Confidential Business Information

This documentation is proprietary information of SunGard Higher Education and is not to be copied, reproduced, lent or disposed of, nor used for any purpose other than that for which it is specifically provided without the written permission of SunGard Higher Education.

Prepared By: SunGard Higher Education
4 Country View Road
Malvern, Pennsylvania 19355
United States of America

© 2004-2006 SunGard. All rights reserved. The unauthorized possession, use, reproduction, distribution, display or disclosure of this material or the information contained herein is prohibited.

In preparing and providing this publication, SunGard Higher Education is not rendering legal, accounting, or other similar professional services. SunGard Higher Education makes no claims that an institution’s use of this publication or the software for which it is provided will insure compliance with applicable federal or state laws, rules, or regulations. Each organization should seek legal, accounting and other similar professional services from competent providers of the organization’s own choosing.

Without limitation, SunGard, the SunGard logo, Banner, Campus Pipeline, Luminis, PowerCAMPUS, Matrix, and Plus are trademarks or registered trademarks of SunGard Data Systems Inc. or its subsidiaries in the U.S. and other countries. Third-party names and marks referenced herein are trademarks or registered trademarks of their respective owners.
Table of Contents

Section A: Introduction ..............................................................................................................8
  Overview .................................................................................................................................8
  Introduction ............................................................................................................................9

Section B: Introduction to Oracle Forms ...............................................................................10
  Overview ...............................................................................................................................10
  Overview of Oracle Forms ....................................................................................................12
  Forms Components ...............................................................................................................13
  Oracle Forms and OC4J ........................................................................................................14
  Starting the Builder ...............................................................................................................15
  The Object Hierarchy ..........................................................................................................16
  The Layout Editor ................................................................................................................17
  The Property Palette ............................................................................................................18

Section C: Creating a Form Module .......................................................................................19
  Overview ................................................................................................................................19
  Creating a Form Module .......................................................................................................20
  Naming a Form Module .........................................................................................................21
  How Forms Relate to Tables ...............................................................................................23
  Data Block Wizard: Navigating the Wizards .......................................................................24
  Layout Wizard ......................................................................................................................30
  Canvases .............................................................................................................................37
  Content Canvas ....................................................................................................................39

Section D: The Layout Editor ..................................................................................................40
  Overview ................................................................................................................................40
  Layout Editor Overview .......................................................................................................42
  Moving Objects ....................................................................................................................43
  Resizing Objects ..................................................................................................................44
  Resizing the Canvas in the Layout Editor ............................................................................45
  Aligning Objects ..................................................................................................................46
  Grouping Objects ................................................................................................................47
  Boilerplate Text ....................................................................................................................48
  The Color Palette ................................................................................................................49
  Windows ...............................................................................................................................50
  Compiling Your Form ..........................................................................................................52
  Runtime Options ................................................................................................................53
  Running Your Form .............................................................................................................54
  File Types ............................................................................................................................55
  Forms Documentation .........................................................................................................56
Section E: Customizing Data Blocks .................................................................57
  Overview .....................................................................................................57
  Data Block Property Categories .................................................................58
  Making Items Navigable and Enabled .........................................................65

Section F: Text and Display Items..................................................................66
  Overview .....................................................................................................66
  Creating a Text Item ..................................................................................68
  General Properties ....................................................................................69
  Functional Properties ................................................................................70
  Creating Multi-Line Text Items .................................................................71
  Navigation Properties ................................................................................72
  Data Properties ..........................................................................................73
  Records Properties ....................................................................................74
  Database Properties ..................................................................................75
  Physical Properties ....................................................................................76
  Visual Attribute Properties .......................................................................77
  Color Properties .........................................................................................78
  Font Properties ..........................................................................................79
  Prompt Properties ......................................................................................80
  Help Properties ..........................................................................................81
  Creating a Data Block Manually .................................................................82
  Master-Detail Page (Optional) .................................................................85
  Joining Blocks Together ............................................................................86

Section G: Initial Values, Format Masks and Display Items ..........................87
  Overview .....................................................................................................87
  Initial Values .............................................................................................88
  Format Masks ............................................................................................89
  Display Items .............................................................................................94

Section H: Additional Items ..........................................................................97
  Overview .....................................................................................................97
  Defining Check Boxes ..............................................................................98
  Defining List Items ..................................................................................100
  Creating and Defining Radio Groups .....................................................104
  Creating and Defining Calculated Items ...............................................108
  Creating and Defining Buttons ...............................................................111
**Table of Contents (Continued)**

<table>
<thead>
<tr>
<th>Section I: Introduction to Triggers</th>
<th>115</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>115</td>
</tr>
<tr>
<td>Triggers Overview</td>
<td>116</td>
</tr>
<tr>
<td>PL/SQL Constructs</td>
<td>117</td>
</tr>
<tr>
<td>PL/SQL Editor</td>
<td>119</td>
</tr>
<tr>
<td>Create a New Trigger</td>
<td>120</td>
</tr>
<tr>
<td>Trigger Definition and Scope</td>
<td>121</td>
</tr>
<tr>
<td>Trigger Properties</td>
<td>126</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section J: Navigation with Triggers</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>128</td>
</tr>
<tr>
<td>WHEN-NEW-object-INSTANCE Triggers</td>
<td>129</td>
</tr>
<tr>
<td>Pre- and Post-triggers</td>
<td>130</td>
</tr>
<tr>
<td>Built-in Navigation Subprograms</td>
<td>132</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section K: Query Triggers</th>
<th>135</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>135</td>
</tr>
<tr>
<td>Query Processing Flowchart</td>
<td>136</td>
</tr>
<tr>
<td>PRE-QUERY</td>
<td>137</td>
</tr>
<tr>
<td>POST-QUERY</td>
<td>138</td>
</tr>
<tr>
<td>Fire in Enter Query Mode</td>
<td>139</td>
</tr>
<tr>
<td>Validating at the Record Level</td>
<td>140</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section L: Validation Triggers</th>
<th>141</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>141</td>
</tr>
<tr>
<td>Validating Items During Data Entry</td>
<td>142</td>
</tr>
<tr>
<td>When-Validate-Item</td>
<td>143</td>
</tr>
<tr>
<td>Comments in the PL/SQL Editor</td>
<td>148</td>
</tr>
<tr>
<td>Function Key Triggers</td>
<td>149</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section M: Transaction Processing</th>
<th>152</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>152</td>
</tr>
<tr>
<td>Transactions</td>
<td>153</td>
</tr>
<tr>
<td>Data Block-Level Transaction Triggers</td>
<td>154</td>
</tr>
<tr>
<td>Form-Level Transactional Triggers</td>
<td>159</td>
</tr>
</tbody>
</table>
Table of Contents (Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>User-Named Triggers</td>
<td>161</td>
</tr>
<tr>
<td></td>
<td>Overview</td>
<td>161</td>
</tr>
<tr>
<td></td>
<td>User-Named Triggers</td>
<td>162</td>
</tr>
<tr>
<td></td>
<td>Setting a Text Item During Runtime</td>
<td>163</td>
</tr>
<tr>
<td></td>
<td>Obtaining Property Values From Items</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>System Variables</td>
<td>167</td>
</tr>
<tr>
<td></td>
<td>Global Variables</td>
<td>169</td>
</tr>
<tr>
<td></td>
<td>DEFAULT_VALUE</td>
<td>170</td>
</tr>
<tr>
<td>O</td>
<td>Creating Lists of Values and Editors</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>Overview</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>What is a List of Values?</td>
<td>172</td>
</tr>
<tr>
<td></td>
<td>Creating an LOV Using the LOV Wizard</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>Invoking an LOV by Double-Clicking the Code Item</td>
<td>186</td>
</tr>
<tr>
<td>P</td>
<td>Canvases, Part II</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>Overview</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>About Canvases</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>Stacked Canvas</td>
<td>190</td>
</tr>
<tr>
<td></td>
<td>Toolbar Canvas</td>
<td>193</td>
</tr>
<tr>
<td></td>
<td>Tab Canvas</td>
<td>195</td>
</tr>
<tr>
<td></td>
<td>Setting a Tab Page During Runtime</td>
<td>198</td>
</tr>
<tr>
<td></td>
<td>Obtaining Property Values from Tab Pages</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Obtaining Properties of the Canvas</td>
<td>202</td>
</tr>
<tr>
<td>Q</td>
<td>Runform Messages and Alerts</td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>Overview</td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>What Kinds of Messages Are Automatically Displayed?</td>
<td>204</td>
</tr>
<tr>
<td></td>
<td>Building Your Own Messages</td>
<td>205</td>
</tr>
<tr>
<td></td>
<td>Built-in Functions that Detect Success and Failure</td>
<td>206</td>
</tr>
<tr>
<td></td>
<td>Triggers That Intercept Messages</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>Alerts</td>
<td>210</td>
</tr>
<tr>
<td></td>
<td>Displaying the Alert</td>
<td>211</td>
</tr>
<tr>
<td></td>
<td>Changing the Alert Message</td>
<td>212</td>
</tr>
</tbody>
</table>
**Section R: Sharing Objects and Code**
- Overview ................................................................. 213
- Property Classes ......................................................... 214
- Creating Property Classes .............................................. 215
- Object Groups ............................................................ 219
- Copying vs. Subclassing ................................................. 220
- Program Units ............................................................. 221
- Creating a Visual Attribute ............................................. 223

**Section S: Multiple-Form Applications**
- Overview ................................................................. 224
- About Multiple-Form Applications .................................. 225
- OPEN_FORM .............................................................. 226
- CALL_FORM built-in .................................................... 227
- NEW_FORM built-in ...................................................... 230
- Banner Form Conversion ............................................ 233

**Section T: Exercises Answer Key**
- Overview ................................................................. 241
- Answer Key ............................................................... 242
Workbook goal
This workbook provides a comprehensive discussion of Oracle Forms 10g.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Section contents
   Introduction ..............................................................................................................................9
Introduction
Oracle Forms is a development tool used for web-based database applications. All Banner forms have been written with this tool.

This course provides an in-depth discussion of how to utilize Oracle Forms 10g.

Objectives
Upon completion of this course, attendees will be able to:

• Run an Oracle Forms 10g application
• Describe the layout editor
• Create and customize a default form
• Explain canvasses, windows, blocks, items properties, and visual attributes
• Describe alerts, record groups, lists, and triggers.

Topics

• Oracle Forms 10g application
• Layout editor
• Default forms: creating and customizing
• Canvasses, windows, block properties, and visual attributes
• Banner design standards
Section B: Introduction to Oracle Forms

Lesson: Overview

Introduction
Oracle Forms is part of Internet Developer, a suite of application development tools, which includes:

- Forms
- Warehouse Builder
- Reports
- JDeveloper
- Designer
- Discover
- Software Configuration Manager
- Query Builder.

The development tools have been designed to work together, and share many components. This course will focus on developing Forms, the most robust development tool within the suite.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:

- What is Forms?
- What can Forms do?
- Forms components
- Starting the Builder
- Examining the Builder components

Prerequisites
To complete this workbook, you should have:

- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.
Section contents

Overview of Oracle Forms .................................................................12
Forms Components ............................................................................13
Oracle Forms and OC4J .................................................................14
Starting the Builder ...........................................................................15
The Object Hierarchy .......................................................................16
The Layout Editor ............................................................................17
The Property Palette ........................................................................18
What is Forms?
Oracle Forms is a development tool used for building web-based database applications that are portable to a variety of web browsers. All Banner forms have been written with this tool.

What can Forms do?
Oracle Forms allows your users to insert, update, delete, and query data from the database through GUI items. These include:

- Buttons
- Checkboxes
- Lists
- Radio Buttons
- Text Items.

As a developer, Forms allows you to quickly create applications which

- can use a number of data sources
- allow code and objects to be easily copied
- are portable across platforms.
Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forms Builder</td>
<td>The development environment.</td>
</tr>
<tr>
<td>Forms Compiler</td>
<td>Used to compile application files to create executable runfiles.</td>
</tr>
<tr>
<td>Forms Servlet</td>
<td>Runtime engine that is used to display an executable form on the web.</td>
</tr>
</tbody>
</table>

Forms modules

Oracle Forms applications include four types of modules:

<table>
<thead>
<tr>
<th>Module type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forms</td>
<td>Collections of objects and data, which allow the user to interact with the database. Data items are arranged into records.</td>
</tr>
<tr>
<td>Menus</td>
<td>Collections of menu objects (main menu, pull-down menu, menu items) and menu command code.</td>
</tr>
<tr>
<td>PL/SQL Library</td>
<td>Collections of PL/SQL procedures, functions, and packages that can be called from other modules.</td>
</tr>
<tr>
<td>Object Library</td>
<td>Collections of form objects (items, data blocks, etc…) that can be called from other modules.</td>
</tr>
</tbody>
</table>

Servlet architecture

The Forms Listener Servlet architecture allows you to deploy your Forms applications in a robust and standard manner. With this new Java servlet, you can run your applications on any network: Internet, intranet, or extranet. Only standard ports in your firewall need to be opened, and authenticating proxies are supported as well.
OC4J
Oracle Containers for J2EE (OC4J) is a complete J2EE (Java 2 Platform Enterprise Edition) server written entirely in Java which executes on the standard Java Development Kit (JDK). It provides a complete J2EE environment that contains, among other things, a Java Servlet engine.

By default, Forms uses the OC4J (J2EE) servlet engine.

Scenario
In a simple scenario, an HTTP request is received by the Oracle HTTP Listener, which passes it off to the Forms Listener Servlet running inside the OC4J process. This Forms Listener Servlet establishes a Forms Server Runtime process and is responsible for ongoing communication between the client browser and the runtime engine.

As more users request Forms sessions, the requests are received by the HTTP Listener Servlet, which again passes them off to the Forms Listener Servlet which will establish more runtime processes.

The Forms Listener Servlet is a multi-threaded process that can handle many Forms Runtime sessions simultaneously. There is, of course, a limit to the number of concurrent users this servlet can support; however, the architecture presents a number of opportunities for tuning and configuration to achieve better performance.
Starting the Builder

The Builder can be started in the following ways:

- Double-click the icon
- Locate the Forms Builder within the Windows menu system
- Enter the following command at the system prompt:
  `ifbld90 [module] [userid/password] [parameters]`

Connecting to the database

- Select **File** → **Connect**. The Connect dialog appears.
- Enter a valid username, password, and database connect string in the appropriate fields.
- Choose Connect. When you first start Oracle Forms Builder, you will see the Object Navigator.
Object Hierarchy
The Object Hierarchy provides a hierarchical display of the objects in all open modules.
- Objects are grouped under the appropriate node
- Objects and nodes in the Navigator are displayed with a + or - symbol to indicate whether they are currently expanded or collapsed
- If no + or – symbol exists, then no nodes exist

Form object types
Although we will be dealing with many types of objects within the Object Navigator during the class, the following are the major objects within a form:

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Interface objects that display information to operators and allow them to interact with your application.</td>
</tr>
<tr>
<td>Data Block</td>
<td>Each item in a form belongs to a data block:</td>
</tr>
<tr>
<td></td>
<td>- Logical containers that have no physical representation - only items are visible in the application interface</td>
</tr>
<tr>
<td></td>
<td>- Provide a mechanism for grouping related items into a functional unit for storing, displaying, and manipulating records.</td>
</tr>
<tr>
<td>Canvas</td>
<td>Area where you can &quot;paint,&quot; or design, the layout of your form. A form can contain more than one canvas (known as a page in earlier Forms versions).</td>
</tr>
</tbody>
</table>
**Layout Editor**

To view the layout of a canvas, open the Layout Editor window by selecting Tools→Layout Editor. When you begin adding objects to the form, you will be able to arrange the objects by dragging and dropping items with the mouse.

The Layout Editor will be discussed in detail later in this workbook.
Property Palette

The Property Palette provides complete control over your form, block, item and other objects.

Components

The Toolbar contains buttons giving convenient access to functions relevant to setting properties: copy, paste, add, delete, class create and inherit.

The Context bar identifies which object's properties are being displayed by the Property Palette.

The Property List is a two-column display showing the property names in the left column and their current values in the right column. The properties are grouped by category. A + in front of a category name indicates that the category is expanded and that the properties within that category are all visible.

Comparing objects

Two or more objects can be compared by selecting all of the items you want to compare, then perusing the Property Palette. Properties having the same value in all selected objects will show the shared value, while properties that do not have the same value will display ***** for that property instead of a value.

When you are showing the properties for multiple objects in a single palette, any property changed will be changed in all of the selected objects, overwriting whatever prior settings the objects had for that property.
Introduction
This section is an introduction to creating and configuring basic form modules via using wizards.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- Creating and naming form modules
- The Data Block Wizard
- The Layout Wizard
- Canvases

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
Creating a Form Module ........................................................................................................20
Naming a Form Module .........................................................................................................21
How Forms Relate to Tables .................................................................................................23
Data Block Wizard: Navigating the Wizards .........................................................................24
Layout Wizard ........................................................................................................................30
Canvases ................................................................................................................................37
Content Canvas .......................................................................................................................39
Form options
When the Form Builder is initially opened, the following options appear:
- Begin building with the Data Block Wizard
- Begin building manually – New Form 'MODULE1' is already created
- Open an existing form
- Build a form based on a template

Create additional forms
To create additional forms, select File→New→Form or highlight 'Forms' in the Object Navigator and click the icon on the toolbar.

Several modules can be opened at the same time.
Section C: Creating a Form Module

Lesson: Naming a Form Module

Changing the default name
By default, when a form is created, the form is named MODULExx, where xx stands for the next number available for the module names.

You can rename the module by doing one of the following:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Double-click the module name and edit the name, or:</td>
</tr>
</tbody>
</table>
| 2    | Access the form's Property Palette via one of the following methods:  
       | • Select Tools→Property Palette  
       | • Double-click the form module icon in the Object Navigator. The first property is the form module name  
       | • Right-click on the Form name and choose Property Palette |

Module naming rules
- Must begin with a letter
- Can include up to 30 characters, including certain special characters ($ , _ )
- Cannot include Oracle or Forms reserved words
Exercise 1
Create a new form module called SWAIDEN. The naming convention follows Banner standards. SWAIDEN stands for:

- Student
- Custom object
- Application form
- Identification

Screen image
Forms
A form is a group of related data blocks. Data Blocks are the links between a form and a database; each data block relates to one table in the database.

Data blocks
A data block is a logical container for interface items. All items, whether they come from a base table or not, must be in a data block.

Base table data blocks
A base table data block is a data block that is associated with a table in the database.

You may create base table data blocks with the Data Block Wizard, or by highlighting 'Data Blocks' in the Object Navigator and clicking the icon on the toolbar.
Lesson: Data Block Wizard: Navigating the Wizards

Buttons

- Cancel – Cancels any changes and exits the wizard
- Help – Displays online help for the current page of the wizard
- Back – Navigates to the previous wizard page
- Next – Navigates to the next wizard page
- Apply – Applies changes without exiting the wizard (Only available when the wizard is reentered)
- Finish – Saves any changes and exits the wizard

Creating a base table data block

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select <strong>Tools</strong>→<strong>Data Block Wizard</strong> to display the Data Block Wizard, or right-click 'Data Blocks' and select the Data Block Wizard.</td>
</tr>
<tr>
<td>2</td>
<td>Enter the data block information for the Data Block Wizard (see below).</td>
</tr>
<tr>
<td>3</td>
<td>Choose Finish to create the data block and dismiss the Data Block wizard.</td>
</tr>
<tr>
<td>4</td>
<td>Enter the data block information for the Layout Wizard (see below).</td>
</tr>
<tr>
<td>5</td>
<td>Choose Finish to create the layout for the data block and dismiss the Layout wizard.</td>
</tr>
</tbody>
</table>

Note: Do not click "Finish" until all the pages have been entered to your satisfaction.

Screen image

![Data Block Wizard](image-url)
Type Page – Data Block Step #1

Choose one of the two data sources:
- Table or View
- Stored Procedure

Screen image
Table Page - Data Block Step #2

<table>
<thead>
<tr>
<th>Component</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table or View</td>
<td>Enter the name of the table or view you want the data block to be based on. You can choose the Browse button to the right of the field to invoke the Tables browser. (You may not always get a scrollbar.) Alternatively, use the Refresh button to populate the Available Columns.</td>
</tr>
<tr>
<td>Enforce Data Integrity</td>
<td>Specifies whether Oracle Forms should enforce the table and column constraints defined in the data dictionary for the data block's base table.</td>
</tr>
<tr>
<td>Available Columns</td>
<td>Displays all the columns from the table that can be used within the data block.</td>
</tr>
</tbody>
</table>
| Tools                    | >  - Include only the selected column(s)  
                             >>> - Include all columns  
                             << - Exclude all columns  
                             <  - Exclude only the selected column(s) |
| Database Items           | Displays the columns from the base table that will be included in the data block. |

Screen image
Name the Block - Data Block Step #3
Select the columns, click **Next**, then name the data block. Oracle Forms will name the block the same as the database table/view by default. The block name does not have to be the same name as the table/view, and may be changed.

Screen image

![Data Block Wizard](image)
Finish Page - Data Block Step #4
The Data Block Wizard is now complete. Your two options are:

- Create the data block, then call the Layout Wizard
- Just create the data block

Screen image
Exercise 2
In the new form module, create a data block based on the SWRIDEN table. Do not include the `swriden_change_ind`, `swriden_activity_date`, `swriden_user_id` and `swriden_data_origin` columns. Display tabular style, and put everything on a new canvas later to be renamed `main_canvas`. 
Screen image
Canvas Page – Layout Step #1 (Component Usage)

<table>
<thead>
<tr>
<th>Component</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canvas</td>
<td>Select the canvas on which you want Oracle Forms to place the items that will be created in the data block. If New Canvas is selected, Oracle Forms automatically will create a new canvas.</td>
</tr>
<tr>
<td>Type</td>
<td>Type of Canvas – (Content, Stacked, Tab, Vertical Toolbar, or Horizontal Toolbar).</td>
</tr>
<tr>
<td>Tab Page</td>
<td>Specific tab to place items – Only used for tab canvas.</td>
</tr>
</tbody>
</table>

Screen image
### Data Block Page – Layout Step #2 (Component Usage)

<table>
<thead>
<tr>
<th>Component</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Block</td>
<td>The data block that is to be displayed.</td>
</tr>
<tr>
<td>Available Items</td>
<td>Specifies the data block columns that are available for display.</td>
</tr>
<tr>
<td>Tools</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt; - Include only the selected column(s)</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt; - Include all columns</td>
</tr>
<tr>
<td></td>
<td>&lt;&lt; - Exclude all columns</td>
</tr>
<tr>
<td></td>
<td>&lt;  - Exclude only the selected column(s)</td>
</tr>
<tr>
<td>Displayed Items</td>
<td>Specifies the data block columns that will be displayed.</td>
</tr>
<tr>
<td>Item Type</td>
<td>Specifies the type of the item (text box, check box, list, etc.)</td>
</tr>
</tbody>
</table>

#### Screen image

![Layout Wizard](image-url)
Section C: Creating a Form Module
Lesson: Layout Wizard (Continued)

Items Page - Layout Step #3 (Component Usage)

<table>
<thead>
<tr>
<th>Component</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the item to be displayed.</td>
</tr>
<tr>
<td>Prompt</td>
<td>Specifies the boilerplate text that Oracle Forms creates for the item.</td>
</tr>
<tr>
<td>Width</td>
<td>Specifies the width of the item's bounding box.</td>
</tr>
<tr>
<td>Height</td>
<td>Specifies the height of the item's bounding box.</td>
</tr>
</tbody>
</table>

Screen image

[Image of the Layout Wizard window with a table and items being resized]
## Style Page -- Layout Step #4 (Component Usage)

<table>
<thead>
<tr>
<th>Component</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Style</td>
<td>Form  - Items in a two-column format, with boilerplate text labels positioned to the left of each item. Tabular – All items next to each other across a single row, with boilerplate labels above each item (similar to a spreadsheet).</td>
</tr>
</tbody>
</table>

### Screen image

![Layout Wizard](image-url)
Row Page - Layout Step #5 (Component Usage)

<table>
<thead>
<tr>
<th>Component</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Title</td>
<td>Title of the data block to be displayed.</td>
</tr>
<tr>
<td>Records Displayed</td>
<td>Specifies the number of records the data block displays.</td>
</tr>
<tr>
<td>Distance Between Records</td>
<td>Specifies the amount of space between each item. Based on coordinate system unit of the form.</td>
</tr>
<tr>
<td>Display Scrollbar</td>
<td>Specifies whether Oracle Forms should create a scroll bar for the base table data block.</td>
</tr>
</tbody>
</table>

Screen image

![Layout Wizard](image)
Finish Page - – Layout Step #6
The Layout Wizard is now complete. Click the Finish button to exit.

Screen image
Canvas and viewport
Think of the viewport as a rectangle positioned on the canvas. The area of the canvas that is within the viewport is what operators see displayed in the window at runtime.
- When the viewport is the same size as the canvas, all of the canvas is visible to the operator.
- When the viewport is smaller than the canvas, only the part of the canvas that is within the viewport is visible.
- For a content or toolbar canvas, the viewport is defined by the window in which the canvas is displayed. Changing the size of the window at runtime (for example, by resizing it with the mouse) effectively changes the size of the viewport for that window's content canvas.
- For a stacked or tab canvas, the size of the viewport can be specified at design time by setting the Viewport Width and Viewport Height properties.

Canvas properties
For any type of canvas, you can set properties that specify the viewport's point of origin on the canvas; that is, to position the viewport rectangle at a specific location on its canvas. Moving the viewport, like resizing the viewport, changes the part of the canvas that operators see in the window at runtime.

When a content canvas is larger than its viewport (that is, larger than its window), the window can be scrolled to change the position of the viewport on the canvas, thus making a different part of the canvas visible.

Canvas creation methods
There are three ways to create a content canvas in Oracle Forms:
- Create a base table data block in the New Block window. A new canvas will be created if the specified name does not exist.
- Invoke the Layout Editor in a new form. (If there are no canvases in a form when you invoke the Layout Editor, Oracle Forms automatically creates a default canvas for you to work on.)
- Create a canvas in the Object Navigator. You can create a canvas of any type (Content, Stacked, Tab, Toolbar) in the Navigator.
Create via Object Navigator

To create a canvas in the Object Navigator:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Highlight the Canvases node.</td>
</tr>
<tr>
<td>2</td>
<td>Select <strong>Navigator</strong>→<strong>Create</strong> or click the <img src="image" alt="plus icon" />. The default canvas is Content.</td>
</tr>
<tr>
<td>3</td>
<td>Highlight the canvas name and choose <strong>Tools</strong> → <strong>Property Palette</strong> to invoke the property palette.</td>
</tr>
<tr>
<td>4</td>
<td>Set the <strong>Window</strong> property to specify the window in which you want the canvas to be displayed. By default, new canvases are assigned to the first window listed under the <strong>Windows</strong> node in the Navigator.</td>
</tr>
<tr>
<td>5</td>
<td>To display the new canvas in the Layout Editor, double-click the canvases object icon in the Navigator.</td>
</tr>
</tbody>
</table>
Characteristics

- Most canvases are content canvases
- A content canvas is the "base" view that occupies the entire content pane of the window in which it is displayed
- You must define at least one content canvas for each window you create
- More than one content canvas can be assigned to the same window at design time, but at runtime, only one of them at a time is displayed in the window
**Introduction**

Although your forms are functional, they probably do not have the appearance you wish. You may want to move items around, add color, change labels, increase or decrease item widths, etc. Within this section, you will be introduced to some basic layout tools that will dramatically improve the look of your forms.

**Intended audience**

Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

**Objectives**

This section will examine the following:

- Moving items with the mouse
- Resizing objects
- Resizing the canvas
- Aligning objects
- Grouping objects
- Creating and modifying boilerplate text
- Adding colors and borders

**Prerequisites**

To complete this workbook, you should have:

- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.
### Section contents

- Layout Editor Overview ................................................................. 42
- Moving Objects .............................................................................. 43
- Resizing Objects ........................................................................... 44
- Resizing the Canvas in the Layout Editor ........................................ 45
- Aligning Objects ........................................................................... 46
- Grouping Objects .......................................................................... 47
- Boilerplate Text ............................................................................ 48
- The Color Palette ......................................................................... 49
- Windows ....................................................................................... 50
- Compiling Your Form ................................................................... 52
- Runtime Options .......................................................................... 53
- Running Your Form ...................................................................... 54
- File Types ..................................................................................... 55
- Forms Documentation ................................................................... 56
What is the Layout Editor?
The Layout Editor is a graphical design facility for creating and arranging interface items and boilerplate text and graphics in a form.

Invoking the Layout Editor
- In the Navigator, double-click the object icon for the canvas-view you want to edit
  Or:
- In the Navigator, choose Layout Editor from the popup menu. (In Windows environments, right-click to display the popup menu.)
  Or:
- Choose Tools→Layout Editor, then indicate the canvas-view you want to work on

You can open more than one Layout Editor at the same time, and can copy and paste objects between Layout Editors as needed.

Closing a Layout Editor
To close a Layout Editor, double-click the Close box in the upper left corner of the window.

Restrictions
You cannot clear, cut, copy, duplicate, or export the following objects in the Layout Editor:
- the canvas object (when the View→Show Canvas option is On)
- the view rectangle (when the View→Show View option is On)
- a data block scroll bar
Moving an object or objects
To move an object, position the Select tool over it, then click and drag the object to the desired location.

To move more than one object at a time, select the objects, then click and drag any selected object to move all of the objects in the selection.

Moving a selection incrementally
To move the current selection incrementally, press the appropriate arrow key:
- [Up], [Down], [Left] or [Right]

When grid snap is turned off, the arrow keys move the current selection one pixel at a time in the direction indicated. When grid snap is turned on, the arrow keys move the current selection the distance of one snap point. Snap point increments are defined in the Ruler Settings dialog.

Constraining a move
To constrain a move to be vertical, diagonal, or horizontal, hold down [Constrain] ([Shift] on most platforms) while dragging the selected object.
Resizing objects

You can resize objects with the mouse, or by specifying precise dimensions in the Size Objects dialog.

To resize an object:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the object you want to resize.</td>
</tr>
</tbody>
</table>
| 2    | • Resize the object by dragging one of its selection handles. To constrain a resize operation, hold down the [Shift] key while resizing the selected object. For example, Shift-dragging constrains a rectangle to a square, or an ellipse to a circle.  
Or:  
• Select **Layout→Size Components** to invoke the Size Objects dialog, set **Width** and/or **Height** to **Custom**, and enter the desired setting. Once you have specified the size of an object, you can apply the same dimensions to other selected objects with the **Layout→Repeat Sizing** command. |

Making objects the same size

To make objects the same size:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the objects you want to make the same size.</td>
</tr>
<tr>
<td>2</td>
<td>Select <strong>Layout→Size Objects</strong> to open the Size Objects dialog.</td>
</tr>
<tr>
<td>3</td>
<td>Specify whether you want the selected objects to be the same height and/or width as the smallest selected object, the largest selected object, or an average of all selected objects. If you want to enter a specific width or height value, choose Custom.</td>
</tr>
<tr>
<td>4</td>
<td>Choose OK to accept the size parameters and dismiss the dialog.</td>
</tr>
</tbody>
</table>
## Resizing the canvas

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Make sure the canvas object is displayed by setting the View→Show Canvas option to On (the default).</td>
</tr>
<tr>
<td>2</td>
<td>Scroll the Layout Editor window until the lower right corner of the canvas is visible, then click the right edge or bottom edge of the canvas to select it.</td>
</tr>
<tr>
<td>3</td>
<td>When the canvas is selected, a black selection handle is displayed at its lower-right corner.</td>
</tr>
<tr>
<td>4</td>
<td>Click and drag the selection handle to resize the canvas as desired.</td>
</tr>
</tbody>
</table>
Aligning objects

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the objects you wish to align.</td>
</tr>
<tr>
<td>2</td>
<td>Select <strong>Layout→Align Components</strong> to display the Alignment Settings dialog, or use the icons on the horizontal toolbar.</td>
</tr>
<tr>
<td>3</td>
<td>Specify how you want the selected objects to be aligned. A single object can be aligned to the grid. Multiple selected objects can be aligned to the grid or to each other. You can specify horizontal and/or vertical alignment options.</td>
</tr>
<tr>
<td>4</td>
<td>Click OK to accept the settings and dismiss the dialog.</td>
</tr>
</tbody>
</table>
Groups of objects
You can select multiple objects and create a group. The group can then be manipulated as a single object. For example, you might want to group several objects together and then move the group as a unit, without changing the relative positions of the objects in the group.

Grouping objects
To group objects:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select two or more objects.</td>
</tr>
</tbody>
</table>
| 2    | Select Layout→Group Operations→Group.  

The new group is automatically selected, and you can manipulate it as you would a single object.

Ungrouping objects
To ungroup objects:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the group.</td>
</tr>
<tr>
<td>2</td>
<td>Select Layout→Group Operations→Ungroup.</td>
</tr>
</tbody>
</table>
Boilerplate graphics
Boilerplate graphics are the lines, circles, text, and images that you draw or import onto a canvas-view. Boilerplate graphics are associated with the canvas-view on which you place them; deleting a canvas-view deletes all of its boilerplate objects.

Unlike items, boilerplate graphics are not named objects. Boilerplate objects do not appear in the Object Navigator and their properties are not displayed in the Properties window.

Create boilerplate text
To create boilerplate text:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the Text tool [A] in the Layout Editor's tool palette.</td>
</tr>
<tr>
<td>2</td>
<td>Click in the workspace where you want to place text.</td>
</tr>
<tr>
<td>3</td>
<td>Type the desired text.</td>
</tr>
<tr>
<td>4</td>
<td>Exit text mode by clicking in the workspace anywhere outside the text object's bounding box.</td>
</tr>
</tbody>
</table>

Edit boilerplate text
To edit boilerplate text:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the Text tool [A] in the Layout Editor's tool palette.</td>
</tr>
<tr>
<td>2</td>
<td>Click the boilerplate text object you want to edit.</td>
</tr>
<tr>
<td>3</td>
<td>Edit the text as desired.</td>
</tr>
<tr>
<td>4</td>
<td>Click in the workspace anywhere outside the text object's bounding box to cancel text mode.</td>
</tr>
</tbody>
</table>
Section D: The Layout Editor
Lesson:   The Color Palette

Color palette toolbar
The toolbar in the Layout Editor contains tools that allow you to easily modify colors for background fill, text, and lines.

Removing boilerplate lines
By default, a boilerplate text object has a line around its bounding box. To remove this line, select the boilerplate text object, then choose the No Line option in the Line Color palette.

Exercise 3
Rename canvas1 to main_canvas.
Windows
A window is a container for all the GUI objects that make up the form application. The window is a frame for the application.

A form can include any number of windows. Every new form automatically includes a default window named WINDOW1.

Create additional windows

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Highlight the Windows node.</td>
</tr>
<tr>
<td>2</td>
<td>Select Edit→Create or click the + icon. The default window is Modeless.</td>
</tr>
<tr>
<td>3</td>
<td>Double-click the Window icon to the left of the text name to display its property palette.</td>
</tr>
</tbody>
</table>

For each window you create, you must also create at least one content canvas. You can associate the canvas with the window by setting the Primary Canvas Window property.

Physical properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X Position</td>
<td>X coordinate of the window</td>
</tr>
<tr>
<td>Y Position</td>
<td>Y coordinate of the window</td>
</tr>
<tr>
<td>Width</td>
<td>Width of the window</td>
</tr>
<tr>
<td>Height</td>
<td>Height of the window</td>
</tr>
</tbody>
</table>
### Functional Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Title of the form.</td>
</tr>
<tr>
<td>Primary Canvas</td>
<td>The name of the canvas to display in the window when invoked programmatically.</td>
</tr>
<tr>
<td>Show Vertical Toolbar</td>
<td>Determines the presence of a vertical toolbar in a window.</td>
</tr>
<tr>
<td>Show Horizontal Toolbar</td>
<td>Determines the presence of a horizontal toolbar in a window.</td>
</tr>
<tr>
<td>Modal</td>
<td>Determines whether the window is modal or modeless.</td>
</tr>
<tr>
<td>Window Style</td>
<td>Specifies whether the window is a Document window or a Dialog window. Document Style windows are fixed and must remain inside the application frame. Dialog Style windows are free and can be moved outside the application frame.</td>
</tr>
<tr>
<td>Icon Filename</td>
<td>Specifies the icon filename used to represent the minimized window.</td>
</tr>
<tr>
<td>Hide on Exit</td>
<td>Whether a modeless window closes on exit.</td>
</tr>
<tr>
<td>Window Title</td>
<td>Window title in title bar.</td>
</tr>
</tbody>
</table>

### Exercise 4

Rename window1 to main_window and give it a title.
Compile your form
To compile your form, select **Program→Compile Module**. (Alternatively, click the **Compile Module** icon (""") on the toolbar.) This will create a runtime executable (.fmx extension).

You can choose to always compile the executable before running the form from the Builder. To examine the setting, select **Edit→Preferences**. By default, the option will be set to build (compile) a new executable before running the form.
**Requirements**
Forms 10g will now only run inside a web browser. To test your modules, IAS Release 2 with Form Services needs to be installed on a server at your site.

**Run a form locally**
To run a form locally, start an OC4J instance (Oracle Containers for Java).

If Jinitiator is not installed, it will download and install.

Clicking on the Runform icon ( atleast ) will then give a default URL with port.

Set the application URL to

http://localpc:8889/forms90/f90servlet

Set the web browser to browse to

C:\Program Files\Internet Explorer\iexplore.exe

* 8889 is most likely the default port.
Run your form

Select Program→Run Form or click on the toolbar. The Runform component of Forms will be executed and you will see the form in a web browser.

Note: Keyboard mappings have changed. To change them back, go to Windows Explorer and navigate to Oracle_Forms_Home/forms90/server. Copy fmrpc.res to fmrpc.res_orig, then copy fmrwebpc.res to fmrpc.res.
File types

Form, menu, and library modules that you create in the Builder are:

- stored in binary format
- capable of being saved to files or to the database
- portable across platforms

When you generate a binary module, Oracle Forms creates a platform-specific runfile.

<table>
<thead>
<tr>
<th>Module</th>
<th>Binary (Design)</th>
<th>Text</th>
<th>Executable Runfile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>.FMB</td>
<td>.FMT</td>
<td>.FMX</td>
</tr>
<tr>
<td>Menu</td>
<td>.MMB</td>
<td>.MMT</td>
<td>.MMX</td>
</tr>
<tr>
<td>PL/SQL Library</td>
<td>.PLL</td>
<td>.PLD</td>
<td>.PLX</td>
</tr>
<tr>
<td>Object Library</td>
<td>.OLB</td>
<td>.OLT</td>
<td></td>
</tr>
</tbody>
</table>

Text versions

Text versions of the binary files can be created by selecting File→Convert.

Choose the module type (form, menu, library), the module name, and the conversion direction (binary to text, or text to binary). The text version can then be converted back, if necessary.
Documentation

Although the text versions of the modules can be opened in a word processor and examined, the text version of a module is not intended for documentation purposes. Instead, open the desired form in the Object Navigator and select **File→Administration→Object List Report.** A .txt version of the form will be created in the same directory as the form.

The Forms documentation cannot be converted into an .fmb.
Introduction
Data block properties may need to be modified after the data block has been created. With this section, we will take a look at the many data block properties you can set to enhance the functionality of your form.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- Data block properties
- Setting properties on multiple objects

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
Data Block Property Categories .............................................................................................58
Making Items Navigable and Enabled ..................................................................................65
Section E: Customizing Data Blocks

Lesson: Data Block Property Categories

General properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the data block.</td>
</tr>
<tr>
<td>Subclass Information</td>
<td>Specifies the class that has passed its properties to the data block.</td>
</tr>
<tr>
<td>Comments</td>
<td>Describes the data block in detail.</td>
</tr>
</tbody>
</table>

Navigation Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation Style</td>
<td>Once the user has navigated to the last item, where should the cursor go?</td>
</tr>
<tr>
<td></td>
<td>• Same Record, Next Record, Next Block</td>
</tr>
<tr>
<td>Next/Previous Navigation Data Block</td>
<td>Usually, the next or previous data block is determined by the order of the data blocks within the Object Navigator, which is also the order that the form posts changes to the database. If this needs to be overridden, then specify the name of the data block within these properties.</td>
</tr>
</tbody>
</table>

Exercise 5

In the SWRIDEN block, increase the number of records displayed to 5, then add a scroll bar.
## Records Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Record Visual Attribute Group</td>
<td>Allows you to associate a visual attribute with the current record. Used to highlight the current record.</td>
</tr>
<tr>
<td>Query Array Size</td>
<td>Specifies the maximum number of records that Form Builder should fetch from the database at one time.</td>
</tr>
<tr>
<td>Number of Records Buffered</td>
<td>Specifies the minimum number of records buffered in memory during a query in the data block.</td>
</tr>
<tr>
<td>Number of Records Displayed</td>
<td>Number of records to be displayed for the data block on the canvas.</td>
</tr>
<tr>
<td>Query All Records</td>
<td>Specifies whether all the records matching the query criteria should be fetched into the data block when a query is executed.</td>
</tr>
<tr>
<td>Record Orientation</td>
<td>Whether the records are oriented vertically or horizontally. The default is vertical.</td>
</tr>
</tbody>
</table>
| Single Record              | Specifies that the control data block always should contain one record.  
Note: This differs from the number of records displayed in a data block. |
Exercise 6
In the ID data block, allow the user to automatically navigate to the next record when tabbing, by modifying the Navigation Style on the data block level.

Exercise 7
Ensure that the records retrieved in the SWRIDEN data block are current (the swriden change_ind is NULL) by adding a WHERE clause.
Exercise 8
Sort the records in the SWRIDEN block by last name.

Exercise 9
Make sure the user is unable to insert, update, or delete records in the SWRIDEN block.
### Section E: Customizing Data Blocks

#### Lesson: Data Block Property Categories (Continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Data Block</td>
<td>Specifies that the data block is based on any of the following data source types: Table, Procedure, Transactional Trigger, or Sub-Query.</td>
</tr>
<tr>
<td>Enforce Primary Key</td>
<td>Indicates that any record inserted or updated in the data block must have a unique key in order to avoid committing duplicate rows to the data block's base table.</td>
</tr>
<tr>
<td>Delete Allowed, Insert Allowed, Update Allowed, Query Allowed</td>
<td>Specifies whether records can be deleted, inserted, updated, or queried within the data block. The default for these properties is Yes.</td>
</tr>
<tr>
<td>Query Data Source Type</td>
<td>Specifies the query data source type for the data block. A query data source type can be a Table, Procedure, Transactional Trigger, or FROM clause query.</td>
</tr>
<tr>
<td>Query Data Source Name</td>
<td>Specifies the name of the data block's query data source.</td>
</tr>
<tr>
<td>Query Data Source Columns</td>
<td>Specifies the names and datatypes of the columns associated with the data block's query data source. Only use when Query Data Source Type property is set to Table, Sub-query, or Procedure.</td>
</tr>
<tr>
<td>WHERE Clause/ ORDER BY Clause</td>
<td>These clauses are automatically appended to the SELECT statement that Oracle Forms constructs and issues whenever the operator or the application executes a query in the data block.</td>
</tr>
<tr>
<td>Optimizer Hint</td>
<td>Specifies a hint string that Oracle Forms passes on to the RDBMS optimizer when constructing queries. Using the optimizer can improve the performance of database transactions.</td>
</tr>
<tr>
<td>Update Changed Columns Only</td>
<td>When queried records have been marked as updates, specifies that only columns whose values were actually changed should be included in the SQL UPDATE statement that is sent to the database during a COMMIT. By default, Update Changed Columns Only is set to No, and all columns are included in the UPDATE statement.</td>
</tr>
</tbody>
</table>
Section E: Customizing Data Blocks

Lesson: Data Block Property Categories (Continued)

Database Properties (cont.)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enforce Column Security</td>
<td>Specifies when Oracle Forms should enforce update privileges on a column-by-column basis for the data block's base table. If an operator does not have update privileges on a particular column in the base table, Oracle Forms makes the corresponding item non-updateable for this operator only, by turning off the Update Allowed item property at form startup.</td>
</tr>
<tr>
<td>Maximum Query Time</td>
<td>Provides the option to abort a query when the elapsed time of the query exceeds the value of this property.</td>
</tr>
<tr>
<td>Maximum Records Fetched</td>
<td>Specifies the number of records fetched when running a query before the query is aborted.</td>
</tr>
<tr>
<td>Key Mode/Locking Mode Triggers</td>
<td>Default settings are appropriate for an Oracle database, but may need to be modified for other database types.</td>
</tr>
</tbody>
</table>

Scrollbar Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Scroll Bar</td>
<td>Specifies whether the scroll bar will be associated with the data block.</td>
</tr>
</tbody>
</table>

Visual Attribute Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Attribute Group</td>
<td>Specifies Visual Attribute for the data block.</td>
</tr>
</tbody>
</table>
Section E: Customizing Data Blocks
Lesson: Data Block Property Categories (Continued)

Property Palette icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle</td>
<td>Specifies that the property value is the default.</td>
</tr>
<tr>
<td>Square</td>
<td>Specifies that the property value has been modified and is no longer the default.</td>
</tr>
<tr>
<td>Arrow</td>
<td>Specifies that the property value is inherited.</td>
</tr>
<tr>
<td>Arrow with a Cross</td>
<td>Specifies that the property value was inherited but has been overridden.</td>
</tr>
</tbody>
</table>

Setting properties on multiple objects
You can select multiple objects at the same time by Shift-clicking or Control-clicking in the Object Navigator or the editors. When two or more objects are selected, the message 'Multiple Selection' is displayed in the Properties window instead of the Name field.

Multiple Object icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersection Icon</td>
<td>The default. Only properties common to all selected objects are displayed.</td>
</tr>
<tr>
<td>Union Icon</td>
<td>All properties of every object selected are displayed.</td>
</tr>
</tbody>
</table>
Navigable items
A navigable item is one that operators can navigate to with the [Tab] key during default navigation, or that Oracle Forms can navigate to by executing a built-in navigational procedure.

The following table shows the valid settings for these properties, and describes the resulting navigational behavior.

<table>
<thead>
<tr>
<th>Keyboard navigable</th>
<th>Enabled</th>
<th>Resulting navigation behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Item is navigable, and Oracle Forms can move the input focus to the item during default navigation. Item is displayed normally.</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Item is non-navigable. During default navigation, Oracle Forms skips over the item to the next navigable item in the sequence. Item is displayed normally, and operators can navigate to and manipulate the item with the mouse.</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Item is non-navigable, and is displayed with reduced contrast to indicate that it is unavailable for input or mouse manipulation.</td>
</tr>
</tbody>
</table>

**Exercise 10**
Make sure that the swriden_pidm is disabled and not keyboard-navigable.
Introduction
All items, just like data blocks, have individual property sheets that allow properties to be changed to enhance the functionality of the form. In this section, we will focus on text item properties.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- Create a new text item
- Examine and modify text item properties
- Create a display item

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.
Section F: Text and Display Items

Lesson: Overview (Continued)

Section contents

Creating a Text Item .................................................................68
General Properties .................................................................69
Functional Properties .........................................................70
Creating Multi-Line Text Items .............................................71
Navigation Properties .........................................................72
Data Properties .................................................................73
Records Properties .............................................................74
Database Properties ...........................................................75
Physical Properties .............................................................76
Visual Attribute Properties ...............................................77
Color Properties .................................................................78
Font Properties .................................................................79
Prompt Properties .............................................................80
Help Properties .................................................................81
Creating a Data Block Manually .........................................82
Master-Detail Page (Optional) .............................................85
Joining Blocks Together ....................................................86
Definition
A text item is an interface control that displays operator-enterable text.

Creating via the Navigator

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Highlight the Items node.</td>
</tr>
<tr>
<td>2</td>
<td>Select <strong>Edit→Create</strong> or click the icon. The default item type is a Text Item.</td>
</tr>
<tr>
<td>3</td>
<td>Double-click the Text icon to the left of the text name to display its property sheet.</td>
</tr>
</tbody>
</table>

Creating via the Layout Editor

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click the Text Item icon on the toolbar.</td>
</tr>
<tr>
<td>2</td>
<td>Click an area on the canvas to create the text item.</td>
</tr>
<tr>
<td>3</td>
<td>Double-click the text item to display its property sheet.</td>
</tr>
</tbody>
</table>
Section F: Text and Display Items

Lesson: General Properties

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Type</td>
<td>Specifies the type of item.</td>
</tr>
</tbody>
</table>
### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Determines whether operators can navigate to an item and manipulate it with the mouse.</td>
</tr>
<tr>
<td>Justification</td>
<td>Determines the text justification of the value within the item.</td>
</tr>
<tr>
<td>Multi-Line</td>
<td>Determines whether the text item is a single-line or multi-line editing region.</td>
</tr>
<tr>
<td>Wrap Style</td>
<td>Specifies how text is displayed when a line of text exceeds the width of a text item or editor window, either None, Character, or Word.</td>
</tr>
<tr>
<td>Case Restriction</td>
<td>Automatically converts the case of the user's input to either Upper, Lower, or Mixed.</td>
</tr>
<tr>
<td>Conceal Data</td>
<td>Hides characters that the operator types into the text item. This setting is typically used for password protection.</td>
</tr>
<tr>
<td>Keep Cursor Position</td>
<td>Specifies that the cursor position be the same upon re-entering the text item as when last exited.</td>
</tr>
<tr>
<td>Automatic Skip</td>
<td>Moves the cursor to the next navigable item when adding or changing data in the last character of the current item. The last character is defined by the Maximum Length property.</td>
</tr>
<tr>
<td>Popup Menu</td>
<td>Specifies the popup menu to display with the item.</td>
</tr>
</tbody>
</table>
Multi-line text items
Multi-line text items are used for displaying large columns, such as comments.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Navigator, select the desired text item.</td>
</tr>
<tr>
<td>2</td>
<td>In the Properties window, set the <strong>Multi-line</strong> property to be <em>True</em>.</td>
</tr>
<tr>
<td>3</td>
<td>Change the <strong>Wrap Style</strong> property to either <em>None, Character</em> or <em>Word</em>.</td>
</tr>
<tr>
<td>4</td>
<td>(Optional) Increase the item <strong>Height</strong>, so that multiple lines on the form can be viewed at the same time.</td>
</tr>
</tbody>
</table>
### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard Navigable</td>
<td>Determines whether the operator or the application can place the input focus in the item during default navigation.</td>
</tr>
<tr>
<td>Next Navigation Item</td>
<td>Specifies the name of the item that is defined as the &quot;next navigation item&quot; with respect to this current item. By default, the next navigation item is the item with the next higher sequence as indicated by the order of items in the Object Navigator. However, you can set this property to redefine the &quot;next item&quot; for navigation purposes.</td>
</tr>
<tr>
<td>Previous Navigation Item</td>
<td>Specifies the name of the item that is defined as the &quot;previous navigation item&quot; with respect to this current item.</td>
</tr>
</tbody>
</table>
### Data Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type</td>
<td>Specifies what kinds of values Oracle Forms allows as input and how Oracle Forms displays those values (Examples: NUMBER, CHAR).</td>
</tr>
<tr>
<td>Maximum Length</td>
<td>Specifies the maximum length of the data value that can be stored in the item.</td>
</tr>
<tr>
<td>Initial Value</td>
<td>Specifies the default value that Oracle Forms should assign to the item whenever a record is created.</td>
</tr>
<tr>
<td>Required</td>
<td>When a new record is being entered, specifies that the item is invalid when its value is NULL.</td>
</tr>
<tr>
<td>Format Mask</td>
<td>Specifies the display format and input accepted for data in text items.</td>
</tr>
<tr>
<td>Lowest / Highest Allowed Value</td>
<td>Determines the maximum value or minimum value, inclusive, that Oracle Forms allows in the text item.</td>
</tr>
<tr>
<td>Copy Value From Item</td>
<td>Specifies the source of the value that Oracle Forms uses to populate the item. When you define a master-detail relation, Oracle Forms sets this property automatically on the foreign key item(s) in the detail data block.</td>
</tr>
<tr>
<td>Synchronize with Item</td>
<td>Specifies the name of the item from which the current item should derive its value. Setting this property synchronizes the values of the two items, so that they effectively mirror each other. Use for items within the same data block.</td>
</tr>
</tbody>
</table>
### Section F: Text and Display Items

**Lesson:** Records Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Record Visual Attribute Group</td>
<td>Specifies the named visual attribute used when an item is part of the current record.</td>
</tr>
<tr>
<td>Distance Between Records</td>
<td>Specifies the amount of space between instances of the item when the item is in a multi-record data block.</td>
</tr>
<tr>
<td>Number of Items Displayed</td>
<td>Specifies the number of item instances displayed for the item when the item is in a multi-record data block.</td>
</tr>
</tbody>
</table>
# Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Item</td>
<td>Determines if the item value is stored in the data block base table.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Specifies that an item corresponds to a column in the table associated with the data block.</td>
</tr>
<tr>
<td>Primary Key</td>
<td>Set automatically for Oracle. Indicates that the item is a base table item in a base table data block and that it corresponds to a primary key column in the base table.</td>
</tr>
<tr>
<td>Query Only</td>
<td>Specifies that the item can be queried but not included in an INSERT or an UPDATE statement.</td>
</tr>
<tr>
<td>Query Allowed</td>
<td>Determines whether the item can be queried.</td>
</tr>
<tr>
<td>Query Length</td>
<td>Determines the maximum length of a query expression for the item. Should be at least as long as the Maximum Length value.</td>
</tr>
<tr>
<td>Case Insensitive Query</td>
<td>Determines whether case should be a factor when queries are performed.</td>
</tr>
<tr>
<td>Insert Allowed</td>
<td>Determines whether the item allows values to be inserted.</td>
</tr>
<tr>
<td>Update Allowed</td>
<td>Determines whether the item can be updated.</td>
</tr>
<tr>
<td>Update Only if Null</td>
<td>Determines whether the item should only be updated if the value is null for the record.</td>
</tr>
<tr>
<td>Lock Record</td>
<td>Determines whether the record is locked when the item is changed; this property is only relevant to non-base table items.</td>
</tr>
</tbody>
</table>
## Section F: Text and Display Items

### Lesson: Physical Properties

#### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>Determines whether an item that is assigned to a canvas is shown or hidden at runtime.</td>
</tr>
<tr>
<td>Canvas</td>
<td>Specifies the canvas on which you want the item to be displayed.</td>
</tr>
<tr>
<td>Tab Page</td>
<td>Specifies the tab page on which the item is located. Tab Canvases only.</td>
</tr>
<tr>
<td>X Position</td>
<td>Specifies the position of the item's upper left corner relative to the upper left corner of the item's canvas. The values you specify are interpreted in the current form coordinate units (character cells, centimeters, inches, pixels, or points), as specified by the Coordinate System form property.</td>
</tr>
<tr>
<td>Y Position</td>
<td>Specifies the position of the item's upper left corner relative to the upper left corner of the item's canvas. The values you specify are interpreted in the current form coordinate units (character cells, centimeters, inches, pixels, or points), as specified by the Coordinate System form property.</td>
</tr>
<tr>
<td>Width</td>
<td>Sets the width of the item.</td>
</tr>
<tr>
<td>Height</td>
<td>Sets the height of the item.</td>
</tr>
<tr>
<td>Bevel</td>
<td>Specifies the appearance of the object border: either RAISED, LOWERED, INSET, OUTSET, PLAIN, or NONE.</td>
</tr>
<tr>
<td>Rendered</td>
<td>Specifies that the item is to be displayed as a rendered object when it does not have focus.</td>
</tr>
<tr>
<td>Show Vertical Scroll Bar</td>
<td>Specifies that a vertical scroll bar is to appear on the side of a canvas or window.</td>
</tr>
</tbody>
</table>
### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Attribute Group</td>
<td>Specifies how the object's individual attribute settings (Font Name, Background Color, Fill Pattern, etc.) are derived.</td>
</tr>
<tr>
<td>Prompt Visual Attribute Group</td>
<td>Specifies the named visual attribute that should be applied to the prompt at runtime.</td>
</tr>
</tbody>
</table>
## Section F: Text and Display Items

### Lesson: Color Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreground Color</td>
<td>Specifies the foreground color for the item.</td>
</tr>
<tr>
<td>Background Color</td>
<td>Specifies the background color for the item.</td>
</tr>
<tr>
<td>Fill Pattern</td>
<td>Specifies the pattern for the item.</td>
</tr>
</tbody>
</table>
### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font Name</td>
<td>Specifies the name of the font to be used for the item.</td>
</tr>
<tr>
<td>Font Size, Font Style, Font Spacing, Font Weight</td>
<td>Specifies attributes for the font chosen.</td>
</tr>
</tbody>
</table>
Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Specifies the text label that displays for an item.</td>
</tr>
<tr>
<td>Prompt Display Style</td>
<td>Specifies how the prompt is displayed: either First Record, Hidden, or All Records.</td>
</tr>
<tr>
<td>Prompt Justification</td>
<td>Specifies justification of the prompt: either Left, Right, Center, Start, or End.</td>
</tr>
<tr>
<td>Prompt Attachment Edge</td>
<td>Specifies which edge the prompt should be attached to: either Start, End, Top, or Bottom.</td>
</tr>
<tr>
<td>Prompt Alignment</td>
<td>Specifies how the prompt is aligned along the item's edge: either Start, End, or Center.</td>
</tr>
<tr>
<td>Prompt Attachment Offset</td>
<td>Specifies the distance between the item and its prompt.</td>
</tr>
<tr>
<td>Prompt Alignment Offset</td>
<td>Specifies the prompt's alignment offset.</td>
</tr>
<tr>
<td>Prompt Reading Order</td>
<td>Specifies the prompt's reading order: either Default, Left to Right, or Right to Left.</td>
</tr>
</tbody>
</table>

Prompts and boilerplate

Prompts specify the text label that is associated with an item. When you move an item around in the Layout Editor, the Boilerplate Label will follow.

Associate boilerplate text with an item

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Layout Editor.</td>
</tr>
<tr>
<td>2</td>
<td>Select the item and the boilerplate text you want as the item's prompt.</td>
</tr>
<tr>
<td>3</td>
<td>Click the <strong>Associate Prompt</strong> icon.</td>
</tr>
</tbody>
</table>
### Section F: Text and Display Items

#### Lesson: Help Properties

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hint</td>
<td>The message to be displayed in the console for the item. During runtime, select Help→Help to display the hint.</td>
</tr>
<tr>
<td>Automatic Hint</td>
<td>Determines if the help text specified by the item property, Hint, is displayed automatically when the cursor enters the item.</td>
</tr>
<tr>
<td>Tooltip</td>
<td>Specifies the help text that should appear in a small box beneath the item when the mouse enters the item. (Bubble Help)</td>
</tr>
<tr>
<td>Tooltip Visual Attribute Group</td>
<td>Specifies the named visual attribute that should be applied to the tooltip at runtime.</td>
</tr>
</tbody>
</table>
Section F: Text and Display Items

Lesson: Creating a Data Block Manually

Key points
- Create the data block and assign the base table in the property palette
- Create items and associate column names
- Assign items to a canvas

Procedure

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Highlight Data Blocks.</td>
</tr>
<tr>
<td>2</td>
<td>Click the + icon.</td>
</tr>
<tr>
<td>3</td>
<td>Select <strong>Build a new data block manually</strong>.</td>
</tr>
<tr>
<td>4</td>
<td>BLOCK# appears. Rename the block <em>SWRADDR</em>.</td>
</tr>
<tr>
<td>5</td>
<td>Open the property palette.</td>
</tr>
<tr>
<td>6</td>
<td>Set <strong>Query Data Source Name</strong> to <em>SWRADDR</em>.</td>
</tr>
<tr>
<td>7</td>
<td>Highlight items.</td>
</tr>
<tr>
<td>8</td>
<td>Click the + icon.</td>
</tr>
<tr>
<td>9</td>
<td>ITEM# appears. Go to the property palette and rename it to <em>swraddr_atyp_code</em>.</td>
</tr>
<tr>
<td>10</td>
<td>Set the maximum length to 2.</td>
</tr>
<tr>
<td>11</td>
<td>Set <strong>column name</strong> to <em>swraddr_atyp_code</em>.</td>
</tr>
</tbody>
</table>
Procedure, continued

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Create a prompt called <em>Address Code</em>.</td>
</tr>
<tr>
<td>13</td>
<td>Set <strong>canvas</strong> to <em>main_canvas</em>.</td>
</tr>
</tbody>
</table>
| 14   | Repeat the above steps so that your layout editor includes the following columns:  
  - swraddr_street_line1  
  - swraddr_street_line2  
  - swraddr_city  
  - swraddr_stat_code  
  - swraddr_zip  
  - swraddr_phone_area  
  - swraddr_phone_number  
  - swraddr_activity_date |
Exercise 11
Make sure that swraddr_pidm and swraddr_activity_date are both disabled and not keyboard navigable.
Master-Detail page
The Master-Detail page allows you to establish a relationship between data blocks. It will only show up if there is a data block already created.

If Auto-join is checked, this page is used to reflect primary and foreign key relationships that have been established within the database.

If Auto-join is not checked, you can set up relationships that are not already defined at the database.

Exercise 12
Using whatever method you choose, create a data block based on the SWBPERS table.
- Include all columns except the swbpers data origin and swbpers user id
- The swbpers pidm and swbpers activity date are not enabled and keyboard navigable
- Do NOT create a master-detail relationship
Specifying a join condition
The join specified is similar to an SQL join, except the specification is
`Data_Block.item = Data_Block.item`, rather than `Table.column = Table.column`.

When the item is prefixed with a colon (:), the item behaves like a variable and represents the current value of the item.

Exercise 13
Join both the swraddr and swbpers blocks to the swrden_pidm.
Section G: Initial Values, Format Masks and Display Items

Lesson: Overview

Introduction
This section discusses date and display items and formatting techniques.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- Initializing date items
- Creating display items
- Formatting numbers and dates

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
- Initial Values .................................................................88
- Format Masks ..............................................................89
- Display Items ...............................................................94
Initial values

You can specify initial values for items by setting the Initial Value item property.

The value you specify must be compatible with the item's data type. For example, the initial value for a text item having a data type of DATE must be a value that can be displayed in a valid date format.

The initial value can be any of the following:

- **Raw Value**
  100, 'Y'

- **System Variables**
  Operating system current date/time:
  - $$DATE$$ DD-MON-YY
  - $$DATETIME$$ DD-MON-YYYY hh:mi[:ss]
  - $$TIME$$ Hh:mi[:ss]

  Current database date/time:
  - $$DBDATE$$ DD-MON-YY
  - $$DBDATETIME$$ DD-MON-YYYY hh:mi[:ss]
  - $$DBTIME$$ Hh:mi[:ss]

- **Form Item Value**
  :swrident_pidm

- **Global Variable**
  :GLOBAL.pidm

- **Form Parameter**
  :PARAMETER.pidm

- **Sequence**
  :SEQUENCE.pidm_sequence.NEXTVAL
Format masks

You can format numbers and dates in a single-line text item with format masks. The format mask must be compatible with the data type of the text item.

Examples of Number Format Masks

You may need to increase the Maximum Length and Query Length properties to take the format mask into account.

<table>
<thead>
<tr>
<th>Format mask</th>
<th>Number entered</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>999</td>
<td>223.4</td>
<td>223</td>
</tr>
<tr>
<td>$,9999.99</td>
<td>3445.34</td>
<td>$3,445.34</td>
</tr>
<tr>
<td>99.99&quot;%&quot;</td>
<td>66.17</td>
<td>66.17%</td>
</tr>
<tr>
<td>999&quot;-&quot; 999&quot;-&quot;9999</td>
<td>123456789</td>
<td>123-45-6789</td>
</tr>
</tbody>
</table>

Format masks for number values

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Represents one numeric character. The number of 9's determines how many digits the text item can display.</td>
</tr>
<tr>
<td>0</td>
<td>Displays leading zeroes when present.</td>
</tr>
<tr>
<td>$</td>
<td>Prefix number with a dollar sign.</td>
</tr>
<tr>
<td>B</td>
<td>Displays preceding zeroes as blank spaces.</td>
</tr>
<tr>
<td>MI</td>
<td>Displays &quot;-&quot; after a negative value.</td>
</tr>
<tr>
<td>PR</td>
<td>Displays negative values in &lt;angle brackets&gt;.</td>
</tr>
<tr>
<td>, (comma)</td>
<td>Displays a comma in this position as required.</td>
</tr>
<tr>
<td>. (decimal)</td>
<td>Displays a decimal in this position.</td>
</tr>
</tbody>
</table>
### Format masks for date values

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Represents one numeric character. The number of 9's determines the number of digits the text item can display.</td>
</tr>
<tr>
<td>MM</td>
<td>Month (1-12).</td>
</tr>
<tr>
<td>MON</td>
<td>Name of month, 3-letter abbreviation.</td>
</tr>
<tr>
<td>MONTH</td>
<td>Name of month, padded with blank spaces to length of 9 characters.</td>
</tr>
<tr>
<td>DD</td>
<td>Day of month (1-31).</td>
</tr>
<tr>
<td>DY</td>
<td>Name of day, 3-letter abbreviation.</td>
</tr>
<tr>
<td>DAY</td>
<td>Name of day, padded with blank spaces to length of 9 characters.</td>
</tr>
<tr>
<td>YYYY, YYY, YY, or Y</td>
<td>4, 3, or 1-digit year.</td>
</tr>
<tr>
<td>HH or HH12</td>
<td>Hour of day (1-12).</td>
</tr>
<tr>
<td>HH24</td>
<td>Hour of day (1-24).</td>
</tr>
<tr>
<td>MI</td>
<td>Minute (0-59).</td>
</tr>
<tr>
<td>SS</td>
<td>Second (0-59).</td>
</tr>
<tr>
<td>AM or A.M. or PM or P.M.</td>
<td>Meridian indicator.</td>
</tr>
<tr>
<td>TH</td>
<td>Ordinal number (e.g. &quot;DDTH&quot; for &quot;15TH&quot;)</td>
</tr>
<tr>
<td>SP</td>
<td>Spelled out number (e.g. &quot;DDSP&quot; for &quot;FIFTEEN&quot;)</td>
</tr>
<tr>
<td>SPTH or THSP</td>
<td>Spelled out ordinal number (e.g., &quot;DDSPTH&quot; for &quot;FIFTEENTH&quot;)</td>
</tr>
<tr>
<td>FM</td>
<td>Prefix used with symbols such as MONTH and DAY to suppress padding added by these symbols.</td>
</tr>
</tbody>
</table>
Format masks for character values

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Represents one alphabetic character. The number of A's determines how many characters the text item can display.</td>
</tr>
<tr>
<td>X</td>
<td>Represents one alphanumeric character. The number of X's determines how many characters the text item can display.</td>
</tr>
</tbody>
</table>

**Exercise 14**

In both the SWRADDR and SWBPERS data blocks:

- Alter the activity dates so that it initializes to the current database date for a new record. Try using an intersection to set the property for both items at the same time.

- In both data blocks, do not allow the activity dates to be changed by the user.

- Set the bubble help to **Activity Date**.

- Set the format mask so that the date appears like the following: 01-JAN-1998. *Remember to alter the maximum length to allocate for the increase in characters*.
Exercise 15
In the Person data block, set the format mask for swbpers_birth_date so that it appears like the following:

01-JAN-1998

Set the bubble help to Birth Date.

Exercise 16
In the Person data block, set the format mask for SSN so that it appears like the following at runtime:

123-45-6789
Exercise 17
In the Address data block, set the format mask for the phone number so that it appears like the following at runtime:

555-1212
Display items
Display items are similar to text items with the exception that display items only store and display fetched or assigned values. Operators cannot navigate to display items or edit display item values.

Display Item Property Sheet
Display items have fewer properties than text items, because by definition the operator cannot manipulate them.

One advantage to using a display item is that it requires less memory than a text item, since there are fewer properties for Oracle Forms to keep track of.

Create via Object Navigator
To create a display item from the Object Navigator:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the data block where you want to insert the display item, select the Items node, and select <strong>Edit → Create</strong>.</td>
</tr>
<tr>
<td>2</td>
<td>In the Properties window, set the <strong>Item Type</strong> property to <strong>Display Item</strong>.</td>
</tr>
</tbody>
</table>

Create via Layout Editor
To create a display item from the Layout Editor:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click the Display Item icon on the toolbar.</td>
</tr>
<tr>
<td>2</td>
<td>Click the canvas to place the display item on the canvas.</td>
</tr>
<tr>
<td>3</td>
<td>Resize the display item if needed.</td>
</tr>
<tr>
<td>4</td>
<td>If needed, change the data block to which the display item is assigned using the data block pop-list.</td>
</tr>
<tr>
<td>5</td>
<td>Double-click the check box to view the Property Sheet.</td>
</tr>
</tbody>
</table>
Exercise 18

In the Address data block, create two new display items named atyp_desc and state_desc.

- Ensure that they are both not base table items.
- In the Object Navigator, the atyp_desc should be under the swraddr_atyp_code. The state_desc should be under the swraddr_stat_code.
- Assign the items to the main_canvas.
- The display items will be populated by a trigger in a later exercise.
- On the canvas, place the atyp_desc to the right of the swraddr_atyp_code and the state_desc to the right of the swraddr_stat_code.
Exercise 19

Create a new data block with all columns from the SWRCMNT table (no master-detail relationship) on a new canvas called Comments. Join this block to the swriden data block. The pidm, user id and activity date are not enabled or keyboard navigable.

Create a display item cmtt_desc. The activity date should have an initial value and a format mask.
Introduction
Although text items alone would allow the user to manipulate data in the database, a form can be enhanced by adding GUI items, such as checkboxes, radio buttons, list items, calculated items, hierarchical tree items, and push buttons.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- Creating checkboxes
- Creating radio buttons
- Creating list items
- Creating calculated items
- Creating hierarchical tree items
- Creating push buttons

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
Defining Check Boxes ............................................................................................................98
Defining List Items ...............................................................................................................100
Creating and Defining Radio Groups ..................................................................................104
Creating and Defining Calculated Items ..............................................................................108
Creating and Defining Buttons .............................................................................................111
Check boxes
A check box is a two-state control that indicates whether a certain condition or value is on or off, true or false.

 Operators toggle the state of a check box by clicking it with the mouse, or by navigating to the check box and pressing [Select].

In Enter Query mode, the operator can exclude a check box as query criteria by setting the check box state to undefined. When a check box is in the undefined state, it appears disabled. Note that the undefined state is valid only in Enter Query mode.

Creating via Object Navigator
To create a check box from the Object Navigator:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the data block where you want to insert the check box, select the Items node, and choose <strong>Edit → Create</strong>.</td>
</tr>
<tr>
<td>2</td>
<td>In the Properties Palette, set the <strong>Item Type</strong> property to <strong>Check Box</strong>.</td>
</tr>
<tr>
<td>3</td>
<td>Create a label for the check box by entering the desired text in the <strong>Label</strong> property field.</td>
</tr>
<tr>
<td>4</td>
<td>Specify the values you want the check box to display as &quot;checked&quot; and &quot;unchecked&quot; in the <strong>Value When Checked</strong> and <strong>Value When Unchecked</strong> property fields.</td>
</tr>
</tbody>
</table>

Creating via Layout Editor
To create a check box from the Layout Editor:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click the Check Box icon on the toolbar.</td>
</tr>
<tr>
<td>2</td>
<td>Click the canvas to place the checkbox on the canvas.</td>
</tr>
<tr>
<td>3</td>
<td>Resize the check box if needed.</td>
</tr>
<tr>
<td>4</td>
<td>If needed, change the data block to which the check box is assigned using the data block pop-list.</td>
</tr>
<tr>
<td>5</td>
<td>Double-click the check box to view the Property Palette.</td>
</tr>
</tbody>
</table>
Check box properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Key</td>
<td>Specifies the character that will be the access key, allowing the operator to select or execute an item by pressing a key combination, such as [Alt-C].</td>
</tr>
<tr>
<td>Label</td>
<td>Specifies the text label that displays for a check box.</td>
</tr>
<tr>
<td>Mouse Navigate</td>
<td>Specifies whether Oracle Forms should perform navigation to the check box when the operator activates the item with a mouse.</td>
</tr>
<tr>
<td>Value When Checked</td>
<td>Specifies the value that is stored in the table that should indicate a checked box. The value must be compatible with the datatype.</td>
</tr>
<tr>
<td>Value When Unchecked</td>
<td>Specifies the value that is stored in the table that should indicate an unchecked box. The value must be compatible with the datatype.</td>
</tr>
<tr>
<td>Check Box Mapping of Other Values</td>
<td>Determines how to handle any value that is not one of the values represented by the checked or unchecked states. Valid choices are NOT ALLOWED, CHECKED, and UNCHECKED.</td>
</tr>
</tbody>
</table>

Exercise 20

In the Person data block, convert the swbpers_confid_ind text item to a check box.

- Set the checked state to represent the base table value of Y and the unchecked state to represent N.
- Ensure that new records are automatically assigned the value N.
- Resize the checkbox appropriately.
- Label the check box Confidential? and remove the prompt.
**List items**

A list item displays a predefined set of choices that
- are mutually exclusive
- can be displayed as either a poplist, text list, or combo box.

<table>
<thead>
<tr>
<th>List item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poplist</td>
<td>Appears initially as a single field (similar to a text item field). When the operator selects the list icon, a list of available choices appears.</td>
</tr>
<tr>
<td>Text List</td>
<td>Appears as a rectangular box that displays a fixed number of values. When the text list contains values that cannot be displayed (due to the displayable area of the item), a vertical scroll bar appears, allowing the operator to view and select undisplayed values.</td>
</tr>
<tr>
<td>Combo Box</td>
<td>Combines the features found in list and text items. Unlike the poplist or the text list style list items, the combo box style list item will display fixed values and accept one operator-entered value. The combo box list item appears as an empty box with an icon to the right. The user can enter text directly into the combo field or click the list icon to display a list of available values.</td>
</tr>
</tbody>
</table>

**List item properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Key</td>
<td>Specifies the character that will be the access key, allowing the operator to select or execute an item by pressing a key combination, such as [Alt-C].</td>
</tr>
<tr>
<td>List Style</td>
<td>Choice of Poplist, T-List, or Combo Box.</td>
</tr>
<tr>
<td>Mapping of Other Values</td>
<td>Determines how to handle any value that is not one of the values represented by the list elements.</td>
</tr>
<tr>
<td>Mouse Navigate</td>
<td>Specifies whether Oracle Forms should perform navigation to the list item when the operator activates the item with a mouse.</td>
</tr>
<tr>
<td>Elements in List</td>
<td>Opens the List Item Elements window.</td>
</tr>
</tbody>
</table>
Elements in List properties

At design time, use Ctrl + Shift + '<' to delete elements in list.

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>List Elements</td>
<td>Specifies the text that appears in the list to the operator.</td>
</tr>
<tr>
<td>List Item Value</td>
<td>Specifies the table value that should be associated with the list</td>
</tr>
<tr>
<td></td>
<td>element.</td>
</tr>
</tbody>
</table>

Defining from Object Navigator

To define a list item from the Object Navigator:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the data block where you want to insert the list item, select</td>
</tr>
<tr>
<td></td>
<td>the Items node, and then select Edit→Create.</td>
</tr>
<tr>
<td>2</td>
<td>In the Properties Palette, set the Item Type property to List Item.</td>
</tr>
<tr>
<td>3</td>
<td>Specify the display style for the list by setting the List Style</td>
</tr>
<tr>
<td></td>
<td>property to Poplist, Text List, or Combo Box.</td>
</tr>
<tr>
<td>4</td>
<td>Double-click the Elements in List property to display the List Items</td>
</tr>
<tr>
<td></td>
<td>Elements dialog, then enter the List Elements exactly as you want</td>
</tr>
<tr>
<td></td>
<td>them to appear in the list item at runtime.</td>
</tr>
<tr>
<td>5</td>
<td>Associate a value with each List Element by entering the desired</td>
</tr>
<tr>
<td></td>
<td>value in the List Item Value field, then choose OK.</td>
</tr>
</tbody>
</table>
Defining from Layout Editor

To define a list item from the Layout Editor:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click the List Item icon on the toolbar.</td>
</tr>
<tr>
<td>2</td>
<td>Click the canvas to place the list item on the canvas.</td>
</tr>
<tr>
<td>3</td>
<td>Resize the list item, if necessary.</td>
</tr>
<tr>
<td>4</td>
<td>If needed, change the data block to which the list item is assigned using the data block poplist.</td>
</tr>
<tr>
<td>5</td>
<td>Double-click the list item to display the Property Palette.</td>
</tr>
<tr>
<td>6</td>
<td>Specify the display style for the list by setting the <strong>List Style</strong> property to <strong>Poplist</strong>, <strong>Text List</strong>, or <strong>Combo Box</strong>.</td>
</tr>
<tr>
<td>7</td>
<td>Double-click the <strong>List Elements</strong> property to display the <strong>List Items Elements</strong> dialog, then enter the List Elements exactly as you want them to appear in the list item at runtime.</td>
</tr>
<tr>
<td>8</td>
<td>Associate a value with each List Element by entering the desired value in the <strong>List Item Value</strong> field, then choose OK.</td>
</tr>
</tbody>
</table>
Exercise 21
In the Person data block, convert the swbpers_mrtl_code text item to a pop-list list item.
- Add list elements of Single, Married, Widowed, and Divorced to represent database values of S, M, W, and D.
- Display any other values as Single.
- Ensure that new records display the default value Single.
- Resize the list item to see your choices at runtime.
Radio groups
A radio group is an interface control that displays a fixed number of mutually exclusive options. Each option is represented by an individual radio button.

A radio group can include any number of radio buttons. Radio buttons can be sized, positioned, and formatted independently of each other.

Radio group properties

<table>
<thead>
<tr>
<th>Radio Group Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping of Other Values</td>
<td>Determines how to handle any value that is not one of the values represented by the radio buttons.</td>
</tr>
<tr>
<td>Access Key</td>
<td>Specifies the character that will be the access key, allowing the operator to select or execute an item by pressing a key combination, such as [Alt-C].</td>
</tr>
<tr>
<td>Mouse Navigate</td>
<td>Specifies whether Oracle Forms should perform navigation to the list item when the operator activates the item with a mouse.</td>
</tr>
</tbody>
</table>

Radio button properties

<table>
<thead>
<tr>
<th>Radio Group Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Specifies the name of the individual radio button.</td>
</tr>
<tr>
<td>Access Key</td>
<td>Specifies the character that will be the access key, allowing the operator to select this button by pressing a key combination, such as [Alt-C].</td>
</tr>
<tr>
<td>Label</td>
<td>Specifies the text that should appear next to the radio button.</td>
</tr>
<tr>
<td>Radio Button Value</td>
<td>Specifies the table value that should be associated with the radio button.</td>
</tr>
</tbody>
</table>
Define via Object Navigator

To define a radio group from the Object Navigator:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the data block where you want to insert the radio group, select the Items node, and then select <strong>Edit→Create</strong>.</td>
</tr>
<tr>
<td>2</td>
<td>In the Property Palette, set the <strong>Item Type</strong> property to <strong>Radio Group</strong>.</td>
</tr>
<tr>
<td>3</td>
<td>Create the desired number of radio buttons.</td>
</tr>
<tr>
<td>4</td>
<td>In the Property Palette, create a label for the radio button by entering the desired text in the <strong>Label</strong> property field.</td>
</tr>
<tr>
<td>5</td>
<td>Enter a value for the currently selected radio button in the <strong>Radio Button Value</strong> property field. The values you assign to each radio button must be compatible with the datatype for the radio group.</td>
</tr>
<tr>
<td>6</td>
<td>Specify the display properties of the currently selected radio button.</td>
</tr>
<tr>
<td>7</td>
<td>Specify how you want the radio group to handle fetched or assigned values that are not one of the values associated with a specific radio button. To do so, set the <strong>Mapping of Other Values</strong> property for the radio group.</td>
</tr>
<tr>
<td>8</td>
<td>Set an initial value.</td>
</tr>
</tbody>
</table>
Define via Layout Editor

To define a radio button from the Layout Editor:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click the Radio Button icon on the toolbar.</td>
</tr>
<tr>
<td>2</td>
<td>Click the canvas to place the radio button on the canvas.</td>
</tr>
<tr>
<td>3</td>
<td>If a radio group exists, you will be asked if the radio button you are creating should belong to an existing radio group. If none exists, it will create a radio group for you.</td>
</tr>
<tr>
<td>4</td>
<td>Resize the radio button, if necessary.</td>
</tr>
<tr>
<td>5</td>
<td>If needed, change the data block to which the radio button is assigned using the data block poplist.</td>
</tr>
<tr>
<td>6</td>
<td>Double-click the radio button to display the Property Palette.</td>
</tr>
<tr>
<td>7</td>
<td>In the Property Palette, create a label for the radio button by entering the desired text in the Label property field.</td>
</tr>
<tr>
<td>8</td>
<td>Enter a value for the currently selected radio button in the Radio Button Value property field. The values you assign to each radio button must be compatible with the datatype for the radio group.</td>
</tr>
<tr>
<td>9</td>
<td>Specify the display properties of the currently selected radio button.</td>
</tr>
<tr>
<td>10</td>
<td>Specify how you want the radio group to handle fetched or assigned values that are not one of the values associated with a specific radio button. To do so, set the Mapping of Other Values property for the radio group.</td>
</tr>
</tbody>
</table>
Exercise 22
In the Person data block, convert the swbpers_sex text item into a radio group.

- Add radio buttons for Male, Female and Other to represent the database values of M, F and O.
- Define access keys of M for male, F for female, and O for Other.
- Define a default value of F for all new records.
Definition
Calculated items are used to hold calculations based on other items in the form. This is a feature brought over from Oracle Reports.

Calculation modes
Formula –
- The calculated item value is the result of a record-specific (horizontal) calculation. This usually involves one or more bind variables, such as form items, global variables, and parameters.

Summary –
- The calculated item value is a data block-specific (vertical) calculation. This involves the values of a single item over all the rows within a single data block.

Create via Object Navigator
To create a calculated item in the Object Navigator:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the data block where you want to insert the calculated item, select the Items node, and then select Edit→Create.</td>
</tr>
<tr>
<td>2</td>
<td>In the Property Palette, set the Item Type property to Display Item and modify the calculation-specific properties.</td>
</tr>
</tbody>
</table>

Create via Layout Editor
To create a calculated item in the Layout Editor:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click the Display Item icon on the toolbar.</td>
</tr>
<tr>
<td>2</td>
<td>Click the canvas to place the button on the canvas.</td>
</tr>
<tr>
<td>3</td>
<td>Resize the Display Item if necessary.</td>
</tr>
<tr>
<td>4</td>
<td>If needed, change the data block to which the button is assigned using the data block pop-list.</td>
</tr>
<tr>
<td>5</td>
<td>Double-click the item to view the Property Palette.</td>
</tr>
<tr>
<td>6</td>
<td>Modify the calculation-specific properties.</td>
</tr>
</tbody>
</table>

Note: Calculated items are read-only; therefore, you should use display items for calculations.
Lesson: Creating and Defining Calculated Items (Continued)

Calculated item properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculation Mode</td>
<td>Specifies the calculation method. Valid choices are Formula, Summary, or None.</td>
</tr>
<tr>
<td>Formula</td>
<td>Specifies a single PL/SQL expression that determines the value for a formula calculated item. The expression can reference built-in or user-written subprograms.</td>
</tr>
<tr>
<td>Summary Function</td>
<td>Specifies the type of Summary Function used for the calculation. Valid choices are AVG, COUNT, MAX, MIN, STDDEV, SUM, VARIANCE, or None.</td>
</tr>
<tr>
<td>Summarized Block</td>
<td>Specifies the data block to be summarized when the Calculation Mode is Summary.</td>
</tr>
<tr>
<td>Summarized Item</td>
<td>Specifies the item to be summarized when the Calculation Mode is Summary.</td>
</tr>
</tbody>
</table>

Formula calculation rules

- The formula cannot use any restricted built-ins.
- The formula cannot execute any DML statements.
- Do not end the PL/SQL formula with a semicolon (;).

Formula calculation example

Suppose that you have a form where each record has Credit Hours and Cost Per Credit Hour Items. You want to find out the Total Cost for each record.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a Calculated Item called <strong>Total Cost</strong>.</td>
</tr>
<tr>
<td>2</td>
<td>Set the Calculation Mode Property to <strong>Formula</strong>.</td>
</tr>
</tbody>
</table>
| 3    | Set the Formula Property to: 

    :DATA_BLOCK.CREDIT_HRS * :DATA_BLOCK.COST_PER_CREDIT_HR |
Summary calculation rules

- Summary items must reside in the same data block as the item being summarized, or in a control data block whose Single Record property is set to Yes.
- The summarized item must reside in a control data block, or in a data block whose Query All Records property is set to Yes.
- Datatype of the summary item must be Number, unless using MAX or MIN.
- If the summarized item values are based on a formula, that item must reside in a data block whose Query All Records property is set to Yes.

Summary calculation example

Suppose that you have a data block where each record has a Total Amount Field. You want to find out the Grand Total for the entire data block.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a Calculated Item called <strong>Grand Total</strong>. Make sure it is associated with the same data block as the item that will be summarized.</td>
</tr>
<tr>
<td>2</td>
<td>Set the Calculation Mode Property to <strong>Summary</strong>.</td>
</tr>
<tr>
<td>3</td>
<td>Set the Summary Function to <strong>SUM</strong>.</td>
</tr>
<tr>
<td>4</td>
<td>Set the Summarized Block to the proper data block.</td>
</tr>
<tr>
<td>5</td>
<td>Set the Summarized Item to <strong>Total Amount</strong>.</td>
</tr>
</tbody>
</table>
**Definition**

Buttons are interface items that operators select to execute commands or initiate actions.

For example, buttons can be used to
- initiate navigation
- display LOVs
- invoke an editor or window
- commit data in a form
- issue a query.

**Create via Object Navigator**

To create a button in the Object Navigator:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the data block where you want to insert the button, select the Items node, and then select <strong>Edit→Create</strong>.</td>
</tr>
<tr>
<td>2</td>
<td>In the Properties Palette, set the <strong>Item Type</strong> property to <strong>Push Button</strong>.</td>
</tr>
</tbody>
</table>

**Create via Layout Editor**

To create a button in the Layout Editor:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click the <strong>Button</strong> icon on the toolbar.</td>
</tr>
<tr>
<td>2</td>
<td>Click the canvas to place the button on the canvas.</td>
</tr>
<tr>
<td>3</td>
<td>Resize the button, if necessary.</td>
</tr>
<tr>
<td>4</td>
<td>If needed, change the data block to which the button is assigned using the data block poplist.</td>
</tr>
<tr>
<td>5</td>
<td>Double-click the button to view the Property Palette.</td>
</tr>
</tbody>
</table>

**Iconic buttons**

To make a button an iconic button:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Navigator, select the desired button.</td>
</tr>
<tr>
<td>2</td>
<td>In the Property Palette, set the <strong>Iconic</strong> property to <strong>Yes</strong>.</td>
</tr>
<tr>
<td>3</td>
<td>Enter the name of the icon in the <strong>Icon Filename</strong> property field. Do not include the icon file extension.</td>
</tr>
</tbody>
</table>
Designating a default button

You can designate one button per canvas as the default button. Operator can select default buttons implicitly by pressing a platform-specific key, without having to navigate to the button or activate it with the mouse.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Navigator, select the desired button.</td>
</tr>
<tr>
<td>2</td>
<td>In the Property Palette, set the <strong>Default Button</strong> property to <em>Yes</em>.</td>
</tr>
</tbody>
</table>

**Button properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Key</td>
<td>Specifies the character that will be the access key, allowing the operator to select or execute an item by pressing a key combination, such as <em>[Alt-C]</em>.</td>
</tr>
<tr>
<td>Label</td>
<td>Specifies the text label that displays for a button.</td>
</tr>
<tr>
<td>Mouse Navigate</td>
<td>Specifies whether Oracle Forms should perform navigation to the button when the operator activates the item with a mouse.</td>
</tr>
<tr>
<td>Default Button</td>
<td>Specifies whether this button is the default one for the data block.</td>
</tr>
<tr>
<td>Iconic</td>
<td>Specifies whether a button is to be an iconic button.</td>
</tr>
<tr>
<td>Iconic Filename</td>
<td>Specifies the name of the icon file. Do not include the icon file extension.</td>
</tr>
</tbody>
</table>
Adding functionality to buttons

Unlike radio buttons, check boxes, and list items, buttons need to be associated with trigger code in order to have functionality. In order for a button to perform a task, a WHEN-BUTTON-PRESSED trigger must be within its scope.

Example
/*WHEN-BUTTON-PRESSED trigger
   Shows the Address Type List of Values */

   IF SHOW_LOV('atyp_lov')
       THEN NULL;
   END IF;

SHOW_LOV

SHOW_LOV is a built-in function. It accepts the parameter of the LOV name. The function returns TRUE or FALSE, dependent upon whether or not the operator selected a value from the list. It does not return the value chosen from the list.
Exercise 23
Create a non-database data block called button_control_block.

Create four non-database items and convert all of them to push buttons:

- **Button 1: Comments**
  - Label: Comments.
  - Add trigger when-button-pressed for navigation to the Comments block. **(HINT: Use the built-in navigation sub program**
    `go_block('blockname');` **).**
  - Place on the main canvas.

- **Button 2: Home**
  - Label: Home.
  - Add trigger when-button-pressed for navigation to the Address block.
  - Place on the comments canvas.

- **Button 3: Exit**
  - Label: Exit.
  - Add trigger when-button-pressed and enter `exit_form`;
  - Place on the main_canvas.

- **Button 4: Save**
  - Label: Save
  - Add trigger when-button-pressed and enter `commit_form`;
  - Place on the main_canvas.
Introduction
Triggers are data blocks of code you write to add functionality to a default application.

You can create a basic application without writing triggers, using only Oracle Forms' default processing to retrieve, add, delete, and change database records. However, you will usually need to write triggers to customize your application.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- Purpose and types of triggers
- Rules for writing triggers

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed SunGard Higher Education's Introduction to Oracle training
- completed SunGard Higher Education’s PL/SQL and Database Objects training.

Section contents
Triggers Overview ................................................................................................................116
PL/SQL Constructs .................................................................................................................117
PL/SQL Editor .........................................................................................................................119
Create a New Trigger ...........................................................................................................120
Trigger Definition and Scope ...............................................................................................121
Trigger Properties ..................................................................................................................126
Definition
Triggers are data blocks of code you write to customize your application. The names of triggers correspond to runtime events, which in turn tell Oracle when the code should be executed.

What are triggers used for?
- Validating data entry
- Protecting the database from operator errors
- Limiting operator access to specified forms
- Displaying related field data by performing table lookups
- Comparing values between fields in the form
- Calculating field values and displaying the results of those calculations
- Performing complex transactions, such as verifying totals
- Displaying customized error and information messages to the operator
- Altering default navigation
- Displaying alert boxes
- Creating, initializing, and incrementing timers

Groups of triggers

<table>
<thead>
<tr>
<th>Group</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>When-triggers</td>
<td>Execute in addition to default processing.</td>
</tr>
<tr>
<td>On-triggers</td>
<td>Replace default processing.</td>
</tr>
<tr>
<td>Pre- and Post-</td>
<td>Add processing before or after an event.</td>
</tr>
<tr>
<td>Key-trigger</td>
<td>Change default processing assigned to a specific key.</td>
</tr>
</tbody>
</table>

Writing trigger code
The code in Oracle Forms triggers and menu item commands is written in Oracle's PL/SQL language. PL/SQL is an extension to the SQL database language, and you can include both SQL statements and PL/SQL statements in an Oracle Forms trigger. You can also make calls to built-in Oracle Forms subprograms and to user-named PL/SQL subprograms you write yourself.
PL/SQL data blocks
The text of an Oracle Forms trigger is an anonymous PL/SQL data block. A data block can consist of three sections:
- a declaration section for variables, constants, cursors, and exceptions (optional)
- executable statements (required)
- exception handlers (optional)

PL/SQL Syntax

```plsql
DECLARE
  -- declarative statements (optional)
BEGIN
  -- executable statements (required)
  EXCEPTION
  -- exception handlers (optional)
END;
```

Without DECLARE section
In a trigger, only the executable section is required. When you write a trigger that does not have a DECLARE section, you do not need to include the BEGIN and END keywords, as they are added for you implicitly.

The following example shows such a trigger:

```plsql
/* Key-CLRREC Trigger: */
IF :System.Record_Status = 'CHANGED' OR
  :System.Record_Status = 'INSERT' THEN
  Commit_Form;
END IF;
Clear_Record;
```
With DECLARE section

If, however, your trigger will have a DECLARE section, you must include the BEGIN and END keywords so the compiler can detect the start of the executable section:

```plsql
DECLARE
    total_owed NUMBER(7,2);
BEGIN
    SELECT SUM(amount)
    INTO total_owed
    FROM twraccd
    WHERE pidm = :twraccd.pidm
        AND paid_date IS NULL;
END;
```

**Note:** Transactional statements, such as COMMIT, SAVEPOINT, and ROLLBACK cannot be included directly in trigger code. Built-in subprograms (commit_form and clear_form) can be called instead.
**Section I: Introduction to Triggers**

**Lesson: PL/SQL Editor**

**PL/SQL Editor**

The PL/SQL Editor is where you enter and compile code objects. Code objects in Oracle Forms include event triggers, subprograms (functions and procedures), menu item commands, menu startup code, and packages.

**Invoking the editor**

To invoke the PL/SQL Editor:

- Choose **Tools→PL/SQL Editor** from the menu
  - Or:
- In the Navigator, double-click the object icon for any code object
  - Or:
- In the Object Navigator, Menu Editor, or Layout Editor, select an object that can have code associated with it and choose **PL/SQL Editor** from the popup menu.
Create a trigger

To create a new trigger:

- Right-click an object in the Object Navigator or Layout Editor and highlight **Smart Triggers** from the pop-up menu. This will list common triggers that are appropriate for the selected object. Select the desired trigger.
  
  Or:

- In the Object Navigator, highlight the Triggers node of the form, data block, or item that needs the trigger. Select **Edit→Create** or click the + icon on the toolbar.
Trigger definition
The object to which a trigger is attached determines the trigger's definition level in the object hierarchy. There are three levels in which a trigger can be defined:

- Form level
- Data block level
- Item level

Trigger scope
A trigger's definition level determines the trigger's scope. The scope of a trigger is its domain within the Oracle Forms object hierarchy, which determines where an event must occur for the trigger to respond to it.

A data block-level trigger fires if the trigger event occurs within that data block, but it does not fire if the same event occurs in some other data block.

Example
If you wanted a trigger to fire when Button1 in your form was pressed, you have the option to define the trigger at the item, data block, and form level.

<table>
<thead>
<tr>
<th>Level</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Level</td>
<td>Fires only when Button1 is pressed.</td>
</tr>
<tr>
<td>Data Block Level</td>
<td>Fires when any button in the data block is pressed.</td>
</tr>
<tr>
<td>Form Level</td>
<td>Fires when any button in the form is pressed.</td>
</tr>
</tbody>
</table>

By default, only the trigger that is most specific to the cursor's current location will fire. Let's expand on the example above.
Diagram 1

Within form MY_FORM, three WHEN-BUTTON-PRESSED triggers are placed.
Diagram 2

**Instance One: Button 1 is pressed.** The trigger defined at the item level is fired (C).
Diagram 3

Instance Two: Button 2 is pressed. Because no trigger is defined at the item level, Oracle Forms looks next at the Data block level. Since a WHEN-BUTTON-PRESSED trigger is defined at the data block level (B), it is fired.
Diagram 4

**Instance Three: User presses Button 3.** Forms looks for a trigger at the item level. Since none exists, it looks at the data block level. Since that does not exist, the WHEN-BUTTON-PRESSED trigger is fired at the form level (A).

![Diagram showing trigger flow](image-url)
### Execution Hierarchy

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.

The following settings are valid for this property:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Override</td>
<td>Specifies that the current trigger fire instead of any trigger by the same name at any higher scope. This is known as &quot;override parent&quot; behavior. The default.</td>
</tr>
<tr>
<td>Before</td>
<td>Specifies that the current trigger fire before firing the same trigger at the next-higher scope. This is known as &quot;fire before parent&quot; behavior.</td>
</tr>
<tr>
<td>After</td>
<td>Specifies that the current trigger fire after firing the same trigger at the next-higher scope. This is known as &quot;fire after parent&quot; behavior.</td>
</tr>
</tbody>
</table>

### Database Trigger Editor

Oracle Forms allows developers to create database triggers through the Database Trigger Editor.

To create a database trigger:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Object Navigator, expand the Database Objects node.</td>
</tr>
<tr>
<td>2</td>
<td>Expand a schema node to display its database objects.</td>
</tr>
<tr>
<td>3</td>
<td>Expand the Tables node to show the schema’s database tables.</td>
</tr>
<tr>
<td>4</td>
<td>Select and expand the desired table.</td>
</tr>
<tr>
<td>5</td>
<td>Highlight the <strong>Triggers</strong> node and select <strong>Edit→Create</strong> or click the icon on the toolbar. This brings up the Database Trigger Editor.</td>
</tr>
<tr>
<td>6</td>
<td>In the Editor, create and save the desired database triggers.</td>
</tr>
</tbody>
</table>
FORM_TRIGGER_FAILURE exception

- A predefined PL/SQL exception available only in Oracle Forms
- You can raise this exception without having to first define it in the declarative section

When does a trigger fail?

- When a FORM_TRIGGER_FAILURE is raised
- When an unhandled exception occurs

If a trigger is not failed, then processing continues. In the example above, if the RAISE FORM_TRIGGER_FAILURE was omitted, the user would receive the message but would be allowed to leave the item.
Introduction
As a user, you have been able to navigate around your form using the mouse, function keys, and the pull-down menu. Because a GUI environment offers so many ways of leaving or entering items, you may want to perform checks before allowing movement. In addition, you may want to automate more for the user, navigating to items for them programmatically.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- Default navigation sequence
- Controlling navigation through triggers
- Built-in navigation subprograms

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
WHEN-NEW-object-INSTANCE Triggers........................................................................129
Pre- and Post-triggers ........................................................................................................130
Built-in Navigation Subprograms ....................................................................................132
### Section J: Navigation with Triggers

**Lesson:** WHEN-NEW-object-INSTANCE Triggers

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHEN-NEW-ITEM-INSTANCE</td>
<td>Fires after the input focus successfully moves to an item.</td>
</tr>
<tr>
<td>WHEN-NEW-RECORD-INSTANCE</td>
<td>Fires after the input focus successfully moves to a record.</td>
</tr>
<tr>
<td>WHEN-NEW-BLOCK-INSTANCE</td>
<td>Fires after the input focus successfully moves to a data block.</td>
</tr>
<tr>
<td>WHEN-NEW-FORM-INSTANCE</td>
<td>Fires after the input focus successfully moves to a form.</td>
</tr>
<tr>
<td>Trigger</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PRE-FORM</td>
<td>Fires during the Enter the Form event, at form startup.</td>
</tr>
<tr>
<td></td>
<td>Uses:</td>
</tr>
<tr>
<td></td>
<td>• Assign unique primary key from sequence</td>
</tr>
<tr>
<td></td>
<td>• Restrict access to a form</td>
</tr>
<tr>
<td></td>
<td>• Initialize global variables.</td>
</tr>
<tr>
<td>POST-FORM</td>
<td>Fires during the Leave the Form process, when a form is exited.</td>
</tr>
<tr>
<td></td>
<td>Uses:</td>
</tr>
<tr>
<td></td>
<td>• To clean up the form before exiting, such as global variables that the form no longer requires</td>
</tr>
<tr>
<td></td>
<td>• To display a message to the operator upon form exit.</td>
</tr>
<tr>
<td>PRE-BLOCK</td>
<td>Fires during the Enter the Data Block process, during navigation from one data block to another.</td>
</tr>
<tr>
<td></td>
<td>Uses:</td>
</tr>
<tr>
<td></td>
<td>• Allow or disallow access to a data block</td>
</tr>
<tr>
<td></td>
<td>• Set variable values.</td>
</tr>
<tr>
<td>POST-BLOCK</td>
<td>Fires during the Leave the Data Block process.</td>
</tr>
<tr>
<td></td>
<td>Uses:</td>
</tr>
<tr>
<td></td>
<td>• Validate the data block's current record</td>
</tr>
<tr>
<td></td>
<td>• To test a condition and prevent the user from leaving a data block based on that condition.</td>
</tr>
<tr>
<td>PRE-RECORD</td>
<td>Fires during the Enter the Record process, during navigation to a different record.</td>
</tr>
<tr>
<td></td>
<td>Uses:</td>
</tr>
<tr>
<td></td>
<td>• Keep a running total.</td>
</tr>
</tbody>
</table>
### Section J: Navigation with Triggers

#### Lesson: Pre- and Post-triggers (Continued)

**Table, continued**

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST-RECORD</td>
<td>Fires during the Leave the Record process. Specifically, the Post-Record trigger fires whenever the operator or the application moves the input focus from one record to another.</td>
</tr>
<tr>
<td></td>
<td>Uses:</td>
</tr>
<tr>
<td></td>
<td>• Perform an action whenever the operator or the application moves the input focus from one record to another.</td>
</tr>
<tr>
<td>PRE-TEXT-ITEM</td>
<td>Fires during the Enter the Item process, during navigation from an item to a text item.</td>
</tr>
<tr>
<td></td>
<td>Uses:</td>
</tr>
<tr>
<td></td>
<td>• Derive a complex default value, based on other items previously entered into the same record</td>
</tr>
<tr>
<td></td>
<td>• Record the current value of the text item for future reference, and store that value in a global variable or form parameter.</td>
</tr>
<tr>
<td>POST-TEXT-ITEM</td>
<td>Fires during the Leave the Item process for a text item. Specifically, this trigger fires when the input focus moves from a text item to any other item.</td>
</tr>
<tr>
<td></td>
<td>Uses:</td>
</tr>
<tr>
<td></td>
<td>• Calculate or change item values.</td>
</tr>
</tbody>
</table>
### Section J: Navigation with Triggers

#### Lesson: Built-in Navigation Subprograms

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO_FORM</td>
<td>In a multiple-form application, navigates from the current form to the indicated target form.</td>
</tr>
<tr>
<td>GO_BLOCK</td>
<td>Navigates to an indicated data block. If the target data block is non-enterable, an error occurs.</td>
</tr>
<tr>
<td>GO_RECORD</td>
<td>Navigates to the record with the specified record number.</td>
</tr>
<tr>
<td>GO_ITEM</td>
<td>Navigates to an indicated item. GO_ITEM succeeds even if the target item has the Navigable property set to False.</td>
</tr>
<tr>
<td>NEXT_BLOCK</td>
<td>Navigates to the first navigable item in the next enterable data block in the navigation sequence.</td>
</tr>
<tr>
<td>NEXT_RECORD</td>
<td>Navigates to the first enabled and navigable item in the record with the next highest sequence number than the current record.</td>
</tr>
<tr>
<td>NEXT_ITEM</td>
<td>Navigates to the navigable item with the next highest sequence number than the current item.</td>
</tr>
<tr>
<td>NEXT_SET</td>
<td>Fetches another set of records from the database, then navigates to the first record that the fetch retrieves.</td>
</tr>
<tr>
<td>NEXT_KEY</td>
<td>Navigates to the enabled and navigable primary key item with the next highest sequence number than the current item.</td>
</tr>
<tr>
<td>DOWN</td>
<td>Navigates to the instance of the current item in the record with the next highest sequence number.</td>
</tr>
<tr>
<td>UP</td>
<td>Navigates to the instance of the current item in the record with the next lowest sequence number.</td>
</tr>
<tr>
<td>SCROLL_DOWN</td>
<td>Scrolls the current data block's list of records so that previously hidden records with higher sequence numbers are displayed.</td>
</tr>
<tr>
<td>SCROLL_UP</td>
<td>Scrolls the current data block's list of records so that previously hidden records with lower sequence numbers are displayed.</td>
</tr>
<tr>
<td>PREVIOUS_BLOCK</td>
<td>Navigates to the first navigable item in the previous enterable data block in the navigation sequence.</td>
</tr>
<tr>
<td>PREVIOUS_RECORD</td>
<td>Navigates to the first enabled and navigable item in the record with the next lowest sequence number than the current record.</td>
</tr>
<tr>
<td>PREVIOUS_ITEM</td>
<td>Navigates to the navigable item with the next lowest sequence number than the current item. If there is no such item, PREVIOUS_ITEM navigates to the navigable item with the highest sequence number.</td>
</tr>
</tbody>
</table>
Lesson: Built-in Navigation Subprograms
(Continued)

Example
/* WHEN-BUTTON-PRESSED-TRIGGER
Navigate to Data Block A
*/
GO_BLOCK('BLOCKA');
IF NOT FORM_SUCCESS THEN
   MESSAGE('Could not navigate to Data Block A');
   RAISE FORM_TRIGGER_FAILURE;
END IF;

FORM_SUCCESS

Returns the outcome of the action most recently performed during the current Runform session.

Use FORM_SUCCESS to test the outcome of a built-in to determine further processing within any trigger. To get the correct results, you must perform the test immediately after the action executes.
Exercise 24
At the form level, create a trigger to immediately execute a query in all blocks when the form is opened. The cursor should remain in the SWRIDEN block.

Exercise 25
When navigating to the next record in the ID block, get the records in the address block to change as well.
**Introduction**
Query triggers allow you to control events just before and just after a query.

**Intended audience**
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

**Objectives**
This section will examine the following:
- Query processing flowchart
- Triggers which screen query triggers
- Triggers which supplement query results.

**Prerequisites**
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

**Section contents**
- Query Processing Flowchart .................................................................136
- PRE-QUERY..................................................................................................137
- POST-QUERY .............................................................................................138
- Fire in Enter Query Mode ........................................................................139
- Validating at the Record Level.................................................................140
Flowchart

User click command to execute the query.

Oracle Forms issues the SELECT statement to the database.

**Pre-Query trigger is fired.** Abort Query on Failure

Query is executed.

Fetch a row into a new record.

Mark record as valid.

**Post-Query trigger is fired.**

Validate any Record Changes.
Pre-Query triggers
Fires during Execute Query or Count Query processing, just before Oracle Forms constructs and issues the SELECT statement to identify rows that match the query criteria.

Use a Pre-Query trigger to modify the example record that determines which rows will be identified by the query.

On Failure
The query is canceled. If the operator or the application had placed the form in Enter Query mode, the form would remain in Enter Query mode.

Example
This example validates or modifies query criteria for a database data block query.

BEGIN
    IF :SWRIDEN_ID IS NULL THEN
        Message('ID must be entered for Query.');
        RAISE Form_Trigger_Failure;
    END IF;
END;
**Post-Query triggers**

When a query is open in the data block, the Post-Query trigger fires each time Oracle Forms fetches a record into a data block. The trigger fires once for each record placed on the data block's list of records.

Use a Post-Query trigger to perform the following tasks:
- Populate control items or items in other data blocks
- Calculate statistics about the records retrieved by a query
- Calculate a running total.

**On Failure**

Oracle Forms flushes the record from the data block and attempts to fetch the next record from the database. If there are no other records in the database, Oracle Forms closes the query and waits for the next operator action.

**Example**

```sql
SELECT description
INTO detc_desc
FROM twvdetc
WHERE detc_code = :twraccd.detc_code;
```
Section K: Query Triggers

Lesson: Fire in Enter Query Mode

Conditions

- Specifies that the trigger should fire when the form is in Enter Query mode, as well as in Normal mode.
- Only applicable to the following triggers:
  - 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
WHEN-VALIDATE-RECORD
The WHEN-VALIDATE-RECORD trigger is used mainly to enforce that a combination of item values is valid.

Example

IF :DATE_ENROLLED > :DATE_GRADUATED THEN
    MESSAGE('The date of enrollment must occur before the date of graduation');
    RAISE FORM_TRIGGER_FAILURE;
END IF;

When does record-level validation occur?
The When-Validate-Record trigger fires during the Leave the Record event, when Oracle Forms checks to ensure that all of the items in the current record are marked valid before navigating to the target record.

Exercise 26
Create post-query triggers on the SWRADDR and SWRCMNT blocks to populate the atyp_desc, state_desc, and cmtt_desc display items.
Section L: Validation Triggers

Lesson: Overview

Introduction
Validation triggers fire when Oracle Forms validates data in an item or record. Oracle Forms performs validation checks during navigation that occur in response to operator input, programmatic control, or default processing, such as commit operation.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- When does Forms validate?
- What are the types of validation triggers?

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
Validating Items During Data Entry .................................................................142
When-Validate-Item .........................................................................................143
Comments in the PL/SQL Editor ...................................................................148
Function Key Triggers ......................................................................................149
Section L: Validation Triggers

Lesson: Validating Items During Data Entry

Setting properties
Many of the most common validation requirements can be handled by setting the following item-level properties:

- Data Type
- Maximum Length
- Required
- Range (Lowest Allowed Value / Highest Allowed Value).

When you need to add application-specific validation, you can write a When-Validate-Item trigger. This will fire after the standard checks listed above.

When does item validation occur?

- When the user tries to leave the item
- When the form is saved
Introductions
The WHEN-VALIDATE-ITEM trigger is used for two main purposes:
- validating an item
- populating non-base table items.

Populate an item
To populate an item with values from another table:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create an item of the desired type in the appropriate base table data block.</td>
</tr>
<tr>
<td>2</td>
<td>Make the item a display item by setting its <strong>Database Item</strong> property to <strong>No</strong>.</td>
</tr>
<tr>
<td>3</td>
<td>Set the item's <strong>Data Type</strong>, <strong>Maximum Length</strong>, and other properties to be compatible with the type of fetched values the item will display.</td>
</tr>
<tr>
<td>4</td>
<td>Write the necessary triggers to populate the control item at runtime.</td>
</tr>
</tbody>
</table>

Example
```
BEGIN
  SELECT swriden_pidm, swriden_last_name||
       ', '||swriden_first_name||' '|| swriden_mi
  INTO :pidm, :name
  FROM swriden
  WHERE swriden_id = :id
    AND swriden_change_ind is null;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    MESSAGE('Invalid ID. Please enter again.');
    RAISE FORM_TRIGGER_FAILURE;
END;
```
Exercise 27
In SQL+, query the stvatyp table and look at the values of the stvatyp_code.

Query the swraddr table and look at the values of the swraddr_atyp_code.

Run your form.

Enter an atyp code, what happens? (Do NOT save the changes)

What item level changes need to be made?
**Exercise 28**

In the SWRADDR block, create a trigger to populate the address description item whenever validation occurs on swraddr_atyp_code (use STVATYP).

Fail the trigger and display a suitable message if the swraddr_atyp_code is not found.

Run the form, and enter a new address with an incorrect address type. Enter a correct address type to see if the display item populates.
Exercise 29

In the SWRADDR block, create a trigger to populate the state description whenever validation occurs on swraddr_stat_code (use STVSTAT).

Fail the trigger and display a suitable message if the swraddr_stat_code is not found.

Run the form, and enter a new address with an incorrect state code. Enter a correct state code to see if the display item populates.
Exercise 30
In the SWRCMNT block, create a trigger to populate the comment description whenever validation occurs on swrcmnt_cmtt_code (use STVCMTT).

Fail the trigger and display a suitable message if the swrcmnt_cmtt_code is not found.

Run the form, and enter a new address with an incorrect comment code. Enter a correct comment code to see if the display item populates.
Comments
A double dash (--) is a single line comment in the PL/SQL editor. To comment out multiple lines of text use /* to start and */ to end.

A trigger has to do something, even if it’s nothing. If there is code in a trigger that needs to be commented out completely, then the trigger must be told to do nothing by having one line of code:

    null;

Exercise 31
Comment out the code in the following triggers:
   WHEN-NEW-FORM-INSTANCE (form level)
   WHEN-NEW-RECORD-INSTANCE (swriden block level).
Introduction

Function key triggers are associated with individual Runform function keys. A function key trigger fires only when an operator presses the associated function key. The actions defined in a function key trigger replace the default action that the function key would normally perform.

The following table shows all function key triggers and the corresponding Runform function keys.

<table>
<thead>
<tr>
<th>Key Trigger</th>
<th>Associated Function Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-CLRBLK</td>
<td>Clear Block</td>
</tr>
<tr>
<td>Key-CLFRM</td>
<td>Clear Form</td>
</tr>
<tr>
<td>Key-CLRREC</td>
<td>Clear Record</td>
</tr>
<tr>
<td>Key-COMMIT</td>
<td>Accept</td>
</tr>
<tr>
<td>Key-CQUERY</td>
<td>Count Query Hits</td>
</tr>
<tr>
<td>Key-CREREC</td>
<td>Insert Record</td>
</tr>
<tr>
<td>Key-DELRREC</td>
<td>Delete Record</td>
</tr>
<tr>
<td>Key-DOWN</td>
<td>Down</td>
</tr>
<tr>
<td>Key-DUP-ITEM</td>
<td>Duplicate Item</td>
</tr>
<tr>
<td>Key-DUPREC</td>
<td>Duplicate Record</td>
</tr>
<tr>
<td>Key-EDIT</td>
<td>Edit</td>
</tr>
<tr>
<td>Key-ENTQRY</td>
<td>Enter Query</td>
</tr>
<tr>
<td>Key-EXEQRY</td>
<td>Execute Query</td>
</tr>
<tr>
<td>Key-EXIT</td>
<td>Exit</td>
</tr>
<tr>
<td>Key-HELP</td>
<td>Help</td>
</tr>
<tr>
<td>Key-LISTVAL</td>
<td>List of Values</td>
</tr>
<tr>
<td>Key-MENU</td>
<td>Block Menu</td>
</tr>
<tr>
<td>Key-NXTBLK</td>
<td>Next Block</td>
</tr>
<tr>
<td>Key-NXT-ITEM</td>
<td>Next Item</td>
</tr>
<tr>
<td>Key-NXTKEY</td>
<td>Next Primary Key</td>
</tr>
<tr>
<td>Key-NXTREC</td>
<td>Next Record</td>
</tr>
<tr>
<td>Key-NXTSET</td>
<td>Next Set of Records</td>
</tr>
<tr>
<td>Key-PRINT</td>
<td>Print</td>
</tr>
<tr>
<td>Key-PRVBLK</td>
<td>Previous Block</td>
</tr>
<tr>
<td>Key-PRV-ITEM</td>
<td>Previous Item</td>
</tr>
<tr>
<td>Key-PRVREC</td>
<td>Previous Record</td>
</tr>
<tr>
<td>Key-SCRDOWN</td>
<td>Scroll Down</td>
</tr>
<tr>
<td>Key-SCRUP</td>
<td>Scroll Up</td>
</tr>
<tr>
<td>Key-UP</td>
<td>Up</td>
</tr>
<tr>
<td>Key-UPDREC</td>
<td>Equivalent to Record, Lock command on the default menu</td>
</tr>
</tbody>
</table>


Introduction (continued)

Note that you cannot redefine all Runform function keys with function key triggers. Specifically, you cannot redefine the following static function keys because they are often performed by the screen or user interfacemanagement system and not by Oracle Forms:

- [Clear Item]
- [Copy]
- [Cut]
- [Delete Character]
- [Delete Line]
- [Display Error]
- [End of Line]
- [First Line]
- [Insert Line]
- [Last Line]
- [Left]
- [Paste]
- [Refresh]
- [Right]
- [Scroll Left]
- [Scroll Right]
- [Search]
- [Select]
- [Show Keys]
- [Toggle Insert/Replace]
- [Transmit]
Exercise 32

- Rename the swriden block to a non-database block called KEY_BLOCK. (Display only one record and remove the scrollbar.)
- Rename the following items and make them non-database items:
  - swriden_pidm to pidm (Don’t forget to change all references from the swriden_pidm to pidm)
  - swriden_id to id (Don’t forget to change all references from the swriden_id to id)
  - swriden_last_name to name.
- Delete the swriden_first_name and swriden_mi.
- Create a validation trigger for the ID. If the ID entered is valid, populate the pidm and name; otherwise, fail the trigger and display a suitable message.
- Create an appropriate key trigger that will go to each block, execute a query and return the cursor to the address block.
- Make sure that the block level properties allow you to enter an ID.
Introduction
When a form's changes are saved during runtime, Oracle Forms enables you to fire triggers before and after events to control these actions.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- What is a transaction
- What happens during transaction processing
- Learn triggers which can be added to enhance transaction processing.

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
Transactions..................................................................................................................................153
Data Block-Level Transaction Triggers.................................................................................................154
Form-Level Transactional Triggers.........................................................................................................159
What is a transaction?
An Oracle Forms transaction is considered to be the set of all DML statements made between saves. If the user saves changes three times during the day, then three transactions occurred.

What occurs during transaction processing?
Two phases always occur once the user saves changes to a form.

- **Posting**
  The changes that were made to the records in the forms are posted to the database in data block sequence order. For each data block, Deletes are posted first, followed by Updates and Inserts.

- **Commit**
  Performs the database commit, making all changes permanent.
Timing

Fire just before or after each DML statement is posted to the database.

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRE-DELETE</td>
<td>Fires before a row is deleted. It fires once for each record that is marked for delete.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: Oracle Forms creates a Pre-Delete trigger automatically for any master-detail relation that has the Master Deletes property set to Cascading.</td>
</tr>
<tr>
<td>POST-DELETE</td>
<td>Fires after a row is deleted. It fires once for each row that is deleted from the database during the commit process.</td>
</tr>
<tr>
<td>PRE-UPDATE</td>
<td>Fires before a row is updated. It fires once for each record that is marked for update.</td>
</tr>
<tr>
<td>POST-UPDATE</td>
<td>Fires after a row is updated. It fires once for each row that is updated in the database during the commit process.</td>
</tr>
<tr>
<td>PRE-INSERT</td>
<td>Fires before a row is inserted. It fires once for each record that is marked for insert.</td>
</tr>
<tr>
<td>POST-INSERT</td>
<td>Fires just after a record is inserted. It fires once for each record that is inserted into the database during the commit process.</td>
</tr>
</tbody>
</table>

Pre-insert trigger

A pre-insert trigger is a good place to generate a one-up-number or to obtain the activity date for the record.

Example:

```sql
SELECT swrilden_pidm_sequence.NEXTVAL
INTO :swrilden.pidm
FROM sys.dual;
```
Exercise 33
Try to insert a record into the swraddr block. What happened?

Exercise 34
If you got the following message:

```
Disabled item swraddr.swraddr_pidm failed validation
```

…what should you do? (Remember, end users do not see or have access to pidms.)

Try to insert a record again. What happened?

Create a swraddr block level pre-insert trigger that puts the current value of the key block pidm into the current value of the swraddr pidm.
Exercise 35
Create pre-insert triggers for the other two blocks.
Pre-update trigger

A pre-update trigger is a good place to update the activity date or the user who is making the change to the record.

:swraddr_activity_date := SYSDATE;

Exercise 36

Create a pre-update trigger that updates the activity date in the SWBPERS, SWRADDR and SWRCMNT blocks. The SWRCMNT block should also update the user id using the keyword user.
Pre-delete trigger
To ensure that detail records do not exist before a master record is deleted, you can use the pre-delete trigger.

**Example**

```
BEGIN
  SELECT 'X'
  INTO :global.dummy
  FROM swraddr
  WHERE swraddr.pidm = :swriden.pidm;
  WHEN SQL%FOUND THEN
    MESSAGE('Data found in address table. Cannot delete Identification record.');
    RAISE FORM_TRIGGER_FAILURE;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN NULL;
END;
```
Section M: Transaction Processing

Lesson: Form-Level Transactional Triggers

PRE-COMMIT trigger
Fires once during the Post and Commit Transactions process, before Oracle Forms processes any records to change. Specifically, it fires after Oracle Forms determines that there are inserts, updates, or deletes in the form to post or commit.

The trigger does not fire when there is an attempt to commit, but validation determines that there are no changed records in the form.

Usage notes
Use a Pre-Commit trigger to perform an action, such as setting up special locking requirements, anytime that a database commit is going to occur.

On failure
The Post and Commit process fails: No records are written to the database and focus remains in the current item.

POST-FORMS-COMMIT
Fires once during the Post and Commit Transactions process. If there are records in the form that have been marked as inserts, updates, or deletes, the Post-Forms-Commit trigger fires after these changes have been written to the database but before Oracle Forms issues the database Commit to finalize the transaction.

If the operator or the application initiates a Commit when there are no records in the form have been marked as inserts, updates, or deletes, Oracle Forms fires the Post-Forms-Commit trigger immediately, without posting changes to the database.

Usage notes
Use a Post-Forms-Commit trigger to perform an action, such as updating an audit trail, anytime that a database commit is about to occur.

On failure
Aborts post and commit processing: Oracle Forms issues a ROLLBACK and decrements the internal Savepoint counter.
POST-DATABASE-COMMIT
Fires once during the Post and Commit Transactions process, after the database commit occurs.

Note that the Post-Forms-Commit trigger fires after inserts, updates, and deletes have been posted to the database, but before the transaction has been finalized by issuing the Commit. The Post-Database-Commit Trigger fires after Oracle Forms issues the Commit to finalize the transaction.

Usage notes
Use a Post-Database-Commit trigger to perform an action anytime a database commit has occurred.

On failure
There is no rollback, because at the point at which this trigger might fail, Oracle Forms has already moved past the point at which a successful rollback operation can be initiated as part of a failure response.
Section N: User-Named Triggers

Lesson: Overview

Introduction
User-named triggers have unique, user-supplied names, and are handled a little differently than other triggers.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- User-named triggers

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
User-Named Triggers ................................................................. 162
Setting a Text Item During Runtime ........................................ 163
Obtaining Property Values From Items .................................... 165
System Variables .................................................................... 167
Global Variables .................................................................... 169
DEFAULT_VALUE ..................................................................... 170
Introduction
A user-named trigger is one that has a unique, user-supplied name. Because its name does not correspond to any Oracle Forms event, a user-named trigger can only be executed by calling it from within a built-in trigger, menu item command, or user-named subprogram.

Calling a user-named trigger
To call a user-named trigger, use the EXECUTE_TRIGGER built-in procedure. This procedure takes a parameter that names the trigger to be fired and must use underscores between words.

Oracle reserves the dash for built-in triggers.

Syntax
```
execute_trigger('my_user_named_trigger');
```
SET_ITEM_PROPERTY

You may want to change an item’s property value during runtime. To do so, use the SET_ITEM_PROPERTY built-in subprogram.

Syntax

SET_ITEM_PROPERTY(item_name, property, value);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item_name</td>
<td>The name you gave the item when you created it. Datatype is VARCHAR2.</td>
</tr>
<tr>
<td>property</td>
<td>The property you want to set for the given item. Possible properties are:</td>
</tr>
<tr>
<td></td>
<td>• DISPLAYED</td>
</tr>
<tr>
<td></td>
<td>Specifies whether the item will be displayed/enabled or hidden/disabled.</td>
</tr>
<tr>
<td></td>
<td>• ENABLED</td>
</tr>
<tr>
<td></td>
<td>Specifies whether end users should be able to manipulate an item. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.</td>
</tr>
<tr>
<td></td>
<td>• REQUIRED</td>
</tr>
<tr>
<td></td>
<td>Specify the constant PROPERTY_TRUE if you want to force the end user to enter a value for the item. Specify the constant PROPERTY_FALSE if the item is not to be required.</td>
</tr>
<tr>
<td>value</td>
<td>Specify the value to be applied to the given property. The data type of the property determines the data type of the value you enter. For instance, if you want to set the VISIBLE property to TRUE, you specify the constant PROPERTY_TRUE for the value. If you want to change the LABEL for the item, you specify the value (in other words, the label) as a VARCHAR2 string.</td>
</tr>
<tr>
<td></td>
<td>• PROPERTY_TRUE</td>
</tr>
<tr>
<td></td>
<td>Specifies that the property is to be set to the TRUE state.</td>
</tr>
<tr>
<td></td>
<td>• PROPERTY_FALSE</td>
</tr>
<tr>
<td></td>
<td>Specifies that the property is to be set to the FALSE state.</td>
</tr>
</tbody>
</table>
Example

/* Hide the item */
SET_ITEM_PROPERTY('TWRACCD.AMOUNT',DISPLAYED,
        PROPERTY_FALSE);
GET_ITEM_PROPERTY

The GET_ITEM_PROPERTY built-in function returns the value of the property for a specified item.

Syntax

    GET_ITEM_PROPERTY(item_name, property);

Example

    /* Hides the account button if it is displayed. */

    IF GET_ITEM_PROPERTY('Control.account_btn',DISPLAYED) = 'TRUE' THEN
        SET_ITEM_PROPERTY('Control.account_btn', DISPLAYED, PROPERTY_FALSE);
    END IF;
Exercise 37
Create a form level user-named trigger called *enable_keys* that enables and allows navigation of the key block items.

Use `get_item_property` to check the status of the ID and Name before enabling.

Exercise 38
Create a form level user-named trigger called *disable_keys* that disables the key block ID and Name.
## Section N: User-Named Triggers

### Lesson: System Variables

**Syntax**

```
SYSTEM.VARIABLE
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODE</td>
<td>NORMAL Indicates that the form is currently in normal processing mode.</td>
</tr>
<tr>
<td></td>
<td>ENTER-QUERY Indicates that the form is currently in Enter Query mode.</td>
</tr>
<tr>
<td></td>
<td>QUERY Indicates that the form is currently in fetch processing mode, meaning that a query is currently being processed.</td>
</tr>
<tr>
<td>CURSOR_BLOCK</td>
<td>The name of the data block where the cursor is located.</td>
</tr>
<tr>
<td>CURSOR_RECORD</td>
<td>The number of the record where the cursor is located.</td>
</tr>
<tr>
<td>CURSOR_ITEM</td>
<td>The name of the data block and item (data block.item) where the cursor is located.</td>
</tr>
<tr>
<td>CURSOR_VALUE</td>
<td>The value of the item where the cursor is located.</td>
</tr>
<tr>
<td>BLOCK_STATUS</td>
<td>The status of the data block where the cursor is located:</td>
</tr>
<tr>
<td></td>
<td>CHANGED Indicates that the block contains at least one changed record.</td>
</tr>
<tr>
<td></td>
<td>NEW Indicates that the block contains only new records.</td>
</tr>
<tr>
<td></td>
<td>QUERY Indicates that the block contains only valid records that have been retrieved from the database.</td>
</tr>
<tr>
<td>CURRENT_ITEM</td>
<td>The name of the item the cursor is in.</td>
</tr>
<tr>
<td>CURRENT_BLOCK</td>
<td>The name of the block the cursor is in. It is recommended by Oracle to use cursor_block because this is included for backwards compatibility.</td>
</tr>
</tbody>
</table>
Table, continued

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORD_STATUS</td>
<td>The status of the record where the cursor is located.</td>
</tr>
<tr>
<td>CHANGED</td>
<td>Indicates that a queried record’s validation status is changed.</td>
</tr>
<tr>
<td></td>
<td>Indicates that the record’s validation status is changed and that the record does not exist in the database.</td>
</tr>
<tr>
<td>NEW</td>
<td>Indicates that the record’s validation status is new.</td>
</tr>
<tr>
<td>QUERY</td>
<td>Indicates that the record’s validation status is valid and that it was retrieved from the database.</td>
</tr>
</tbody>
</table>

Exercise 39
Create a form level when new block instance trigger. Using system variables, check for the cursor location. If it is not in the key block, execute the disable keys trigger.
Introduction
A global variable is an Oracle Forms variable whose value is accessible to triggers and subprograms in any module that is active during the current session.

A global variable stores a character string of up to 255 characters in length. Because global variables are accessible throughout an entire session, they are frequently used to keep track of variables used in a multiple-form application.

Declaration
Global variables are not formally declared the way PL/SQL local variables are. Rather, you initialize a global variable the first time you assign a value to it:

```
:GLOBAL.pidm := :swriden_pidm;
```

Referencing
To reference a global variable, prefix the variable name with the word GLOBAL and a colon.

Referencing a global variable that has not been initialized through assignment causes a runtime error.

Destroying
To destroy a global variable and release its memory, use the ERASE built-in procedure:

```
ERASE('GLOBAL.pidm');
```
Introduction
You can use the DEFAULT_VALUE built-in procedure to assign a value to a variable whose value is NULL.

If the value of the indicated variable is not NULL, DEFAULT_VALUE does nothing. If the variable to which the value is being assigned is an undefined global variable, Oracle Forms creates the variable.

Example
The following example creates a global variable named pidm and initializes it to "'":

```
DEFAULT_VALUE('','GLOBAL.pidm');
```

Exercise 40
Create a button called clear, that when pressed will
- execute a user named trigger called save keys that saves the key block id into a global variable called key_idno,
- enable the key block,
- clear the form using the clear_form built-in function,
- executes another user named trigger called global copy that will assign the global key idno back into the key block id, and
- validate the id to populate the name.

Return the cursor to the key block.
Introduction
Lists of Values provide the user with choices of values for a given item. A List of Values can contain either suggestions or a validation list.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
  • What can LOVs do?
  • How do I create an LOV?

Prerequisites
To complete this workbook, you should have:
  • equivalent experience navigating in the Banner system
  • completed Sungard Higher Education's Introduction to Oracle training
  • completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
What is a List of Values? .....................................................................................................172
Creating an LOV Using the LOV Wizard............................................................................173
Invoking an LOV by Double-Clicking the Code Item.........................................................186
Definition
A List of Values (LOV) is a scrollable pop-up window that provides the operator with either a
single or multi-column selection list.

LOVs provide the following functionality:
- Can be displayed by operator request when the operator navigates to a text item with an
  associated LOV, or programmatically, independent of any specific text items
- LOV auto-reduction and search features allow operators to locate specific values
- Values are selected by the operator can be assigned to form items according to the return
  items you designate
- At design time, an LOV can be attached to one or more text items in the form
- LOV values are derived from record groups.
Start the LOV Wizard
In the Navigator, create an LOV object by selecting the LOV’s node. Then choose Edit→Create, or click the + icon on the toolbar, or choose Tools→LOV Wizard.

Specify whether your LOV will be based on an existing record group or create a new query record group.

Specify the query
On the SQL Query page, specify the query used to construct the record group. This page is only used for new record groups.
Section O: Creating Lists of Values and Editors

Lesson: Creating an LOV Using the LOV Wizard (Continued)

Build the Query

- Click the Build SQL Query button to invoke Query Builder.
- Click the Import SQL Query button to import an existing query from a file.
- To enter a query directly, just type the query into the SQL Query Statement field.

Screen image

Additional Buttons

- Click the Connect button to connect with the database if you are not currently connected.
- Click the Check Syntax button to verify the validity of the query.
Select Record Group columns
Choose the Record Group columns that will be included in the LOV.

Screen image
Define the LOV columns
Define the look of the LOV columns, including which items will be assigned the returned LOV value.

Screen image
Design the LOV window

Design the look of the LOV window.

Screen image
Set up advanced options

Set up some advanced options:

- Number of rows fetched
- Whether the record group will be queried each time the LOV is invoked
- Whether the user will have to enter additional criteria before the LOV is displayed.

Screen image
Attach the LOV
If there are LOV values being returned, then attach the LOV to a specific Form Item.

Screen image

In the finish page, Click Finish to complete the LOV creation process.

Naming
Now that you have created an LOV and a Record Group, rename the LOV and Record Group.

Note: The standard Banner naming convention is tablename_lov for LOVs and tablename_rg for record groups.
Exercise 41
Rename the LOV and Record Group.

In the Property Palette of the swraddr atyp code, set the validate from list to yes. Comment out the when validate item trigger.

Screen image
Creating Lists of Values and Editors

Lesson: Creating an LOV Using the LOV Wizard (Continued)

Using LOV Values to Validate Text Items

Follow these steps to validate text items.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Navigator, select the desired text item.</td>
</tr>
<tr>
<td>2</td>
<td>In the Property Palette window, set the <strong>Validate from List</strong> property to Yes.</td>
</tr>
</tbody>
</table>

Creating a new LOV manually

Follow these steps to create a new LOV manually.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a record group named <em>stvstat_rg</em>.</td>
</tr>
<tr>
<td>2</td>
<td>Create a LOV named <em>stvstat_LOV</em>.</td>
</tr>
<tr>
<td>3</td>
<td>Give the LOV a title.</td>
</tr>
</tbody>
</table>
Procedure, continued

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Assign the RG.</td>
</tr>
</tbody>
</table>

![Image showing the procedure and action steps](image-url)
Procedure, continued

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Map the columns.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Attach the LOV to the item state.</td>
</tr>
<tr>
<td>7</td>
<td>Set the coordinates.</td>
</tr>
<tr>
<td>8</td>
<td>Validate from list property.</td>
</tr>
</tbody>
</table>
Lesson: Creating an LOV Using the LOV Wizard (Continued)

Screen image
Exercise 42
Create another LOV for swrcmtt_cmtt_code. In the Property Palette, set the Validate from List to yes and comment out the When-Validate-Item trigger.
SHOW_LOV
SHOW_LOV is a built-in function that displays a list of values (LOV) window at the given coordinates, returning TRUE if the operator selects a value from the list and FALSE if the operator cancels and dismisses the list.

Syntax
SHOW_LOV(lov_id LOV);
• lov_id
  Specifies the unique ID that Oracle Forms assigns the LOV when created. Use the FIND_LOV built-in to return the ID to an appropriately typed variable. The data type of the ID is LOV.

SHOW_LOV(lov_id LOV, x NUMBER, y NUMBER);
• lov_name  The name you gave to the LOV when you defined it. The data type of the name is VARCHAR2.
• x  Specifies the x coordinate of the LOV.
• y  Specifies the y coordinate of the LOV.

SHOW_LOV(lov_name VARCHAR2);
SHOW_LOV(lov_name VARCHAR2, x NUMBER, y NUMBER);

DO_KEY
DO_KEY is a built-in function that executes the key trigger that corresponds to the specified built-in subprogram. If no such key trigger exists, then the specified subprogram executes. This behavior is analogous to pressing the corresponding function key.

Syntax
PROCEDURE DO_KEY(built_in_subprogram_name VARCHAR2);

<table>
<thead>
<tr>
<th>Built-in</th>
<th>Key Trigger</th>
<th>Associated Function Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST_VALUES</td>
<td>Key-LISTVAL</td>
<td>[List]</td>
</tr>
</tbody>
</table>
Exercise 43
Under each of the code items, create a mouse double-click trigger. Either use the show_lov built in to display the LOV or the do_key built in.
Introduction
A GUI environment takes advantage of tiling, minimizing, cascading, etc. Oracle Forms allows you to take advantage of these GUI attributes.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- Creating and modifying Content Canvases
- Types of canvases
- Windows and their properties

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
About Canvases .................................................................................................................... 189
Stacked Canvas ..................................................................................................................... 190
Toolbar Canvas ..................................................................................................................... 193
Tab Canvas ........................................................................................................................... 195
Setting a Tab Page During Runtime .................................................................................... 198
Obtaining Property Values from Tab Pages ........................................................................ 200
Obtaining Properties of the Canvas ..................................................................................... 202
Introduction
Think of the viewport as a rectangle positioned on the canvas. The area of the canvas that is within the viewport is what operators see displayed in the window at runtime.

- When the viewport is the same size as the canvas, all of the canvas is visible to the operator
- When the viewport is smaller than the canvas, only the part of the canvas that is within the viewport is visible
- For a content or toolbar canvas, the viewport is defined by the window in which the canvas is displayed. Changing the size of the window at runtime (for example, by resizing it with the mouse) effectively changes the size of the viewport for that window's content canvas
- For a stacked or tab canvas, the size of the viewport can be specified at design time by setting the Viewport Width and Viewport Height properties.

For any type of canvas, you can set properties that specify the viewport’s point of origin on the canvas; that is, to position the viewport rectangle at a specific location on its canvas. Moving the viewport, like resizing the viewport rectangle at a specific location on its canvas. Moving the viewport, changes the part of the canvas that operators see in the window at runtime.

Types of canvases
- Content Canvas
- Stacked Canvas
- Toolbar Canvas
- Tab Canvas

Content canvas
Content canvases are discussed in Section C.
Section P: Canvases, Part II

Lesson:  Stacked Canvas

Characteristics
A stacked canvas is displayed in a window on top of, or "stacked" on the content canvas assigned to that same window.
- Stacked canvases obscure some part of the underlying content canvas, and are often shown and hidden programmatically
- More than one stacked canvas can be displayed in a window at the same time.

Create via Object Navigator
To create a stacked canvas in the Object Navigator:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Object Navigator, position the cursor on the Canvases node, then select Edit→Create to insert a new canvas.</td>
</tr>
<tr>
<td>2</td>
<td>In the Properties Palette, set the Canvas Type property to Stacked.</td>
</tr>
<tr>
<td>3</td>
<td>Set the other properties of the canvas object as described on the next page.</td>
</tr>
</tbody>
</table>

Create via Layout Editor
To create a stacked canvas in the Layout Editor:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Go to the Layout Editor and bring up the content canvas on which you are placing the stacked canvas.</td>
</tr>
<tr>
<td>2</td>
<td>Click the Stacked Canvas icon on the toolbar.</td>
</tr>
<tr>
<td>3</td>
<td>Click and drag the mouse in the canvas to the position where you want to place the stacked canvas.</td>
</tr>
<tr>
<td>4</td>
<td>Open the Properties Palette and set the properties of the canvas object as described on the next page.</td>
</tr>
</tbody>
</table>
Lesson:  Stacked Canvas (Continued)

Stacked Canvas properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>Set this to Yes if you want the stacked canvas to be visible when the window is invoked, or to No if you want to it to be hidden until it is shown in response to navigation or programmatic events.</td>
</tr>
<tr>
<td>Width/Height</td>
<td>Specify the size of the canvas. A stacked canvas is usually smaller than the content canvas displayed in the same window.</td>
</tr>
<tr>
<td>Bevel</td>
<td>Determines whether the stacked view has a border. A border can visually separate a stacked view from other views displayed in the same window. To make a borderless view, set Bevel to None.</td>
</tr>
<tr>
<td>Viewport Width /Viewport Height</td>
<td>Specifies the size of the viewport for the stacked canvas. If you make the view smaller than the canvas, the stacked canvas can be scrolled at runtime.</td>
</tr>
<tr>
<td>Viewport X Position /Viewport Y Position</td>
<td>Specifies the x,y display coordinates of the viewport's upper-left corner relative to the upper-left corner of the content viewport currently displayed in the window. The default setting (0,0) displays the stacked viewport at the upper-left corner of the content viewport.</td>
</tr>
<tr>
<td>Show Horizontal Scrollbar /ShowVertical Scrollbar</td>
<td>Specifies that the stacked canvas should have a horizontal and/or vertical scroll bar. Operators can scroll the stacked canvas independently of the underlying content canvas.</td>
</tr>
<tr>
<td>Window</td>
<td>Specify the window on which you want the toolbar to display.</td>
</tr>
</tbody>
</table>

Positioning

If the stacked canvas is not being shown programmatically or in response to navigation, make sure that its position in the canvas stacking order places it in front of the content canvas assigned to the same window. If it is not, it will be shown behind the content canvas, and will not be visible at runtime.
Stacking order
The stacking order of canvases in a window is defined by the sequence in which they are listed under the Canvases node in the Object Navigator.

Displaying a stacked view
To display a stacked canvas in the Layout Editor, choose View ➔ Stacked View and select the appropriate stacked canvas.

To hide the stacked canvas, hold the [CTRL] key and select the appropriate stacked canvas.

Exercise 44
Convert the comments canvas to a stacked canvas. Adjust the viewport x and y positions to make the canvas pop up in a nice place.
Characteristics
Toolbar canvases are used to create toolbars for individual windows.

- Horizontal toolbars are displayed at the top of a window, just under its menu bar
- Vertical toolbars are displayed along the left side of a window
- MDI toolbars are used to avoid creating multiple toolbars for a Multi-Form Application. Banner uses MDI Toolbars for its forms
- Associate the toolbar with a window, and remember to update the window's toolbar property.
Create a toolbar

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Object Navigator, position the cursor on the Canvases node.</td>
</tr>
<tr>
<td>2</td>
<td>Choose <strong>Edit</strong>→<strong>Create</strong> to insert a new canvas in the object hierarchy.</td>
</tr>
<tr>
<td>3</td>
<td>In the Property Palette, set the properties of the canvas as follows:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canvas Type</td>
<td>Set to Horizontal Toolbar or Vertical Toolbar.</td>
</tr>
<tr>
<td>Window</td>
<td>Specify the window on which you want the toolbar to display.</td>
</tr>
<tr>
<td>Width/Height</td>
<td>Oracle Forms will display whatever size toolbar you create, even one that completely obscures the window's content canvas-view. It is up to you to set the <strong>Width</strong> and <strong>Height</strong> properties to appropriate values, depending on how large you want the toolbar to be.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>For the window to which you assigned the toolbar canvas, set the <strong>Horizontal Toolbar Canvas</strong> or <strong>Vertical Toolbar Canvas</strong> property by specifying the name of the toolbar canvas you created in step 2.</td>
</tr>
<tr>
<td>5</td>
<td>For the Form to which you want to assign a MDI Toolbar, set the <strong>Form Horizontal Toolbar Canvas</strong> or <strong>Form Vertical Toolbar Canvas</strong> property by specifying the name of the toolbar canvas you created in step 2.</td>
</tr>
<tr>
<td>6</td>
<td>Add items and boilerplate graphics to the toolbar canvas as you would for any other canvas.</td>
</tr>
</tbody>
</table>

**Exercise 45**
Save your form as swaiden_tab.fmb.
Characteristics
A tab canvas is displayed in a window on top of the content canvas assigned to that same window.

- Tab canvases enable you to organize and display related information on separate tabs
- Tab canvases are made up of one or more tab pages, which have labeled tabs that comprise an equal amount of space on the tab canvas
- Tab canvases can be used to display large amount of information on a single canvas and give the application a "Web" feel.

Create via Object Navigator
To create a tab canvas in the Object Navigator:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Object Navigator, position the cursor on the Canvases node, then select Edit→Create to insert a new canvas.</td>
</tr>
<tr>
<td>2</td>
<td>In the Property Palette, set the Canvas Type property to Tab.</td>
</tr>
<tr>
<td>3</td>
<td>Set the other properties of the tab canvas object as described below.</td>
</tr>
<tr>
<td>4</td>
<td>Expand the canvas node in the Object Navigator so it displays Tab Pages.</td>
</tr>
<tr>
<td>5</td>
<td>Highlight the Tab Pages node and then choose Edit→Create to insert a new tab page.</td>
</tr>
<tr>
<td>6</td>
<td>Set the tab page properties as described below.</td>
</tr>
<tr>
<td>7</td>
<td>Create additional tab pages by repeating steps 5 and 6.</td>
</tr>
</tbody>
</table>

Creating via Layout Editor
To create a tab canvas in the Layout Editor:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Go to the Layout Editor and bring up the content canvas on which you are placing the tab canvas.</td>
</tr>
<tr>
<td>2</td>
<td>Click the Tab Canvas icon on the toolbar.</td>
</tr>
<tr>
<td>3</td>
<td>Click and drag the mouse in the canvas to the position where you want to place the tab canvas.</td>
</tr>
<tr>
<td>4</td>
<td>Open the Properties Palette and set the properties of the canvas object as described below.</td>
</tr>
<tr>
<td>5</td>
<td>Create additional tab pages in the Object Navigator.</td>
</tr>
<tr>
<td>6</td>
<td>Set the tab page properties as described below.</td>
</tr>
<tr>
<td>7</td>
<td>Create additional tab pages by repeating steps 5 and 6.</td>
</tr>
</tbody>
</table>
**Section P: Canvases, Part II**

**Lesson: Tab Canvas (Continued)**

### Tab Canvas Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>Set this to Yes if you want the tab canvas to be visible when the window is invoked, or to No if you want it to be hidden until it is shown in response to navigation or programmatic events.</td>
</tr>
<tr>
<td>Width/Height</td>
<td>Specify the size of the canvas. A tab canvas is usually smaller than the content canvas displayed in the same window.</td>
</tr>
<tr>
<td>Bevel</td>
<td>Determines whether the tab canvas has a border. A border can visually separate a stacked view from other views displayed in the same window. To make a borderless view, set Bevel to None.</td>
</tr>
<tr>
<td>Viewport Width / Viewport Height</td>
<td>Specifies the size of the viewport for the tab canvas.</td>
</tr>
<tr>
<td>Viewport X Position / Viewport Y Position</td>
<td>Specifies the x,y display coordinates of the viewport's upper-left corner relative to the upper-left corner of the content viewport currently displayed in the window. The default setting (0,0) displays the stacked viewport at the upper-left corner of the content viewport.</td>
</tr>
<tr>
<td>Corner Style</td>
<td>Specifies the shape of the label tabs. Choose from Chambered, Square, and Rounded.</td>
</tr>
<tr>
<td>Tab Attachment Edge</td>
<td>Specifies the location where the tabs are attached to the tab canvas.</td>
</tr>
</tbody>
</table>

### Tab Page Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Specifies the text label for the tab page.</td>
</tr>
</tbody>
</table>
Stacking order
If the tab canvas is not being shown programmatically or in response to navigation, make sure that its position in the canvas stacking order places it in front of the content canvas assigned to the same window. If it is not, it will be shown behind the content canvas, and will not be visible at runtime.

The stacking order of canvases in a window is defined by the sequence in which they are listed under the Canvases node in the Object Navigator.

Displaying a Tab Canvas
To display a tab canvas in the Layout Editor, choose View ➔ Stacked View and select the appropriate tab canvas.

To hide the tab canvas, hold the [Ctrl] key and select the appropriate tab canvas.

Placing Items on a Tab Page
In order to use the new tab canvas, you must place individual items onto the tab pages. To accomplish this, do the following:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the item's Property Palette.</td>
</tr>
<tr>
<td>2</td>
<td>Set the item's Canvas Property and Tab Page property to the desired tab canvas and tab page.</td>
</tr>
</tbody>
</table>

Exercise 46
Convert the comments canvas to a tab canvas.

Create three new pages: Address, Bio/Demo and Comments.
- Move items on the swraddr block to the comments canvas Address page.
- Move items on the swbpers block to the comments canvas Bio/Demo page.
- Move items on the comment block to the comments canvas Comments page.
SET_TAB_PAGE_PROPERTY
You may want to change a tab page's property value during runtime. To do so, use the
SET_TAB_PAGE_PROPERTY built-in subprogram.

Syntax
SET_TAB_PAGE_PROPERTY(tab_page,property,value);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab_page</td>
<td>The name you gave the tab page when you created it. Datatype is VARCHAR2.</td>
</tr>
<tr>
<td>property</td>
<td>The property you want to set for the given tab page. Possible properties are:</td>
</tr>
<tr>
<td></td>
<td>• ENABLED Specifies whether end users should be able to manipulate a tab page. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.</td>
</tr>
<tr>
<td>value</td>
<td>Specify the value to be applied to the given property. The data type of the property determines the data type of the value you enter.</td>
</tr>
<tr>
<td></td>
<td>• PROPERTY_TRUE Specifies that the property is to be set to the TRUE state.</td>
</tr>
<tr>
<td></td>
<td>• PROPERTY_FALSE Specifies that the property is to be set to the FALSE state.</td>
</tr>
</tbody>
</table>

Note: If all tabbed pages are disabled programmatically, Oracle forms will enable the last tabbed page defined for the tabbed canvas.
Exercise 47
Create a user-named trigger called DISABLE_PAGES that will disable the BIO_DEMO and COMMENTS pages.
Section P: Canvases, Part II

Lesson: Obtaining Property Values from Tab Pages

GET_TAB_PAGE_PROPERTY
The GET_TAB_PAGE_PROPERTY built-in function returns the value of the tab page.

Syntax
GET_TAB_PAGE_PROPERTY(tab_page,property);

Example
if GET_TAB_PAGE_PROPERTY('my_page',enabled) = 'FALSE' then
  SET_TAB_PAGE_PROPERTY('my_page',enabled,property_true);
end if;

Exercise 48
Create a user-named trigger called ENABLE_PAGES that will enable the BIO_DEMO and COMMENTS pages. Be sure to check the status to ensure that it's disabled.
Exercise 49
Add calls to these triggers to work in line with ENABLE_KEYS and DISABLE_KEYS.
GET_CANVAS_PROPERTY
The GET_CANVAS_PROPERTY built-in function returns the property of the given canvas.

Syntax
GET_CANVAS_PROPERTY(canvas_id,property);

or
GET_CANVAS_PROPERTY(canvas_name,property);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>canvas_id</td>
<td>The unique id of the canvas. In order to use the canvas id, use the FIND_CANVAS built in to get the unique id.</td>
</tr>
<tr>
<td>canvas_name</td>
<td>The name of the canvas in the object navigator.</td>
</tr>
<tr>
<td>property</td>
<td>The property in which you want returned for the canvas.</td>
</tr>
<tr>
<td></td>
<td>• TOPMOST_TAB_PAGE</td>
</tr>
<tr>
<td></td>
<td>Returns the active tab page on the canvas.</td>
</tr>
<tr>
<td></td>
<td>• FOREGROUND_COLOR</td>
</tr>
<tr>
<td></td>
<td>Returns the foreground color of the canvas.</td>
</tr>
</tbody>
</table>

Exercise 50
When a tab page changes, the cursor should move into the block on the page. Create a form level trigger that will fire when a tab page changes and navigate to the current block.
Introduction
By default, Oracle and Forms messages appear to the user during runtime. You can add your own messages, and replace the default error and warning messages that Forms displays.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- Message types
- Replacing system messages
- Creating and displaying alerts
- Finding and changing property values

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
What Kinds of Messages Are Automatically Displayed? ....................................................204
Building Your Own Messages .............................................................................................205
Built-in Functions that Detect Success and Failure..............................................................206
Triggers That Intercept Messages .......................................................................................207
Alerts ...............................................................................................................................210
Displaying the Alert .............................................................................................................211
Changing the Alert Message ...............................................................................................212
Section Q: Runform Messages and Alerts

Lesson: What Kinds of Messages Are Automatically Displayed?

Displayed messages
Oracle Forms communicates to the operator for:

- **Error messages, such as:**
  - FRM-10212: Login failed for this username and password.
  - ORA-00913: Too many values

- **Warning messages, such as:**
  - FRM-10205: Menu <menu module name> not found.

- **Working messages, such as:**
  - Attempting to reserve record for update or delete (CTRL-C to cancel)...

MESSAGE function
Displays specified text on the message line.

Syntax
MESSAGE('message_string');

To display the value of a variable with text:
MESSAGE('message string'||variable);

Example
IF :twraccd.bill_date > :twraccd.paid_date THEN
    MESSAGE('Bill date cannot be greater than date paid');
END IF;
## Section Q: Runform Messages and Alerts

### Lesson: Built-in Functions that Detect Success and Failure

**Functions**

Functions used to indicate whether the last action in the form was successful.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORM_SUCCESS</td>
<td>TRUE: Action was successful</td>
</tr>
<tr>
<td></td>
<td>FALSE: A error or fatal error occurred</td>
</tr>
<tr>
<td>FORM_FAILURE</td>
<td>TRUE: A nonfatal error occurred</td>
</tr>
<tr>
<td></td>
<td>FALSE: No error or a fatal error occurred</td>
</tr>
<tr>
<td>FORM_FATAL</td>
<td>TRUE: A fatal error occurred</td>
</tr>
<tr>
<td></td>
<td>FALSE: No error or a nonfatal error occurred</td>
</tr>
</tbody>
</table>
Triggers

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON ERROR</td>
<td>Fires instead of displaying the system error message.</td>
</tr>
<tr>
<td>ON MESSAGE</td>
<td>Fires instead of displaying the informative system message.</td>
</tr>
</tbody>
</table>

Intercepting functions

To capture error details, you can use the built-in functions `ERROR_CODE`, `ERROR_TEXT`, `ERROR_TYPE`.

To capture Oracle database errors, you can use the built-in functions `DBMS_ERROR_CODE`, `DBMS_ERROR_TEXT`.

To capture informative message details, you can use the built-in functions `MESSAGE_CODE`, `MESSAGE_TEXT`, `MESSAGE_TYPE`. 
Error-handling system variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM. MESSAGE_LEVEL</td>
<td>Suppresses all message with a severity level at the same or lower then the indicated message level.</td>
</tr>
<tr>
<td></td>
<td>All message, working, and error statements should have a pre-defined severity level. Check the Forms Online help for such documentation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All types of messages from the other levels of severity.</td>
</tr>
<tr>
<td>5</td>
<td>Reaffirms an obvious condition.</td>
</tr>
<tr>
<td>10</td>
<td>Indicates that the operator has made a procedural mistake.</td>
</tr>
<tr>
<td>15</td>
<td>Declares that the operator is attempting to perform function for which the form is not designed.</td>
</tr>
<tr>
<td>20</td>
<td>Indicates condition where the operator cannot continue action due to a problem with a trigger or another outstanding condition.</td>
</tr>
<tr>
<td>25</td>
<td>Indicates a condition that could result in the form performing incorrectly.</td>
</tr>
<tr>
<td>&gt; 25</td>
<td>Severity level too high to suppress with the message_level system variable.</td>
</tr>
</tbody>
</table>
ON-ERROR Example

DECLARE
  errnum NUMBER := ERROR_CODE;
  errtxt VARCHAR2(80) := ERROR_TEXT;
  errtyp VARCHAR2(3) := ERROR_TYPE;
BEGIN
  IF errnum = 40301 THEN
    MESSAGE('Your search criteria identified no matches... Try Again.');
  ELSE
    /*
     ** Print the Normal Message that would have appeared
     **
     ** Default Error Message Text Goes Here     */
    MESSAGE(errtyp||'-'||TO_CHAR(errnum)||': '||errtxt);
    RAISE Form_Trigger_Failure;
  END IF;
END;
Alerts
An alert is a modal window that displays a message notifying the operator of some application condition. The operator must respond to the alert's message by selecting one of the predefined alert buttons. Selecting any button immediately dismisses the alert.

Create an alert
To create an alert:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Navigator, select the Alerts node and then choose Edit→Create.</td>
</tr>
<tr>
<td>2</td>
<td>In the Property Palette window, set the Alert Style property to the style that corresponds to the severity of your message. Valid choices are Stop, Caution, or Note.</td>
</tr>
</tbody>
</table>

At runtime, an icon representing the style you select displays next to the message in the alert window. |
| 3    | Set the Message property by entering the message you want the alert to display at runtime. You can enter up to 200 characters. |
| 4    | Define one or more buttons for the alert by entering a text label in the Button 1, Button 2, and Button 3 fields. (The default text labels are "OK" for Button 1 and "Cancel" for Button 2.) |

At least one button must have a label. Buttons that do not have labels are not displayed. |
| 5    | Choose the Default Alert Button (either Button 1, Button 2, or Button 3). |

The default button is the button that is selected implicitly when the operator presses [Accept]. On most window managers, the default button has a distinctive appearance.
SHOW_ALERT
To display an alert, your application must execute the SHOW_ALERT built-in subprogram from a trigger or user-named subprogram. SHOW_ALERT is a function that returns a numeric constant.

The constant returned by the SHOW_ALERT function indicates which alert button the operator selected and is one of the following: ALERT_BUTTON1, ALERT_BUTTON2, ALERT_BUTTON3.

Example
DECLARE
  alert_button NUMBER;
BEGIN
  IF :twraccd.bill_date > :twraccd.paid_date THEN
    alert_button :=
      SHOW_ALERT('invalid_billing_dates');
  END IF;
END;
Change an alert message

You can change an alert message at runtime by executing the SET_ALERT_PROPERTY built-in procedure. Changing an alert's message allows you to reuse the same alert object, but display a different message each time it is invoked.

For example, it is common to have several alerts in an application that all use the same two buttons (for example, OK and Cancel) but that display different messages depending on application context.

Example

The following example changes the message of an alert named generic_alert. This alert is informational only and has only one button, labeled "OK":

```sql
DECLARE
   alert_button NUMBER;
BEGIN
   IF :twraccd.bill_date > :twraccd.paid_date THEN
      Set_Alert_Property('Date_Alert',
         ALERT_MESSAGE_TEXT,
         'The Bill date cannot be after the date paid.');
      alert_button := SHOW_ALERT('Date_Alert');
   END IF;
END;
```

Exercise 51

Create an alert called HELP_ALERT.

- The alert should be a Note type, with one OK button.
- Briefly describe what the form is used for within the message property.
- Create a button called help that will show the help alert message.
- Assign the canvas to the main canvas Address page.
Section R: Sharing Objects and Code

Lesson: Overview

Introduction
Oracle Forms was designed so that objects and code could be reused and shared.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- Property Classes
- Object Groups
- Copying and Subclassing Objects
- Object Libraries
- SmartClasses
- Modularizing code through program units and libraries

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
Property Classes ...................................................................................................................214
Creating Property Classes .................................................................................................215
Object Groups.......................................................................................................................219
Copying vs. Subclassing.......................................................................................................220
Program Units.......................................................................................................................221
Creating a Visual Attribute.................................................................................................223
Property classes
A property class is a named object that contains a list of properties and their settings. Once you create a property class, you can base other objects on it. An object based on a property class can inherit the setting of any property in the class that makes sense for that object.

Characteristics

- Property class inheritance is a powerful feature that allows you to quickly define objects that conform to your own interface and functionality standards.
- Property classes also allow you to make global changes to applications quickly. By simply changing the definition of a property class, you can change the definition of all objects that inherit properties from that class.
Methods
There are two ways to create a property class:
- You can create a new property class in the Object Navigator and then add properties to it as desired
- You can create a property class from an existing list of properties in the Property Palette window.

Create via Object Navigator
To create a property class in the Object Navigator:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Object Navigator, position the cursor on the Property Classes node and choose Edit→Create, or click the Create icon.</td>
</tr>
<tr>
<td>2</td>
<td>In the Property Palette window, add properties to the class as desired.</td>
</tr>
</tbody>
</table>

Create via Property Palette window
To create a property class in the Property Palette window:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Object Navigator or an editor, select one or more objects so that the Property Palette window displays the properties (and settings) that you want to be in the property class.</td>
</tr>
<tr>
<td>2</td>
<td>In the Property Palette window, click the Property Class icon on the toolbar. An alert is displayed to confirm the name of the class being created. Add a property: Delete a property:</td>
</tr>
</tbody>
</table>
Exercise 52
Create a property class called DATE_CLASS.
- Set the initial value to the database date.
- Set the format mask.
- Set enabled to no.
- Set keyboard navigable to no.
- Set the datatype to type date.
- Set the maximum length to 11.
Inherit property values

To inherit property values from a Property Class:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open up the Property Palette for the item you want to apply the properties from the property class.</td>
</tr>
<tr>
<td>2</td>
<td>Set the Subclass Information Property to the property class you want to use.</td>
</tr>
<tr>
<td>3</td>
<td>Select Property Class.</td>
</tr>
<tr>
<td>4</td>
<td>Select DATE_CLASS.</td>
</tr>
</tbody>
</table>

Screen image

Inherited Property

- Takes its value from the property class that is associated with the object
- Displayed in the Property Palette with an arrow.

Variant Property

- Has modified the inherited value from the property class that is associated with the object
- Displayed in the Property Palette with a cross over the arrow.

Converting Variant to Inherited Properties

To convert a modified value back to the original inherited value, highlight the specific property and click the Inherit Property icon on the Property Palette.
Exercise 53
Subclass all activity dates with the DATE_CLASS.
Object groups
An object group is a container for a group of objects. You define an object group when you want to package related objects so you can copy or reference them in another module.

Object groups provide a way to bundle objects into higher-level building data blocks that can be used in other parts of an application and in subsequent development projects.

Once you create an object group, you can add and remove objects to it as desired.

Create an object group
To create an object group:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Object Navigator, position the cursor on the Object Groups node and choose Edit→Create, or click the Create icon.</td>
</tr>
<tr>
<td>2</td>
<td>In the Object Navigator, drag the desired object(s) under the Object Group Children node.</td>
</tr>
</tbody>
</table>

Moving and copying objects in the navigator
You can use the following commands to move and copy objects within modules in the Navigator:

- To move the current selection, drag it to the desired destination.
- To copy the current selection, Control-drag the selection to the desired location. Alternatively, select Edit→Copy, move the cursor to the desired destination, then select Edit→Paste.
- To cut or copy all objects within a node, position the cursor on the node, then select Edit→Cut or Edit→Copy.
Characteristics
When you drag objects to copy them from one module to another, an alert is displayed that prompts you to specify whether you want to create a copy of the object or create a subclass object.

- Copying creates a new and separate instance of the object in the target module. Any objects owned by the copied object are also copied
- Subclassing creates a new object that has true object-oriented capabilities:
  - Inherits from its parent object
    - This includes changes to the parent object
  - Ability to alter or override inherited properties from the parent object
  - Ability to add new objects to itself that were not inherited.
Program units

If you discover you are using the same code repeatedly within a form, you may want to create a program unit. The unit can then be called from within multiple triggers.

Example

```sql
PROCEDURE populate_detc_desc IS
    BEGIN
        SELECT description
        INTO :twraccd.detc_desc
        FROM twvdetc
        WHERE detc_code = :twraccd.detc_code;
    END;
```

The POST-QUERY trigger just needs to call the program unit:

```sql
/*POST-QUERY */
    populate_detc_desc;
```

The WHEN-VALIDATE-ITEM also calls the program unit, but also validates with the exception handler.

```sql
/* WHEN-VALIDATE-ITEM */
    populate_detc_desc;
    EXCEPTION
    WHEN NO_DATA_FOUND THEN
        MESSAGE('Invalid debt type. Please try again.');
        RAISE FORM_TRIGGER_FAILURE;
```
Exercise 54
Create a non-database display item in the key block called status. Create a procedure that passes in the pidm and sends out the status to determine if the person is a student (SWBSTDN) or an employee (PWBEMPL).

Where should it be called?
Visual attributes

Visual attributes are the font, color, and pattern properties that you set for form and menu objects that appear in your application's interface.

Creating

To create a named visual attribute:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the Object Navigator, position the cursor on the Visual Attributes node.</td>
</tr>
<tr>
<td>2</td>
<td>Choose <strong>Edit→Create</strong>, or click the <strong>+</strong> icon on the toolbar.</td>
</tr>
<tr>
<td>3</td>
<td>In the Properties window, set the font, color, and pattern attributes of the named visual attribute as desired.</td>
</tr>
</tbody>
</table>

Exercise 55

Create another form (stvatyp) based on the stvatyp table’s code and description, displaying 15 records and a scrollbar on a main canvas with *Address Type Codes* for the title.

When the form is opened, it should automatically execute a query. Create a visual attribute group with a prominent background color to highlight the current record.
Introduction
So far, you have run one form at a time to view and modify data. However, rarely would an application consist of one form. In this section, you connect your two forms together by having one call the other.

Intended audience
Programmers with Oracle backgrounds who develop add-on modules for Banner or modify the baseline Banner forms.

Objectives
This section will examine the following:
- Methods of calling one form from another
- How to establish variables that can be used across forms.

Prerequisites
To complete this workbook, you should have:
- equivalent experience navigating in the Banner system
- completed Sungard Higher Education's Introduction to Oracle training
- completed Sungard Higher Education's PL/SQL and Database Objects training.

Section contents
About Multiple-Form Applications.................................................................225
OPEN_FORM ........................................................................................................226
CALL_FORM built-in ..........................................................................................227
NEW_FORM built-in...........................................................................................230
Banner Form Conversion ....................................................................................233
Multiple-form applications
A multiple-form application is one that is designed to open more than one form during a single Runform session.

Every invocation of Runform begins the same way by starting a single form module. Once the first form is loaded into memory and begins execution, it can programmatically invoke any number of additional forms.

How one form finds another
When one form programmatically invokes another, Oracle Forms looks for the new form in the appropriate directory and then loads it into memory.

When you deliver a multiple-form application to end users, all of the .FMX, .MMX, and .PLL (form, menu, and library) files that will be called during the session must reside in the working directory or search path defined for your system. In a 32-bit Windows environment, the Forms Path resides in the registry.

Invoking forms
There are three ways that one form can programmatically invoke another form:

- Execute the OPEN_FORM procedure to open an independent form
- Execute the NEW_FORM procedure to replace the current form with a different form
- Execute the CALL_FORM procedure to call a modal form.
Procedure
The first form remains displayed, and operators can navigate between the forms as desired. The new form can share the same database session as the form from which it was invoked, or it can create a separate session of its own.

Syntax
OPEN_FORM(form_name [,activate_mode] [,session_mode] [,data_mode] [,paramlist_id]);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>form_name</td>
<td>Specifies the name of the form to open.</td>
</tr>
<tr>
<td>activate_mode</td>
<td>• ACTIVATE: Sets focus to the form to make it the active form in the application.</td>
</tr>
<tr>
<td></td>
<td>• NO_ACTIVATE: Opens the form but does not set focus to the form. The current form remains current.</td>
</tr>
<tr>
<td>session_mode</td>
<td>• NO_SESSION: Specifies that the opened form should share the same database session as the current form. A COMMIT operation in any form will cause validation and commit processing to occur for all forms running in the same session.</td>
</tr>
<tr>
<td></td>
<td>• SESSION: Specifies that a new, separate database session should be created for the opened form.</td>
</tr>
<tr>
<td>data_mode</td>
<td>• NO_SHARE_LIBRARY_DATA: At runtime, Form Builder will not share data between forms that have identical libraries attached (At design time).</td>
</tr>
<tr>
<td></td>
<td>• SHARE_LIBRARY_DATA: At runtime, Form Builder will share data between forms that have identical libraries attached (At design time).</td>
</tr>
<tr>
<td>paramlist_name</td>
<td>Specifies the CHAR name of a parameter list to be passed to the opened form.</td>
</tr>
<tr>
<td>paramlist_id</td>
<td>Specifies the unique ID that Oracle Forms assigns to the parameter list at the time it is created. Use the GET_PARAMETER_LIST function to return the ID to a variable of type PARAMLIST.</td>
</tr>
</tbody>
</table>
Section S: Multiple-Form Applications

Lesson: CALL_FORM built-in

Procedure

Runs an indicated form while keeping the parent form active.

Oracle Forms runs the called form with the same Runform preferences as the parent form. When the called form is exited, Oracle Forms processing resumes in the calling form at the point from which you initiated the call to CALL_FORM.

When you call a form, Oracle Forms issues a savepoint for the called form. If the CLEAR_FORM function causes a rollback when the called form is current, Oracle Forms rolls back uncommitted changes to this savepoint.
**Syntax**

```
CALL_FORM(formmodule_name [, display][, switch_menu]
[, query_mode] [, data_mode] [, paramlist_name]);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formmodule_name</td>
<td>Specifies the form module name of the called form.</td>
</tr>
</tbody>
</table>
| display            | • HIDE causes Oracle Forms to clear the calling form from the screen before drawing the called form. HIDE is the default parameter.  
                     • NO_HIDE causes Oracle Forms to display the called form without clearing the calling form from the screen. |
| switch_menu        | • NO_REPLACE causes Oracle Forms to keep the default menu application of the calling form active for the called form.  
                     • DO_REPLACE causes Oracle Forms to replace the default menu application of the calling form with the default menu application of the called form. |
| query_mode         | • NO_QUERY_ONLY causes Oracle Forms to run the indicated form in normal mode, allowing the operator to perform inserts, updates, and deletes from within the called form.  
                     • QUERY_ONLY causes Oracle Forms to run the indicated form in Query Only mode, allowing the operator to query, but not to insert, update, or delete records. |
| data_mode          | • NO_SHARE_LIBRARY_DATA At runtime, Form Builder will not share data between forms that have identical libraries attached (At design time).  
                     • SHARE_LIBRARY_DATA At runtime, Form Builder will share data between forms that have identical libraries attached (At design time). |
| paramlist_id       | Specifies the unique ID Oracle Forms assigns when it creates the parameter list. You can optionally include a parameter list as initial input to the called form. The data type of the ID is PARAMLIST. |
| paramlist_name     | The name you gave the parameter list object when you defined it. |
Example

/*
 ** Built-in: CALL_FORM
 ** Example: Calls a form in query-only mode.
 */
BEGIN
   Call_Form('TWAACCD',NO_HIDE,NO_REPLACE,QUERY_ONLY);
END;
Section S: Multiple-Form Applications

Lesson: NEW_FORM built-in

Procedure
Exits the current form and enters the indicated form. The calling form is terminated as the parent form.

Oracle Forms runs the new form with the same Runform options as the parent form. If the parent form was a called form, Oracle Forms runs the new form with the same options as the parent form.
Syntax

NEW_FORM(formmodule_name [, rollback_mode]
[,query_mode] [, data_mode] [, paramlist_name] );

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formmodule_name</td>
<td>Specifies the form module name of the called form. The name must be enclosed in single quotes. The data type of the name is CHAR.</td>
</tr>
</tbody>
</table>
| rollback_mode   | • TO_SAVEPOINT rolls back all uncommitted changes (including posted changes) to the current form's savepoint.  
• NO_ROLLBACK exits the current form without rolling back to a savepoint.  
• FULL_ROLLBACK rolls back all uncommitted changes (including posted changes) that were made during the current Runform session. |
| query_mode      | • NO_QUERY_ONLY runs the indicated form normally, allowing the operator to perform inserts, updates, and deletes in the form.  
• QUERY_ONLY runs the indicated form as a query-only form. |
| data_mode       | • NO_SHARE_LIBRARY_DATA At runtime, Form Builder will not share data between forms that have identical libraries attached (At design time).  
• SHARE_LIBRARY_DATA At runtime, Form Builder will share data between forms that have identical libraries attached (At design time). |
| paramlist_id    | Specifies the unique ID Oracle Forms assigns when it creates the parameter list. |
| paramlist_name  | The name you gave the parameter list object when you defined it. The data type of the name is CHAR. A parameter list passed to a form via NEW_FORM cannot contain parameters of type DATA_PARAMETER (a pointer to record group). |
Exercise 56
Which of the form invoke methods would be the most useful when you double-click the swraddr atyp code to go to the STVATYP form?

Change the code in the when mouse double click trigger to go to the STVATYP form. Using global variables in both forms, enable the user to double-click on a code from STVATYP and return it to SWAIDEN. Populate the description without using a global variable and move the cursor to the next navigable item.
Section S: Multiple-Form Applications

Lesson: Banner Form Conversion

Reference

This section contains some of the Banner Standards. For a more detailed explanation, please refer to the Banner 7.0 UI Methodology Handbook and the Banner Forms Architecture chapter in the Banner 7.0 General Technical Reference Manual.

* Both documents may be downloaded from Sungard Higher Education's ActionWeb service (connect.sungardhe.com).

Form types

For conversion purposes, forms are divided into three families:

- A **validation form** generally has one navigable Oracle block without a key block
- An **application form** has a key block and at least one navigable Oracle block
- An **inquiry form** does not allow you to create, change or delete records

Configuration

Modify your forms builder settings:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the layout editor for the main_canvas.</td>
</tr>
<tr>
<td>2</td>
<td>Select Edit.</td>
</tr>
<tr>
<td>3</td>
<td>Select Layout Options.</td>
</tr>
<tr>
<td>4</td>
<td>Select Ruler.</td>
</tr>
<tr>
<td>5</td>
<td>Change units to points.</td>
</tr>
<tr>
<td>6</td>
<td>Set horizontal = 6 and Vertical = 17.25.</td>
</tr>
</tbody>
</table>

Change coordinate system

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the property palette of the form, change the <strong>Coordinate System</strong> to <strong>Real</strong>.</td>
</tr>
<tr>
<td>2</td>
<td>Set the width = 6 and Height = 17.</td>
</tr>
</tbody>
</table>
| 3    | Click OK.  
The following message appears:  
*Changing coordinates of entire form. Rounding may occur. Continue?*
| 4    | Click OK. |
Create a new audit trail program unit

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create procedure AUDIT_TRAIL_7_0_TRNG (program unit) with the appropriate comments.</td>
</tr>
</tbody>
</table>

Library attachment

Attach the GOQWFLW library as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Click attached libraries.</td>
</tr>
<tr>
<td>2</td>
<td>Click Create.</td>
</tr>
<tr>
<td>3</td>
<td>Browse for the GOQWFLW.pll, click Attach.</td>
</tr>
<tr>
<td>4</td>
<td>When prompted with <em>Library contains a non-portable directory path, remove path?</em>, click YES.</td>
</tr>
</tbody>
</table>

Note: There should be three libraries: goqrplsl, eoqrplsl and goqwflw.

Form-level trigger changes

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rename WHEN-NEW-BLOCK-INSTANCE trigger to WHEN_NEW_BLOCK_INSTANCE_TRG.</td>
</tr>
<tr>
<td>2</td>
<td>Create a form level user named trigger LOAD_CURRENT_RELEASE set the value of :current_release := '7.0';</td>
</tr>
</tbody>
</table>
### Form level property class

**Banner Form Conversion**

(open up the form goqolib.fmb)

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>If your form is an application form, subclass the object group G$_APPLICATION_FORM from GOQOLIB. In the property palette of the form, set the subclass information to G$_APPL_FORM_CLASS.</td>
</tr>
<tr>
<td>2</td>
<td>If your form is a validation form, subclass the object group G$_VALIDATION_FORM from GOQOLIB. In the property palette of the form, set the subclass information to G$_VAL_FORM_CLASS.</td>
</tr>
<tr>
<td>3</td>
<td>If your form is an inquiry form, subclass the object group G$_INQUIRY_FORM from GOQOLIB. In the property palette of the form, set the subclass information to G$_INQ_FORM_CLASS.</td>
</tr>
</tbody>
</table>

### Key block changes

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>If a form has a key block, in the property palette of the KEY_BLOCK, set the subclass information to G$_KEY_BLOCK_CLASS.</td>
</tr>
</tbody>
</table>

### Toolbars

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subclass the G$_TOOLBAR Object Group from GOQOLIB.</td>
</tr>
</tbody>
</table>
## ID and Name Search

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subclass the G$_IDNAME_SEARCH object group from GOQOLIB.</td>
</tr>
<tr>
<td>2</td>
<td>Subclass the window G$_EXTENDED_SEARCH_WINDOW from GOQOLIB.</td>
</tr>
<tr>
<td>3</td>
<td>Subclass the G$_SEARCH object group from GOQOLIB.</td>
</tr>
<tr>
<td>4</td>
<td>In the Object Navigator, check the order of the ID and name items. The name item must be positioned right after the ID item.</td>
</tr>
<tr>
<td>5</td>
<td>In the property palette of the ID, set the subclass information to G$_ID_CLASS.</td>
</tr>
<tr>
<td>6</td>
<td>In the property palette of the name, set the subclass information to G$_NAME_CLASS. Make sure the visual attribute on the name item is G$_NVA_TEXT_ITEM.</td>
</tr>
<tr>
<td>7</td>
<td>Create a push button in the KEY_BLOCK called ID_LBT. In the property palette of the ID list button, set the visual attribute to default and set the subclass information to G$_FLASHLITE_BTN_CLASS. Place on the main canvas.</td>
</tr>
<tr>
<td>8</td>
<td>Create a non-database item in the BUTTON_CONTROL_BLOCK called CHECK_KEYS, maximum length 1.</td>
</tr>
<tr>
<td>9</td>
<td>Any person form, such as SOAIDEN, must be executed by the trigger KEY-LISTVAL. Any non-person form such as SOACOMP must be executed by KEY-CQUERY. If the ID has the above triggers, the auto hint text must contain the words LIST to reflect the trigger KEY-LISTVAL and COUNTHITS to reflect the trigger KEY-CQUERY.</td>
</tr>
</tbody>
</table>
**Code/ Description lookup**

A code item that has a button that calls an LOV, a validation form, or an Option List window that contains an LOV or a validation form is converted using the following steps.

The code item may or may not have a description item associated with it. If the code item has an associated description item, the description item must have the same y coordinate as the code item, and it should be placed after the code in the Object Navigator.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subclass G$ _SEARCH object group from GOQOLIB, if not already referenced.</td>
</tr>
<tr>
<td>2</td>
<td>In the code item’s property sheet, subclass G$ _CODE_CLASS.</td>
</tr>
<tr>
<td>3</td>
<td>If the form is a query-only form (the third letter of the form name is I or Q), change the Update Allowed property to FALSE.</td>
</tr>
</tbody>
</table>
| 4    | If the description item is a database item, do not attach a property class to the description. Otherwise, make these changes in the description item property sheet:  
  - Subclass G$ _DESC_CLASS.  
  - Change the Visual Attribute to Default. |
### Procedure, continued

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 5    | If the code item has an associated LOV that conforms to all of the following standards, no item specific modifications are required.  
  LOV is named like tablename_LOV.  
  LOV table name has columns named tablename_CODE and tablename_DESC.  
  LOV’s associated record group does not have a WHERE clause.  
  If the above conditions are not met, then you must modify the user-named trigger G$_SEARCH_PARAMETERS at the code item level. This trigger exists as part of applying the G$_CODE_CLASS to the code item. This trigger has the following code:  
  
  ```g$_search.parameters('param1','param2','param3');
  ```  
  where  
  param1 = database column name from the validation table for code  
  param2 = database column name from the validation table for description  
  param3 = WHERE clause from the record group associated with the LOV (optional) |
| 6    | Subclass the property class G$_ICON_BTN_CLASS from GOQOLIB. |
| 7    | Create push buttons for each code item. The button must have the same name as the item with “_LBT” at the end of the button name. Subclass the button with g$_icon_btn_class and set the property icon filename to search.  
  Place the button directly to the right of the code item before the description. Place the button on the canvas. |
## Windows and Canvases

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subclass the main canvas with G$_FS_CANVAS_CLASS.</td>
</tr>
<tr>
<td>2</td>
<td>Subclass the main_window with G$_FS_WINDOW_CLASS.</td>
</tr>
</tbody>
</table>

## Date items

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>For all date datatype items, in the property palette assign the subclass G$_DATE_CLASS.</td>
</tr>
<tr>
<td>2</td>
<td>For enterable date items, create a date button. The button must have the same name as the item with &quot;_DBT&quot; at the end of the button name. Subclass G$_CALENDAR_BTN_CLASS.</td>
</tr>
</tbody>
</table>

## Appearance on Canvases

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The data separator line is to be placed above the data to be grouped in your display. Create an item in the FORM_HEADER_BLOCK</td>
</tr>
<tr>
<td>2</td>
<td>Set the subclass information to G$_DATA_SEPARATOR_LINE_CLASS.</td>
</tr>
<tr>
<td>3</td>
<td>Change the item name to DATA_SEPARATOR_LINE_&lt;num&gt;. This is where num is incremented for each data separator line that is needed in your form (i.e. DATA_SEPARATOR_LINE_1 will be the first data separator line in the form, DATA_SEPARATOR_LINE_2 will be the second, and so forth).</td>
</tr>
<tr>
<td>4</td>
<td>Change the width if a full-size bar is not needed (full-size bar is the default when applying the G$_DATA_SEPARATOR_LINE_CLASS class).</td>
</tr>
</tbody>
</table>
Section S: Multiple-Form Applications

Lesson: Banner Form Conversion

(Continued)

Procedure, continued

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Once the data separator line is in place, you can then draw the non-beveled gray box to complete the data grouping graphic. To draw the box (if it does not already exist), follow steps 6-11 below.</td>
</tr>
<tr>
<td>6</td>
<td>Select <em>Rectangle</em> in the toolbar.</td>
</tr>
<tr>
<td>7</td>
<td>Start drawing the box immediately under the left side of the data separator line and ending a few pixels away from the right side of the window/canvas edge.</td>
</tr>
<tr>
<td>8</td>
<td>With the line still selected, select <em>Line Color</em> in the toolbar and select the gray color (first color on second row from the top row).</td>
</tr>
<tr>
<td>9</td>
<td>Select <em>No Fill</em>.</td>
</tr>
<tr>
<td>10</td>
<td>Select <em>Layout</em> from the Menu. Select <em>Bevel</em> and <em>None</em>.</td>
</tr>
<tr>
<td>11</td>
<td>Select <em>Layout</em> from the Menu. Select <em>Line Width</em> of 1 pt. The Y position of the data separator line should be five positions less than the Y position of the non-beveled gray box. You should be able to see a very small (about a pixel) amount of space between the data separator line and the gray box.</td>
</tr>
<tr>
<td>12</td>
<td>All prompts and boilerplates are now Verdana, 9-point bold.</td>
</tr>
</tbody>
</table>

Order of data blocks

The order of data blocks is as follows:
- form header
- key block
- data blocks
- all referenced blocks

Renaming exit form

Rename any instance of exit_form to g$$_goqolib_key_trigger.b2k_exit_form.

Generating the form

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Save and close the form.</td>
</tr>
<tr>
<td>2</td>
<td>Reopen the form.</td>
</tr>
<tr>
<td>3</td>
<td>Using the Oracle Forms 10g compiler, generate the form.</td>
</tr>
</tbody>
</table>
Section T: Exercises Answer Key

Lesson: Overview

Introduction
This section contains answer keys for the exercises found in this workbook.

Section contents
Answer Key  242
Exercise 1
Create a new form module called SWAIDEN. The naming convention follows Banner standards. SWAIDEN stands for:

- Student
- Custom object
- Application form
- Identification

Select File->New->Form. Double-click the form name to rename it to SWAIDEN.

Exercise 2
In the new form module, create a data block based on the SWRIDEN table. Do not include the swriden_change_ind, swriden_activity_date, swriden_user_id and swriden_data_origin columns. Display tabular style, and put everything on a new canvas later to be renamed main_canvas.

In the new form, select Tools->Data Block Wizard. When the dialog appears, enter the following properties:

- Base Table: SWRIDEN
- Select all columns except swriden_change_ind and swriden_activity_date
- Canvas: (New Canvas)
- Display all columns
- Make meaningful prompt names
- Display Style: Tabular
- Frame Name: Identification
- Distance between records: 0
- Records displayed: 1

Exercise 3
Rename canvas1 to main_canvas.

Double-click canvas1 and rename it to main_canvas.
Exercise 4
Rename window1 to main_window and give it a title.

Double-click window1 and rename it to main_window. In the Property Palette under Functional, title the form Identification Form.

Exercise 5
In the SWRIDEN block, increase the number of records displayed to 5, then add a scroll bar.

Block property: records displayed 5

Exercise 6
In the ID data block, allow the user to automatically navigate to the next record when tabbing, by modifying the Navigation Style on the data block level.

Navigation Style: Change record

Exercise 7
Ensure that the records retrieved in the SWRIDEN data block are current (the swriden change_ind is NULL) by adding a WHERE clause.

Block property where clause: swriden_change_ind is null

Exercise 8
Sort the records in the SWRIDEN block by last name.

Block property order by clause: swriden_last_name

Exercise 9
Make sure the user is unable to insert, update, or delete records in the SWRIDEN block.

Block properties insert, update and delete NO

Exercise 10
Make sure that the swriden_pidm is disabled and not keyboard-navigable.

Block properties enabled NO, keyboard navigable NO
Exercise 11
Make sure that swraddr_pidm and swraddr_activity_date are both disabled and not keyboard navigable.

Block properties enabled NO, keyboard navigable NO

Exercise 12
Using whatever method you choose, create a data block based on the SWBPERS table.
- Include all columns except the swbpers data origin and swbpers user id
- The swbpers pidm and swbpers activity date are not enabled and keyboard navigable
- Do NOT create a master-detail relationship

This solution is based on the previous exercises.

Exercise 13
Join both the swraddr and swbpers blocks to the swriden_pidm.

Block properties

```
SWRADDR where clause:
    swraddr_pidm = :swriden_pidm

SWBPERS where clause:
    swbpers_pidm = :swriden_pidm
```
Exercise 14
In both the SWRADDR and SWBPERS data blocks:

- Alter the activity dates so that it initializes to the current database date for a new record. Try using an intersection to set the property for both items at the same time.

- In both data blocks, do not allow the activity dates to be changed by the user.

- Set the bubble help to Activity Date.

- Set the format mask so that the date appears like the following:
  01-JAN-1998

  *Remember to alter the maximum length to allocate for the increase in characters*.

To create an intersection: within the Object Navigator, highlight activity date in both the Address and Person data blocks.

To highlight items, click the first item, and then hold the [Ctrl] key and click the second item.

Enter the Property Palette by selecting Tools_Property Palette.
Set the following properties:

- Format Mask: DD-MON-YYYY
- Initial Value: $$DBDATE$$
- Maximum Length: 11
- Enabled: No
- Keyboard Navigable: No
Exercise 15
In the Person data block, set the format mask for swbpers_birth_date so that it appears like the following:

01-JAN-1998

Set the bubble help to Birth Date.

Format Mask: DD-MON-YYYY
Maximum Length: 11
Tooltip: Birth Date

Exercise 16
In the Person data block, set the format mask for SSN so that it appears like the following at runtime:

123-45-6789

Format Mask: 999"-“99”-“9999
Maximum Length: 11

Exercise 17
In the Address data block, set the format mask for the phone number so that it appears like the following at runtime:

555-1212

Format Mask: 999"-“9999
Maximum Length: 8
Exercise 18
In the Address data block, create two new display items named atyp_desc and state_desc.
- Ensure that they are both not base table items.
- In the Object Navigator, the atyp_desc should be under the swraddr_atyp_code. The state_desc should be under the swraddr_stat_code.
- Assign the items to the main_canvas.
- The display items will be populated by a trigger in a later exercise.
- On the canvas, place the atyp_desc to the right of the swraddr_atyp_code and the state_desc to the right of the swraddr_stat_code.

In the Address data block, highlight the Items node and click the icon twice.

Rename one item to atyp_desc and the other to state_desc. Drag the atyp_desc under swraddr_atyp_code and state_desc under swraddr_stat_code.

In the property sheet for both, set the following properties:
- Item Type: Display Item
- Canvas: main_canvas
- Database Item: No

In the layout editor, place the atyp_desc to the right of the swraddr_atyp_code and the state_desc to the right of the swraddr_stat_code.

Exercise 19
Create a new data block with all columns from the SWRCMNT table (no master-detail relationship) on a new canvas called Comments. Join this block to the swriden data block. The pidm, user id and activity date are not enabled or keyboard navigable.

Create a display item cmtt_desc. The activity date should have an initial value and a format mask.

The solution to this is based on all of the previous exercises.
Exercise 20
In the Person data block, convert the swbpers_confid_ind text item to a check box.
- Set the checked state to represent the base table value of Y and the unchecked state to represent N.
- Ensure that new records are automatically assigned the value N.
- Resize the checkbox appropriately.
- Label the check box Confidential? and remove the prompt.

Set the following properties for swbpers_confid_ind:
- Item Type: Check Box
- Initial Value: N
- Label: Confidential?
- Value When Checked: Y
- Value When Unchecked: N
- Check Box Mapping of Other Values: Not Allowed

Exercise 21
In the Person data block, convert the swbpers_mrtl_code text item to a pop-list list item.
- Add list elements of Single, Married, Widowed, and Divorced to represent database values of S, M, W, and D.
- Display any other values as Single.
- Ensure that new records display the default value Single.
- Resize the list item to see your choices at runtime.

Within the swbpers_mrtl_code Property Palette, set the following properties:
- Item Type: List Item
- Initial Value: S
- Mapping of Other Values: S
- List Style: Poplist

Double-click List Elements, and enter:

<table>
<thead>
<tr>
<th>List Element</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>S</td>
</tr>
<tr>
<td>Married</td>
<td>M</td>
</tr>
<tr>
<td>Widowed</td>
<td>W</td>
</tr>
<tr>
<td>Divorced</td>
<td>D</td>
</tr>
</tbody>
</table>
Exercise 22
In the Person data block, convert the swbpers_sex text item into a radio group.
- Add radio buttons for Male, Female and Other to represent the database values of M, F and O.
- Define access keys of M for male, F for female, and O for Other.
- Define a default value of F for all new records.

Enter the following properties for the swbpers_sex item:
- Item Type: Radio Group
- Initial Value: M

Create two radio buttons underneath the group, and set the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Male</th>
<th>Female</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Key</td>
<td>M</td>
<td>F</td>
<td>O</td>
</tr>
<tr>
<td>Label</td>
<td>Male</td>
<td>Female</td>
<td>Other</td>
</tr>
<tr>
<td>Radio Button Value</td>
<td>M</td>
<td>F</td>
<td>O</td>
</tr>
</tbody>
</table>
Exercise 23
Create a non-database data block called button_control_block.

Create four non-database items and convert all of them to push buttons:

- Button 1: Comments
  - Label: Comments.
  - Add trigger when-button-pressed for navigation to the Comments block. (HINT: Use the built in navigation sub program go_block('blockname');).
  - Place on the main canvas.

- Button 2: Home
  - Label: Home.
  - Add trigger when-button-pressed for navigation to the Address block.
  - Place on the comments canvas.

- Button 3: Exit
  - Label: Exit.
  - Add trigger when-button-pressed and enter exit_form;
  - Place on the main_canvas.

- Button 4: Save
  - Label: Save.
  - Add trigger when-button-pressed and enter commit_form;
  - Place on the main_canvas.

- Highlight data blocks and click the icon.
- Rename the block button_control. In the Property Palette, change the setting database data block to NO.
- Add three text items and change the item types to push button.

Button 1
- Item name: COMMENTS
- Label: Comments
- Add trigger when-button-pressed and enter go_block('SWBPERS');
- Place on main_canvas
Exercise 23 (cont.)

Button 2
- Item name: HOME
- Label: Home
- Add trigger when-button-pressed and enter go_block('SWRADDR');
- Place on comments canvas

Button 3
- Item name: EXIT
- Label: Exit
- Add trigger when-button-pressed and enter exit_form;
- Place on main_canvas

Button 4
- Item name: SAVE
- Label: Save
- Add trigger when-button-pressed and enter commit_form;
- Place on main_canvas

Exercise 24
At the form level, create a trigger to immediately execute a query in all blocks when the form is opened. The cursor should remain in the SWRIDENT block.

Create a form level trigger WHEN-NEW-FORM-INSTANCE.
Use either the code under Code A or Code B below.
**Either set of code will work for this exercise**

<table>
<thead>
<tr>
<th>Code A</th>
<th>Code B</th>
</tr>
</thead>
<tbody>
<tr>
<td>execute_query;</td>
<td>execute_query;</td>
</tr>
<tr>
<td>next_block;</td>
<td>go_block('swraddr');</td>
</tr>
<tr>
<td>execute_query;</td>
<td>execute_query;</td>
</tr>
<tr>
<td>next_block;</td>
<td>go_block('swbpers');</td>
</tr>
<tr>
<td>execute_query;</td>
<td>execute_query;</td>
</tr>
<tr>
<td>next_block;</td>
<td>go_block('swcmnt');</td>
</tr>
<tr>
<td>execute_query;</td>
<td>execute_query;</td>
</tr>
<tr>
<td>previous_block;</td>
<td>go_block('swrident');</td>
</tr>
<tr>
<td>previous_block;</td>
<td>previous_block;</td>
</tr>
<tr>
<td>previous_block;</td>
<td>previous_block;</td>
</tr>
</tbody>
</table>
Exercise 25
When navigating to the next record in the ID block, get the records in the address block to change as well.

Create a trigger on the SWRIDEN block called WHEN-NEW-RECORD-INSTANCE and use either set of code below:

<table>
<thead>
<tr>
<th>Code A</th>
<th>Code B</th>
</tr>
</thead>
<tbody>
<tr>
<td>next_block;</td>
<td>go_block('swraddr');</td>
</tr>
<tr>
<td>execute_query;</td>
<td>execute_query;</td>
</tr>
<tr>
<td>next_block;</td>
<td>go_block('swbpers');</td>
</tr>
<tr>
<td>execute_query;</td>
<td>execute_query;</td>
</tr>
<tr>
<td>next_block;</td>
<td>go_block('swrcmnt');</td>
</tr>
<tr>
<td>execute_query;</td>
<td>execute_query;</td>
</tr>
<tr>
<td>previous_block;</td>
<td>go_block('swrcmnt');</td>
</tr>
<tr>
<td>previous_block;</td>
<td>go_block('swrdden');</td>
</tr>
<tr>
<td>previous_block;</td>
<td></td>
</tr>
</tbody>
</table>
Exercise 26
Create post-query triggers on the SWRADDR and SWRCMNT blocks to populate the atyp_desc, state_desc, and cmtt_desc display items.

**Block: SWRADDR**
**Trigger: POST-QUERY**
**PL/SQL Code:**
```plsql
begin
  select stvatyp_desc into :atyp_desc
  from stvatyp
  where stvatyp_code = :swraddr_atyp_code;
end;

begin
  select stvstat_desc into :state_desc
  from stvstat
  where stvstat_code = :swraddr_stat_code;
end;
```

**Block: SWRCMNT**
**Trigger: POST-QUERY**
**PL/SQL Code:**
```plsql
begin
  select stvcmtt_desc into :cmtt_desc
  from stvcmtt
  where stvcmtt_code = :swrcmnt_cmtt_code;
end;
```
Exercise 27
In SQL+, query the stvatyp table and look at the values of the stvatyp_code.

Query the swraddr table and look at the values of the swraddr_atyp_code.

Run your form.

Enter an atyp code, what happens? (Do NOT save the changes)

What item level changes need to be made?

The atyp code values in both tables are in upper case. The swraddr_atyp_code, swraddr_stat_code and swrcmnt_cmtt_code should have a case restriction set in the property palette of UPPER.
Exercise 28
In the SWRADDR block, create a trigger to populate the address description item whenever validation occurs on swraddr_atyp_code (use STVATYP).

Fail the trigger and display a suitable message if the swraddr_atyp_code is not found.

Run the form, and enter a new address with an incorrect address type. Enter a correct address type to see if the display item populates.

Block: SWRADDR
Item: SWRADDR_ATYP_CODE
Trigger: WHEN-VALIDATE_ITEM
PL/SQL Code:
begin
    select stvatyp_code into :atyp_desc
    from stvatyp
    where stvatyp_code = :swraddr_atyp_code;
    exception
    when no_data_found then
        message('Invalid Address Code!');
        message('Invalid Address Code!');
        raise form_trigger_failure;
end;
**Exercise 29**

In the SWRADDR block, create a trigger to populate the state description whenever validation occurs on swraddr_stat_code (use STVSTAT).

Fail the trigger and display a suitable message if the swraddr_stat_code is not found.

Run the form, and enter a new address with an incorrect state code. Enter a correct state code to see if the display item populates.

**Block: SWRADDR**

**Item: SWRADDR_STAT_CODE**

**Trigger: WHEN-VALIDATE_ITEM**

**PL/SQL Code:**

```plsql
declare
  cursor c1 is
    select stvstat_code
    from stvstat
    where stvstat_code = :swraddr_stat_code;
begin
  open c1;
  fetch c1 into :swraddr.stat_desc;
exception
  when no_data_found then
    message('Invalid State Code!');
    message('Invalid State Code!');
    raise form_trigger_failure;
  close c1;
end;
```
Exercise 30
In the SWRCMNT block, create a trigger to populate the comment description whenever validation occurs on swrcmnt_cmtt_code (use STVCMTT).

Fail the trigger and display a suitable message if the swrcmnt_cmtt_code is not found.

Run the form, and enter a new address with an incorrect comment code. Enter a correct comment code to see if the display item populates.

Block: SWRCMNT
Item: SWRCMNT_CMTT_CODE
Trigger: WHEN-VALIDATE_ITEM
PL/SQL Code:
begin
    select stvcmtt_code into :cmtt_desc
    from stvcmtt
    where stvcmtt_code = :swrcmnt_cmtt_code;

    exception
    when no_data_found then
        message('Invalid Comment Code!');
        message('Invalid Comment Code!');
        raise form_trigger_failure;
end;
Exercise 31

Comment out the code in the following triggers:

- WHEN-NEW-FORM-INSTANCE (form level)

- WHEN-NEW-RECORD-INSTANCE (swriden block level)
Exercise 32
Rename the swriden block to a non-database block called KEY_BLOCK. (Display only one record and remove the scrollbar.)

In the Property Palette,
- change the Name property to KEY_BLOCK
- change the database data block to NO
- delete query data source name
- change number of records displayed to 1
- change display scrollbar to NO
- delete where clause

Rename the following items and make them non-database items:
- swriden_pidm to pidm (Don’t forget to change all references from the swriden_pidm to pidm).

In the Property Palette,
- change the Name property to PIDM
- change the database data item to NO
- delete column name
- change where clauses on all blocks

- swriden_id to id (Don’t forget to change all references from the swriden_id to id)

In the Property Palette,
- change the Name property to ID
- change the database data item to to NO
- delete column name
- change the post query trigger on the SWRIDEN block
Exercise 32 (continued)

- swriden_last_name to name

In the Property Palette,
- change the Name property to NAME
- delete column name
- change the database data item to NO

Delete the swriden_first_name and swriden_mi.

Delete these two items from the Object Navigator.

Create a validation trigger for the ID. If the ID entered is valid, populate the pidm and name; otherwise, fail the trigger and display a suitable message.

Block: KEY_BLOCK
Item: ID
Trigger: WHEN-VALIDATE_ITEM
PL/SQL Code:
begin
    select swriden_pidm, swriden_last_name
    ||', '||swriden_first_name
    ||' '||swriden_mi
    into :pidm, :name
    from swriden
    where swriden_id  = :key_block.id
    and swriden_change_ind is null;
except
    when no_data_found then
        message('Invalid ID!');
        message(' ',no_acknowledge);
        raise form_trigger_failure;
end;
Exercise 32 (continued)
Create an appropriate key trigger that will go to each block, execute a query and return the cursor to the address block.

Block: KEY_BLOCK
Trigger:  KEY-NXTBLK
PL/SQL Code:
   next_block;
   execute_query;
   next_block;
   execute_query;
   next_block;
   execute_query;
   previous_block;
   previous_block;

Make sure that the block level properties allow you to enter an ID.

Block properties Insert, Update, Delete allowed changed to yes.

Exercise 33
Try to insert a record into the swraddr block. What happened?

You may receive one of the following errors:
   a. Disabled item swraddr.swraddr_pidm failed validation

          Oracle is attempting to validate the record before it inserts. The swraddr_pidm is required and therefore must contain a value before it inserts a record.

   b. Oracle error. Unable to INSERT record.
Exercise 34
If you got the following message:
   Disabled item swraddr.swraddr_pidm failed validation

...what should you do? (Remember, end users do not see or have access to pidms.)

In the Property Palette for the swraddr_pidm, set required to NO.

Try to insert a record again. What happened?

IF you get the following error:
   Oracle error. Unable to INSERT record.
On the main toolbar, click HELP-DISPLAY ERROR
It should say "unable to insert NULL into owner.swraddr.swraddr_pidm".

Create a swraddr block level pre-insert trigger that puts the current value of the key block pidm into the current value of the swraddr pidm.

Block: SWRADDR
Trigger: PRE-INSERT
PL/SQL Code:
   :swraddr_pidm := :key_block.pidm;

Exercise 35
Create pre-insert triggers for the other two blocks.

Block: SWBPERS
Trigger: PRE-INSERT
PL/SQL Code:
   :swbpers_pidm := :key_block.pidm;

Block: SWRCMNT
Trigger: PRE-INSERT
PL/SQL Code:
   :swrcmnt_pidm := :key_block.pidm;
Exercise 36
Create a pre-update trigger that updates the activity date in the SWBPERS, SWRADDR and SWRCMNT blocks. The SWRCMNT block should also update the user id using the keyword `user`.

**Block: SWRADDR**
**Trigger:** PRE-UPDATE
**PL/SQL Code:**
```plsql
:swraddr_activity_date := sysdate;
```

**Block: SWBPERS**
**Trigger:** PRE-UPDATE
**PL/SQL Code:**
```plsql
:swbpers_activity_date := sysdate;
```

**Block: SWRCMNT**
**Trigger:** PRE-UPDATE
**PL/SQL Code:**
```plsql
:swrcmnt_activity_date := sysdate;
:swrcmnt_user_id := user;
```

Exercise 37
Create a form level user-named trigger called `enable_keys` that enables and allows navigation of the key block items.

Use `get_item_property` to check the status of the ID and Name before enabling.

**Form Level**
**Trigger:** (user-named) ENABLE_KEYS
**PL/SQL Code:**
```plsql
if get_item_property('key_block.id',enabled) = 'FALSE' then
    set_item_property('key_block.id',enabled,property_true);
    set_item_property('key_block.id', navigable, property_true);
end if;

if get_item_property('key_block.name',enabled) = 'FALSE' then
    set_item_property('key_block.name',enabled,property_true);
    set_item_property('key_block.name', navigable, property_true);
end if;
```
Exercise 38
Create a form level user-named trigger called `disable_keys` that disables the key block ID and Name.

**Form Level**
**Trigger:** (user-named) DISABLE_KEYS
**PL/SQL Code:**
```
set_item_property('key_block.id',enabled,property_false);
set_item_property('key_block.name',enabled,property_false);
```

Exercise 39
Create a form level when new block instance trigger. Using system variables, check for the cursor location. If it is not in the key block, execute the disable keys trigger.

**Form Level**
**Trigger:** WHEN-NEW-BLOCK-INSTANCE
**PL/SQL Code:**
```
if :system.cursor_block != 'KEY_BLOCK' THEN
  execute_trigger('disable_keys');
end if;
```
Exercise 40
Create a button called clear, that when pressed will

- execute a user named trigger called save keys that saves the key block id into a global variable called key_idno,
- enable the key block,
- clear the form using the clear_form built-in function,
- executes another user named trigger called global copy that will assign the global key idno back into the key block id, and
- validates the id to populate the name.

Return the cursor to the key block.

Block: BUTTON_CONTROL_BLOCK
Item: Clear
Item Type: Push Button
Label: Clear
Trigger: WHEN-BUTTON-PRESSED
PL/SQL Code:
```plsql
execute_trigger('save_keys');
execute_trigger('enable_keys');
clear_form;
execute_trigger('global_copy');
go_block('key_block');
```

Form Level
Trigger: SAVE_KEYS
PL/SQL Code:
```plsql
:global.key_idno := :key_block.id;
```

Form Level
Trigger: GLOBAL_COPY
PL/SQL Code:
```plsql
if :global.key_idno is not null then
  :key_block.id := :global.key_idno;
  next_item;
  previous_item;
end if;
```
Exercise 41
Rename the LOV and Record Group.

In the Property Palette of the swraddr_atyp code, set validate from list to yes. Comment out the when validate item trigger.

In the Property Palette of the Record Group, change the name to STVATYP_RG. In the Property Palette of the LOV, change the name to STVATYP_LOV. In the Property Palette of the swraddr_atyp_code in the LOV section, change validate from list to yes.

Block: SWRADDR
Item: swraddr_atyp_code
Trigger: WHEN-VALIDATE_ITEM
PL/SQL Code:
/*select stvatyp_desc
   into :atyp_desc
   from stvatyp
   where stvatyp_code = :swraddr_atyp_code;

   exception
     when no_data_found then
       message('Invalid Address Type Code!');
       message('Invalid Address Type Code!');
       raise form_trigger_failure; */

null;

Exercise 42
Create another LOV for swrcmtt_cmtt_code. In the Property Palette, set the Validate from List to yes and comment out the When-Validate-Item trigger.

Use one of the previously discussed methods to create this LOV.
Exercise 43
Under each of the code items, create a mouse double-click trigger. Either use the show_lov built in to display the LOV or the do_key built in.

Block: SWRADDR
Item: swraddr_atyp_code
Trigger: WHEN-MOUSE-DOUBLE-CLICK
PL/SQL Code:
   do_key('list_values');

Block: SWRADDR
Item: swraddr_stat_code
Trigger: WHEN-MOUSE-DOUBLE-CLICK
PL/SQL Code:
   declare
      misc   boolean;
   begin
      misc := show_lov('stvstat_lov');
   end;

Block: SWCMNT
Item: swrcmnt_cmtt_code
Trigger: WHEN-MOUSE-DOUBLE-CLICK
PL/SQL Code:
   do_key('list_values');

Exercise 44
Convert the comments canvas to a stacked canvas. Adjust the viewport x and y positions to make the canvas pop up in a nice place.

- Viewport X position  10
- Viewport Y position  10
- Viewport width       350
- Viewport height      150
Exercise 45
Save your form as swaiden_tab.fmb.

File → Save as → swaiden_tab

Exercise 46
Convert the main canvas to a tab canvas.

Create three new pages: Address, Bio/Demo and Comments.

- Move items on the swraddr block to the comments canvas Address page.
- Move items on the swbpers block to the comments canvas Bio/Demo page.
- Move items on the comment block to the comments canvas Comments page.

- Highlight comments, in the property palette change canvas type to tab.
- Expand the comments node in the object navigator, highlight tab pages and add three tab pages by clicking the + icon three times.
- Rename the first tab page ADDR. In the property palette, change the label to Address.
- Rename the second tab page BIO_DEMO. In the property palette, change the label to Bio/Demo.
- Rename the third tab page COMMENTS. In the property palette, change the label to Comments.
- Select the items in each block and assign them to the tab page in the property palette

Exercise 47
Create a user-named trigger called DISABLE_PAGES that will disable the BIO_DEMO and COMMENTS pages.

Form Level
Trigger: user-name (renamed to) DISABLE_PAGES
PL/SQL Code:

set_tab_page_property('bio_demo',enabled,property_false);
set_tab_page_property('comments',enabled,property_false);
Exercise 48
Create a user-named trigger called ENABLE_PAGES that will enable the BIO_DEMO and COMMENTS pages. Be sure to check the status to ensure that it's disabled.

Form Level
Trigger: user-name (renamed to) ENABLE_PAGES
PL/SQL Code:
```plsql
if get_tab_page_property('bio_demo', enabled) = 'FALSE' then
    set_tab_page_property('bio_demo', enabled, property_true);
end if;
if get_tab_page_property('comments', enabled) = 'FALSE' then
    set_tab_page_property('comments', enabled, property_true);
end if;
```

Exercise 49
Add calls to these triggers to work in line with ENABLE_KEYS and DISABLE_KEYS.

Block: BUTTON_CONTROL_BLOCK
Item: Clear
Item Type: Push Button
Label: Clear
Trigger: WHEN-BUTTON-PRESSED
PL/SQL Code:
```plsql
execute_trigger('save_keys');
execute_trigger('enable_keys');
execute_trigger('disable_pages');
clear_form;
execute_trigger('global_copy');
go_block('key_block');
```

Form Level
Trigger: WHEN-NEW-BLOCK-INSTANCE
PL/SQL Code:
```plsql
if :system.cursor_block != 'KEY_BLOCK' THEN
    execute_trigger('disable_keys');
    execute_trigger('enable_pages');
end if;
```
Exercise 50
When a tab page changes, the cursor should move into the block on the page. Create a form level trigger that will fire when a tab page changes and navigate to the current block.

```
WHEN-TAB-PAGE-CHANGED
    declare
    tab_topmost VARCHAR2(30) :=
    GET_CANVAS_PROPERTY('COMMENTS',TOPMOST_TAB_PAGE);
    BEGIN
    if tab_topmost = 'ADDRESS' then
        go_block('SWRADDR');
    elsif
        tab_topmost = 'BIO_DEMO' then
            go_block('SWBPERS');
    elsif
        tab_topmost = 'COMMENTS' then
            go_block('SWRCMNT');
    end if;
    END;
```

Exercise 51
Create an alert called HELP_ALERT.

- The alert should be a Note type, with one OK button.
- Briefly describe what the form is used for within the message property.
- Create a button called help that will show the help alert message.
- Assign the canvas to the main canvas Address page.

- Highlight alerts
- Click the Create icon
- Title the alert Help
- ID, address and personal information form
- Create an item in the control block and change the type to push button
- Create a when-button-pressed trigger on the button and add:
```
declare
    alert_button number;
    begin
        alert_button := show_alert('HELP');
        end;
```
Exercise 52
Create a property class called DATE_CLASS.

- Set the initial value to the database date.
- Set the format mask.
- Set enabled to no.
- Set keyboard navigable to no.
- Set the datatype to type date
- Set the maximum length to 11.

Add the following properties with the values:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial value</td>
<td>$$dbdbate$$</td>
</tr>
<tr>
<td>Format mask</td>
<td>DD-MON-YYYY</td>
</tr>
<tr>
<td>Enabled</td>
<td>No</td>
</tr>
<tr>
<td>Keyboard Navigable</td>
<td>No</td>
</tr>
<tr>
<td>Datatype</td>
<td>Date</td>
</tr>
<tr>
<td>Maximum Length</td>
<td>11</td>
</tr>
</tbody>
</table>

Exercise 53
Subclass all activity dates with the DATE_CLASS.

Select swraddr_activity_date, swbpers_activity_date and swrcmnt_activity_date. In the Property Palette, set the subclass information to the property class DATE_CLASS.
Exercise 54
Create a non-database display item in the key block called status. Create a procedure that passes in the pidm and sends out the status to determine if the person is a student (SWBSTDN) or an employee (PWBEMPL).

Where should it be called?

Highlight items under the KEY_BLOCK, click the + icon.
Rename it to STATUS.
Properties:
- item type ➔ display
- database item ➔ no
- maximum length ➔ 20
- canvas ➔ MAIN_CANVAS

PROCEDURE get_status(pidm_var in number , status_var in out varchar2) is
  cursor emp_cursor is
    select 'Employee'
    from pwbempl
    where pwbempl_pidm = pidm_var;

  cursor stu_cursor is
    select 'Student'
    from swbstdn
    where swbstdn_pidm = pidm_var;

BEGIN
  open emp_cursor;
  fetch emp_cursor into status_var;
  if emp_cursor%notfound then
    close emp_cursor;
    open stu_cursor;
    fetch stu_cursor into status_var;
    close stu_cursor;
  end if;
Exercise 54 (Continued)

    exception
    when no_data_found then
        status_var := 'No Status Found';
    END;

The call should be placed in the when-validate-item trigger before the exception.

Exercise 55

Create another form stvatyp based on the stvatyp table’s code and description, displaying 15 records and a scrollbar on a main canvas with Address Type Codes for the title.

This is based on previous exercises.

When the form is opened, it should automatically execute a query. Create a visual attribute group with a prominent background color to highlight the current record.

Form:  STVATYP
Trigger:  WHEN-NEW-FORM-INSTANCE
PL/SQL Code:

    execute_query;

Highlight visual attributes

Click the + icon. Rename it to CURRENT_RECORD, open up the properties and select a background color. At the block level, set the current record visual attribute group to CURRENT_RECORD.
Exercise 56
Which of the form invoke methods would be the most useful when you double-click the swraddr atyp code to go to the STVATYP form?

call_form

Change the code in the when mouse double click trigger to go to the STVATYP form. Using global variables in both forms enable the user to double-click on a code from STVATYP and return it to SWAIDEN. Populate the description without using a global variable and move the cursor to the next navigable item.

Form: SWAIDEN
Block: SWRADDR
Item: swraddr_atyp_code
Trigger: WHEN-MOUSE-DOUBLE-CLICK
PL/SQL Code:
--do_key('list_values');
call_form('stvatyp',hide,do_replace,query_only);
:swraddr_atyp_code := :global.value;
next_item;

Form: STVATYP
Block: STVATYP
Item: stvatyp_code
Trigger: WHEN-MOUSE-DOUBLE-CLICK
PL/SQL Code:
:global.value := :stvatyp_code;
exit_form;
This workbook was last updated on 12/06/2005.