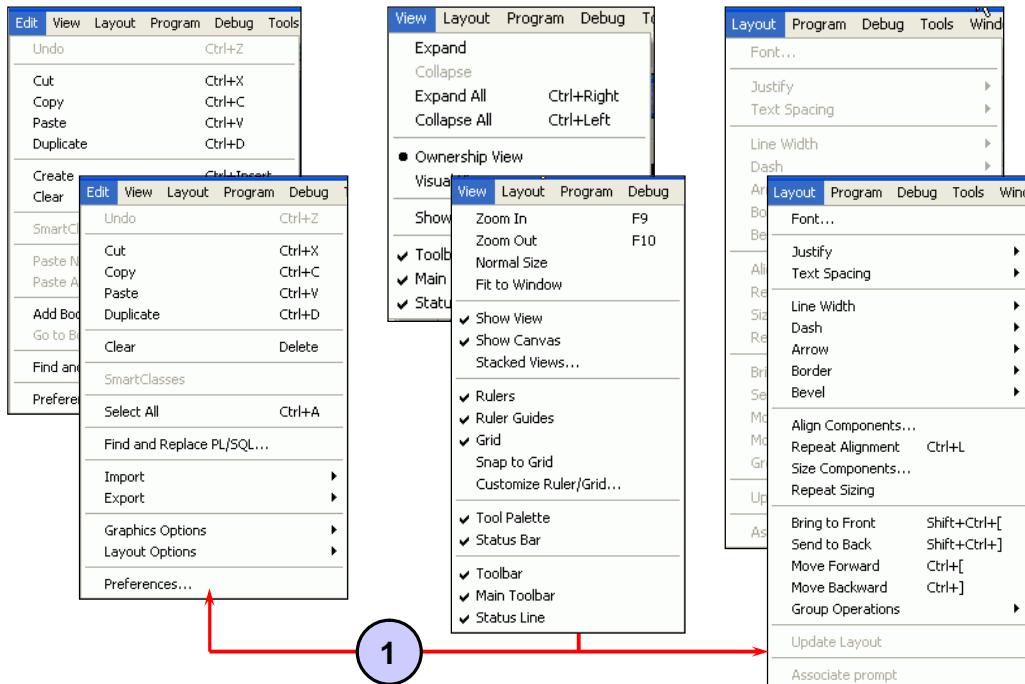
Opening from the Builder Menus

- Make sure that you have a context for a form by selecting the appropriate objects in the Navigator.
- Select Tools > Layout Editor from the Builder menu, as shown in the screenshot at the left of the slide.

Closing the Layout Editor

You can close or minimize the Layout Editor window or windows as you would any window.

Components of the Layout Editor



Components of the Layout Editor

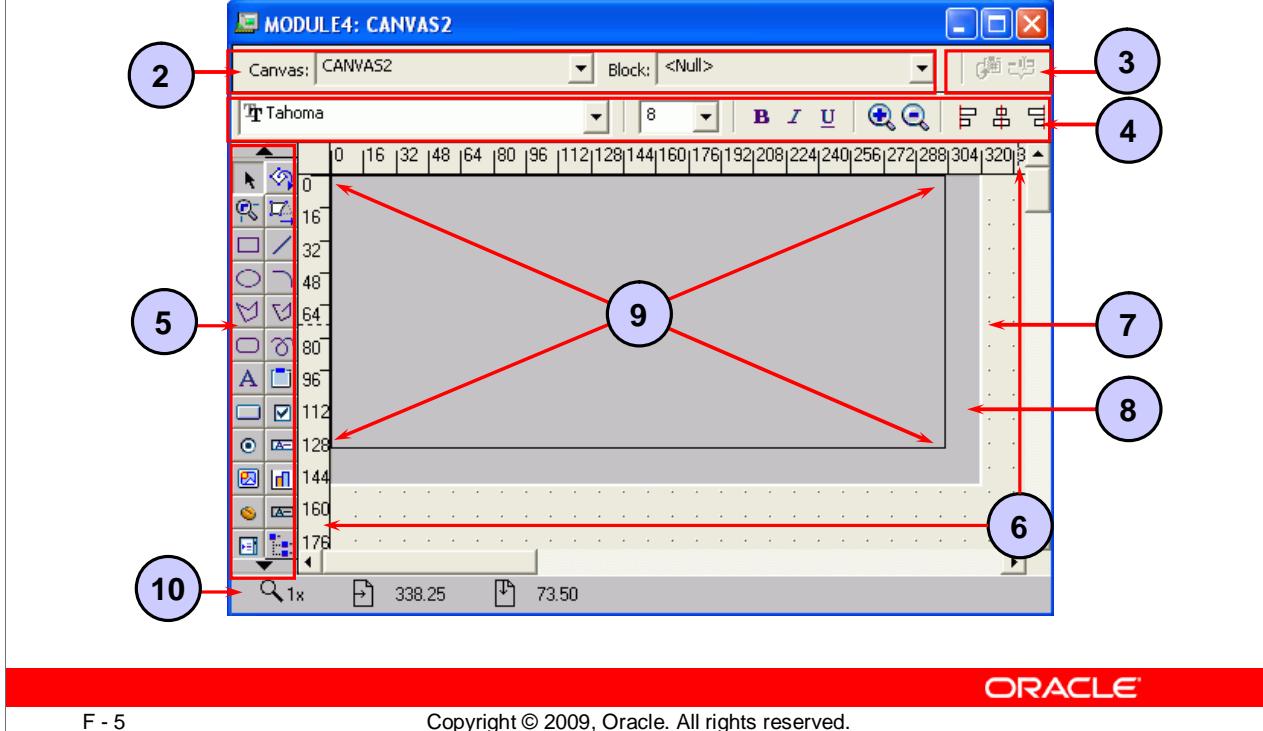
Common Components of the Layout Editor

1. **Menu facilities:** While the Layout Editor window is active, the options available on the Edit, View, and Layout menus change. The new choices are submenus for controlling the Layout Editor.

The slide shows the Edit, View, and Layout menus. The screenshots at the top show what these menus look like when the Object Navigator is active, whereas those at the bottom show the expanded choices when the Layout Editor is active. The added choices are:

- For the Edit menu: Select All, Import, Export, Graphics Options, and Layout Options
- For the View menu: Zoom In, Zoom Out, Normal Size, Fit to Window, Stacked Views, Rulers, Ruler Guides, Grid, Snap to Grid, Customize Ruler/Grid, Tool Palette, Status Bar
- For the Layout menu: All the menu options are disabled when the Object Navigator is active, but when the Layout Editor is active, the applicable options are enabled.

Components of the Layout Editor



Components of the Layout Editor (continued)

2. **Title bar:** Displays the name of the current form, the name of the canvas being edited, and the name of the current block. When you create an item by drawing it on a canvas in the Layout Editor, Forms Builder assigns the item to the current block, as indicated by Layout Editor block context. You can change the Layout Editor block context using the Block pop-up list on the title bar.
3. **Horizontal toolbar:** Contains buttons that enable you to update the layout or associate text with an item prompt
4. **Style bar:** Appears under the title bar and horizontal toolbar. The style bar contains buttons that are a subset of the View and Layout menus.
5. **Vertical toolbar:** Contains the tools for creating and modifying objects on the layout. Note that some tools in the palette may be hidden if you have reduced the size of the Layout Editor window. If so, scroll buttons appear on the vertical toolbar.
There are three types of tools:
 - Graphics tools for creating and modifying lines and shapes
 - Tools for creating specific types of items
 - Manipulation tools for controlling color and patterns

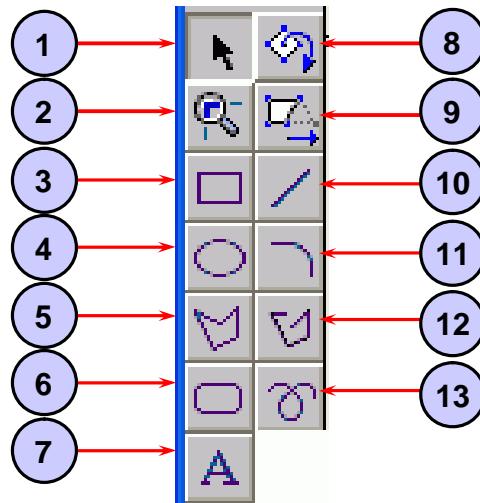
Components of the Layout Editor (continued)

6. **Rulers and ruler guides:** Rulers are horizontal and vertical markers to aid alignment; they appear at the top and side of the layout region. You can switch these off or have their units altered, as required. Drag ruler guides from the rulers across the layout region to mark positions in the layout.
7. **Layout/Painting region:** Is the main area where you can place and manipulate objects. A grid pattern can be displayed in this area to aid alignment of objects. You can switch off or rescale this grid if required. (The grid is hidden in the canvas portion of the layout/painting region if the View > Show Canvas option is switched on.)
8. **Canvas:** Is the portion of the layout/painting area where objects must be placed in order for the form module to successfully compile. If objects are outside this area upon compilation, you will receive an error.
9. **Viewport:** The portion of the canvas that can be seen in the containing window; its height and width are determined by properties of the window rather than by canvas properties.
10. **Status line:** Appears at the bottom of the window. It shows you the mouse position and drag distance (when moving objects), and the current magnification level.

The screenshot in the slide shows the Layout Editor with numbers corresponding to the list.

Creating and Modifying Objects in the Layout

The Layout Editor's Tool Palette



ORACLE®

F - 7

Copyright © 2009, Oracle. All rights reserved.

Creating and Modifying Objects in the Layout

You can perform actions in the Layout Editor by selecting from the vertical toolbar and the Builder menus and by controlling objects directly in the layout region.

Creating Lines and Shapes

Create geometric lines and shapes by selecting from the graphics tools on the vertical toolbar. These include: Rectangles/squares, ellipses/circles, polygons and polylines, lines and arcs, and the freehand tool. The slide shows numbered screenshots of the tools:

1	Select	8	Rotate
2	Magnify	9	Reshape
3	Rectangle	10	Line
4	Ellipse	11	Arc
5	Polygon	12	Polyline
6	Rounded rectangle	13	Freehand
7	Text		

Creating and Modifying Objects in the Layout (continued)

Creating a New Line or Shape

1. Select the required graphic tool from the vertical toolbar with a click. This selects the tool for a single operation on the layout (a double-click causes the tool to remain active for subsequent operations).
2. Position the cursor at the start point for the new object in the layout, and then click and drag to the required size and shape.
3. Release the mouse button.

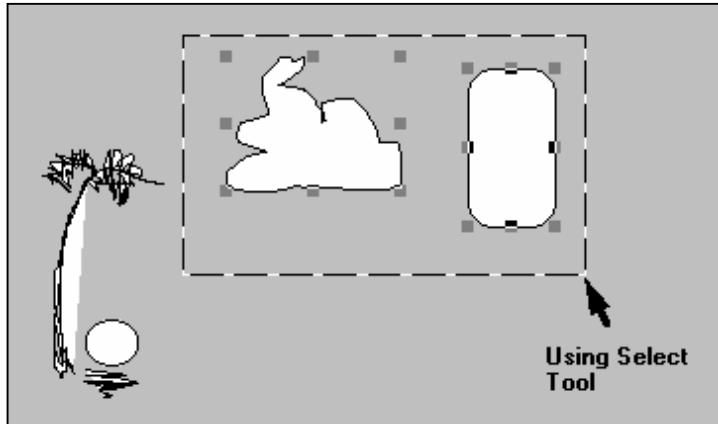
Notice that the object remains selected after this procedure (selection handles appear on its boundaries) until you deselect it by clicking elsewhere.

Note: You can produce constrained shapes (for example, a circle instead of an ellipse) by pressing Shift during step 2.

Creating Text

The Text tool (T) lets you open a Boilerplate Text object on the layout. You can enter one or more lines of text into this object while it is selected with the Text tool. Uses of text objects vary according to the Oracle Developer tool in which you create them.

Selecting Objects



ORACLE®

F - 9

Copyright © 2009, Oracle. All rights reserved.

Selecting Objects for Modification

The default tool in the vertical toolbar is Select, with which you can select and move objects. If the Select tool is active, you can select one or more objects on the layout to move or modify.

To select *one* object, do one of the following:

- Click the object.
- Draw a bounding box around it, as shown in the screenshot.

If the object is small or narrow, it is sometimes easier to use the second method. Also, an object may be transparent (No Fill), which can present a similar problem where it has no center region to select.

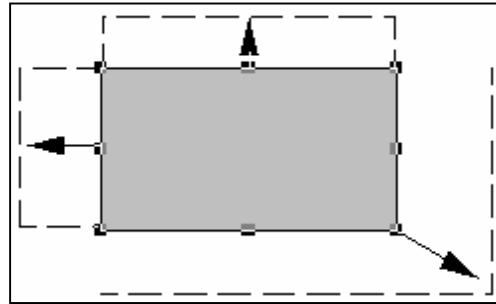
It is convenient to select *several* objects, so that an operation can be performed on them simultaneously.

To select several objects together, do one of the following:

- Press and hold Shift, and then click each object to be selected.
- Draw a bounding box around the objects (providing the objects are adjacent to each other).

Manipulating Objects

Expand/contract
in one direction



Expand/contract
diagonally

ORACLE®

F - 10

Copyright © 2009, Oracle. All rights reserved.

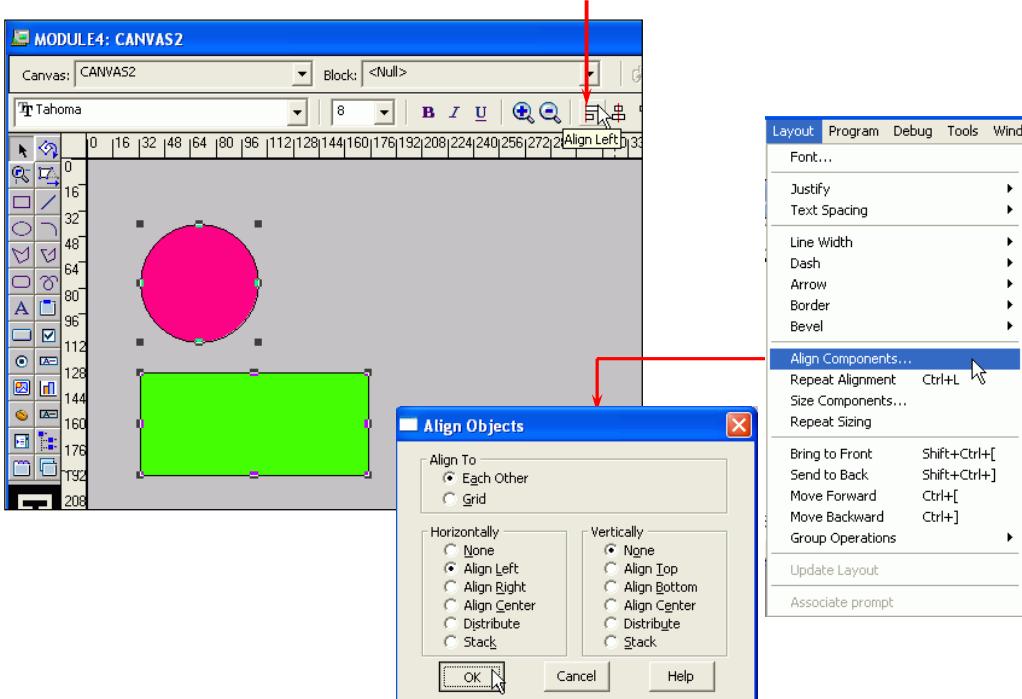
Changing the Size or Ratio

When an object is selected, two types of selection handles are visible, as shown in the graphic in the slide:

- **Corner handles:** Position the cursor on one of these to change the size or ratio of the object diagonally.
- **Midpoint handles:** Position the cursor on one of these to change the size or ratio in a horizontal or vertical direction.

Note: By pressing and holding Shift, you can resize an object without changing its ratios. This means that squares remain as squares, and images do not become distorted when resized.

Moving, Aligning, and Overlapping



ORACLE®

Moving and Aligning Objects

When one or more objects are selected in the layout, you can:

- **Move them to a new location:** Do this by dragging to the required position. You can use the grid and ruler lines to help you position them properly.
- **Align the objects with each other:** Objects can be aligned with the leftmost, rightmost, highest, or lowest object selected. They can also be centered and aligned with the grid. You can do this using the Alignment feature in the Layout menu:
Select Layout > Align Components, and then set the options required in the Align Objects dialog box. You can also use the Align icons on the horizontal toolbar. The graphics in the slide show selecting two objects on the Layout Editor and then selecting to align them with each other horizontally. You can align objects horizontally at the left, right, or center, or vertically at the top, bottom, or center. You can also distribute the objects horizontally or vertically or stack them.

Grid-Snap Alignment: You can ensure that all objects that you move align with snap points that are defined on the grid. To activate these, select View > "Snap to Grid" from the menu. You can also use the View options to change the grid-snap spacing and units.

Note: If you position an object using one grid-snap spacing, and then try to position other objects under different settings, it may prove difficult to align them with each other. Try to stick to the same snap units, if you use grid-snap at all.

Moving and Aligning Objects (continued)

Overlapping Objects

You can position objects on top of each other. If they are transparent, one object can be seen through another (this is explained in detail later in this lesson).

Change the stacking order of overlapping objects by selecting the object to move and then choosing the following as required, from the Layout menu:

- “Bring to Front”
- “Send to Back”
- Move Forward
- Move Backward

Groups in the Layout

- Groups allow several objects to be repeatedly treated as one.
- Groups can be colored, moved, or resized.
- Tool-specific operations exist for groups.
- Groups have a single set of selection handles.
- Members can be added or removed.

ORACLE®

F - 13

Copyright © 2009, Oracle. All rights reserved.

Manipulating Objects As a Group

Sometimes, you want to group objects together in the layout so that they behave as a single object.

Placing Objects into a Group

1. Select the objects in the layout that are to be grouped together.
2. Select Layout > Group Operations > Group from the menu.

You will notice there is now a single set of selection handles for the group, which you can treat as a single object whenever you select a member within it. The Layout menu also gives you options to add and remove members. Resizing, moving, coloring, and other operations will now apply to the group.

Manipulating Individual Group Members

To manipulate group members individually, select the group and then click the individual member. You can use options in the Group Operations menu to remove objects from the group or to reselect the parent group.

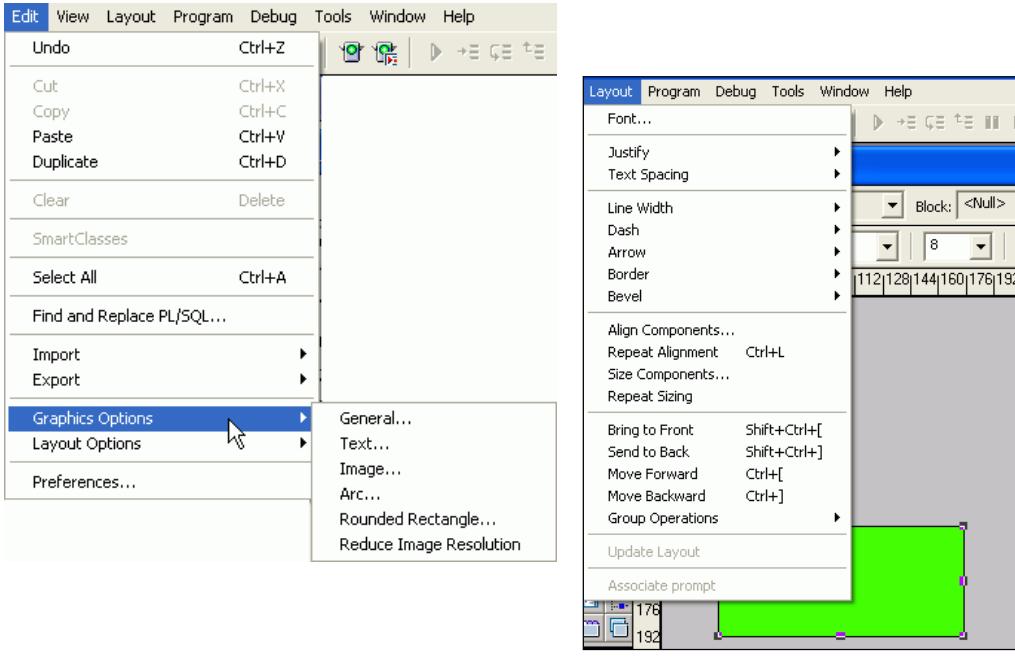
Manipulating Objects As a Group (continued)

Other Tools for Manipulating Objects

- **Rotate:** You can rotate a line or shape through an angle, using the tool's selection handles.
- **Reshape:** You can change size/ratio of a shape that has been rotated or change the sweep angle of an arc. You can also reshape a polygon or polyline.
- **Magnify:** You can increase magnification when you click at a desired zoom position on the layout region. Pressing and holding Shift while you click, reduces magnification.

Note: You can undo your previous action in the current Layout Editor session by selecting Edit > Undo from the menu.

Formatting Objects in the Layout



Formatting Objects in the Layout

The Builder's Layout and Edit menus, shown in the screenshots in the slide, provide a variety of facilities for changing the style and appearance of objects in the layout. These include:

- Font sizes and styles
- Spacing in lines of text
- Alignment of text in a text object
- Line thickness and dashing of lines
- Bevel (3D) effects on objects
- General drawing options (for example, style of curves and corners)

Whichever formatting option you intend to use, first select the objects that you intend to change on the layout, and then choose the necessary option from the menu.

Some of the format options are available on the style bar.

Formatting Objects in the Layout (continued)

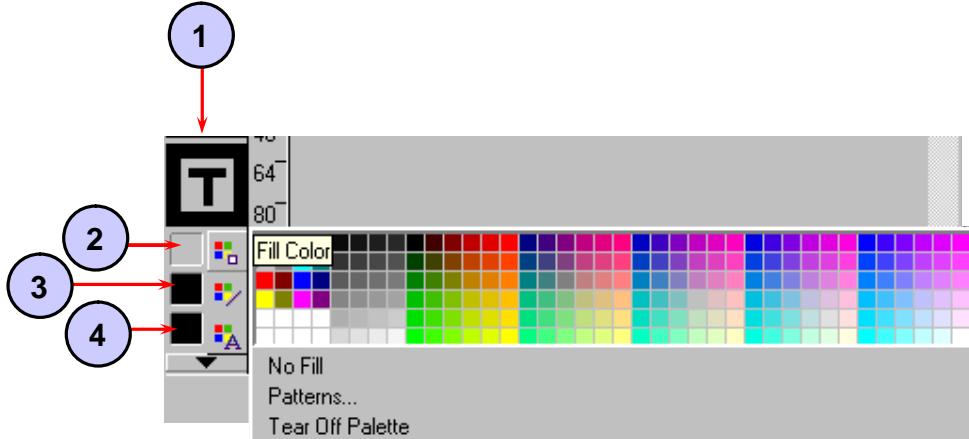
Changing Fonts on Textual Objects

There are a number of ways to change the font characteristics of textual objects. The common steps provided by the Layout Editor are the following:

- Select the objects in the layout whose content text you want to change. (These may be boilerplate text objects and other textual object types.)
- Select the font style and size that you require from the style bar or select Layout from the Builder menu, and then select the font style and size that you require.

Note: In Windows, selecting Font from the menu opens the standard Windows Font dialog box. In other GUI environments, the font choices may appear in the Layout menu itself. Font settings are ignored if the application is run in character mode.

Coloring Objects and Text



ORACLE®

F - 17

Copyright © 2009, Oracle. All rights reserved.

Coloring Objects and Text

The vertical toolbar contains tools for coloring objects. These are numbered in the screenshot above as follows:

1. **Sample Window:** This shows the fill, line, and text color of the currently selected object. If no object is selected, it shows the settings for new objects.
2. **Fill Color:** Use this tool to define the colors and pattern for an object's body.
3. **Line Color:** Use this tool to define the color of a line or the boundary line around an object.
4. **Text Color:** Use this tool to choose the color for the text.

Coloring Objects

You can separately color the fill area (body) of an object and its boundary line. (A line object has no body.)

Coloring a Line

- With the desired layout objects selected, click the Line Color tool. The color palette opens.
- Select the color for lines and bounding lines from this color palette. Select No Line at the bottom of this window to remove the objects' boundary lines.

Note: If you set both No Fill and No Line, the affected objects become invisible.

Coloring Objects and Text (continued)

Coloring an Area

1. Select the objects whose color you want to change.
2. Click the Fill Color tool. The color palette appears, as shown in the screenshot.
If you want the objects to become transparent, select No Fill at the bottom of the color palette.
3. Select a color from the color palette. If you want the fill area to be patterned instead of plain, select Patterns from the bottom of the color palette, and perform the next step.
4. If you select Patterns, another window appears with patterns for you to choose. Select one. You can define separate colors for the two shades of the pattern by clicking the pattern color buttons at the bottom of the Fill Pattern palette; each button opens a further color palette.

Coloring Text

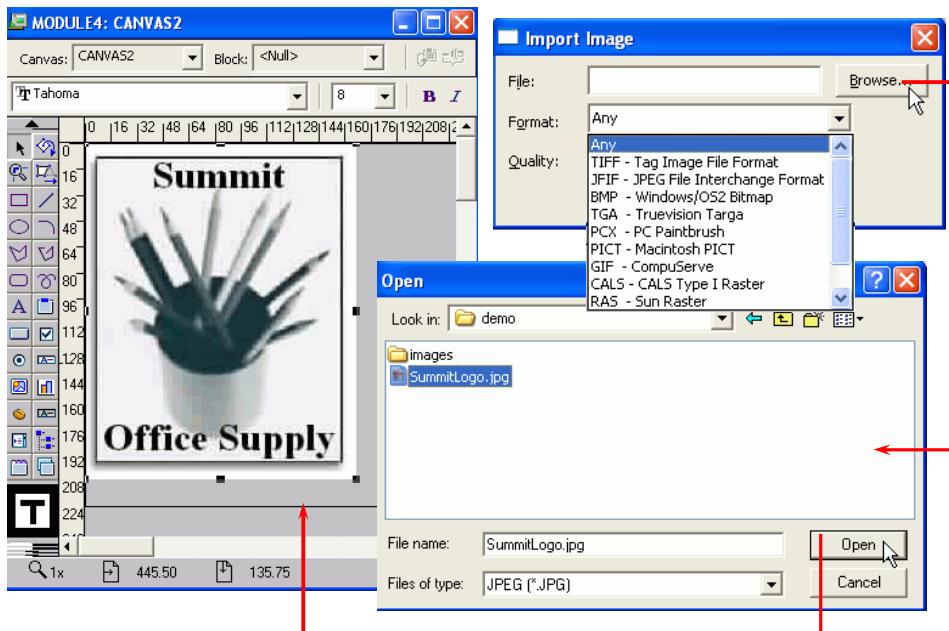
1. Select the textual objects whose color you want to change in the layout.
2. Select the Text Color tool. The color palette appears, showing the available colors, as shown in the screenshot in the slide.
3. Select a color from the color palette. (Click the appropriate square.)
4. Notice that the selected objects in the layout have adopted the chosen text color. Also, the sample area in the vertical toolbar shows a T with the selected color, and this color appears next to the Text Color tool. This indicates the current text color setting.

Altering the Color Palette

By selecting Edit > Layout Options > Color Palette from the menu, you can edit the color palette that is presented when selecting colors. This option is available only when the Builder option Color Palette Mode is set to Editable in the Preferences dialog box (Edit > Preferences). Changes to the color palette are saved with the current module.

Note: Modifications to the color palette will not be apparent until you close the document and reopen it.

Importing Images



ORACLE®

Importing Images

Oracle Forms Builder enables both static and dynamic bitmapped images to be integrated into the application. This section discusses how you can import static images—that is, those that are fixed in the layout. These might include:

- Company logos
- Photographs for inclusion in a report or display
- Background artwork

You can import images into the layout from the file system. Select Edit > Import > Image from the menus. The Import Image dialog box appears, as shown in the top-right screenshot in the slide. You can set the following Import Image options:

- **Filename:** File to be imported (click Browse to locate file, as shown in the bottom-right screenshot, which depicts selecting to import `SummitLogo.jpg`)
- **Format:** Image format; choices are Any, TIFF, JFIF (JPEG), BMP, TGA, PCX, PICT, GIF, CALS, RAS
- **Quality:** Range from Excellent to Poor. There is a trade-off between image resolution and memory required.
- **Manipulating the Imported Image:** When the imported image appears in the layout, you can move it or resize it like other objects in the layout. It is usually better to resize images in Constrained mode (while pressing the Shift key), so the image does not become distorted.

